

GS. PHẠM VĂN ẤT (Chủ biên)
LÊ TRƯỜNG THÔNG

Giáo trình

C++ **& Lập trình** **hướng** **đối tượng**



NHÀ XUẤT BẢN HỒNG ĐỨC

GS. PHẠM VĂN ẤT (Chủ biên) – LÊ TRƯỜNG THÔNG

GIÁO TRÌNH

C⁺⁺

&

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

NHÀ XUẤT BẢN HỒNG ĐỨC

LỜI NÓI ĐẦU

Lập trình cấu trúc là phương pháp tổ chức, phân chia chương trình thành các hàm, thủ tục. Chúng được dùng để xử lý dữ liệu nhưng lại tách rời các cấu trúc dữ liệu. Thông qua các ngôn ngữ Foxpro, Pascal, C đa số những người làm Tin học đã khá quen biết với phương pháp lập trình này.

Lập trình hướng đối tượng (object-oriented programming) dựa trên việc tổ chức chương trình thành các lớp (*class*). Khác với hàm và thủ tục, lớp là một đơn vị bao gồm cả dữ liệu và các phương thức xử lý. Vì vậy lớp có thể mô tả các thực thể một cách chân thực, đầy đủ cả phần dữ liệu và yêu cầu quản lý. Tư tưởng lập trình hướng đối tượng được áp dụng cho hầu hết các ngôn ngữ mới chạy trên môi trường Windows như Microsoft Access, Visual Basic, Visual C. Vì vậy việc nghiên cứu phương pháp lập trình mới này là rất cần thiết đối với tất cả những người quan tâm, yêu thích Tin học.

C⁽¹⁾ ra đời năm 1973 với mục đích ban đầu là để viết hệ điều hành Unix trên máy tính mini PDP. Sau đó C đã được sử dụng rộng rãi trên nhiều loại máy tính khác nhau và đã trở thành một ngôn ngữ lập trình cấu trúc rất được ưa chuộng trên toàn thế giới.

Để đưa C vào thế giới hướng đối tượng, năm 1980 nhà khoa học người Mỹ B. Stroustrup⁽²⁾ đã cho ra đời một ngôn ngữ C mới có tên ban đầu là "C có lớp", sau đó đến năm 1983 thì gọi là C⁺⁺. Ngôn ngữ C⁺⁺ là một sự phát triển mạnh mẽ của C. Trong C⁺⁺ chẳng những đưa vào tất cả các khái niệm, công cụ của lập trình hướng đối tượng mà còn đưa vào nhiều khả năng mới mẻ cho hàm. Như vậy C⁺⁺ là một ngôn ngữ lai cho phép tổ chức chương trình theo các lớp và các hàm. Có thể nói C⁺⁺ đã thúc đẩy ngôn ngữ C vốn đã rất thuyết phục đi vào thế giới lập trình hướng đối tượng và C⁺⁺ đã trở thành ngôn ngữ hướng đối tượng nổi bật trong những năm 90.

Cuốn sách này sẽ trình bày một cách hệ thống các khái niệm của lập trình hướng đối tượng được cài đặt trong C⁺⁺ như lớp, đối tượng (*object*), sự thừa kế (*inheritance*), tính tương ứng bội (*polymorphism*) và các khả năng mới trong xây dựng, sử dụng hàm như: đối tham chiếu, đối mặc định, hàm trùng tên, hàm toán tử. Có một số vấn đề còn ít được biết đến như cách xây dựng hàm với số đối bất định trong C cũng sẽ được giới thiệu. Các chương từ 1 đến 11 với cách giải thích tỉ mỉ và với gần 100 chương trình minh họa sẽ cung cấp cho bạn đọc các khái niệm, phương pháp và kinh nghiệm lập trình hướng đối tượng trên C⁺⁺. Có một phụ lục cuối sách (Phụ lục 6) sẽ hệ thống ngắn gọn phương pháp phân tích, thiết kế và lập trình hướng đối tượng trên bình diện chung.

Cuốn sách gồm 12 chương và 7 phụ lục.

Chương 1 hướng dẫn cách làm việc với phần mềm TC⁺⁺ 3.0 để thử nghiệm các chương trình, trình bày sơ lược về các phương pháp lập trình và giới thiệu một số mở rộng đơn giản của C⁺⁺.

Chương 2 trình bày các khả năng mới trong việc xây dựng và sử dụng hàm trong C⁺⁺ như biến tham chiếu, đối có kiểu tham chiếu, đối có giá trị mặc định, hàm trực tuyến, hàm trùng tên, hàm toán tử.

Chương 3 nói về một khái niệm trung tâm của lập trình hướng đối tượng là lớp gồm: Định nghĩa lớp, khai báo các biến, mảng đối tượng (kiểu lớp), phương thức, dùng con trỏ this trong phương thức, phạm vi truy xuất của các thành phần, các phương thức toán tử.

⁽¹⁾ Tác giả của C là nhà khoa học Dennis Ritchie.

⁽²⁾ Bjarne Stroustrup's và Dennis Ritchie đều làm việc ở Bell Lab (Mỹ) - nơi quy tụ rất nhiều nhà khoa học và nghiên cứu hàng đầu về lĩnh vực máy tính.

Chương 4 trình bày các vấn đề tạo dựng, sao chép, huỷ bỏ các đối tượng và các vấn đề khác có liên quan như: Hàm tạo, hàm tạo sao chép, hàm huỷ, toán tử gán, cấp phát bộ nhớ cho đối tượng, hàm bạn, lớp bạn.

Chương 5 trình bày một khái niệm quan trọng tạo nên khả năng mạnh của lập trình hướng đối tượng trong việc phát triển, mở rộng phần mềm, đó là khả năng thừa kế của các lớp.

Chương 6 trình bày một khái niệm quan trọng khác cho phép xử lý các vấn đề khác nhau, các thực thể khác nhau, các thuật toán khác nhau theo cùng một lược đồ thống nhất, đó là tính tương ứng bội và phương thức ảo. Các công cụ này cho phép dễ dàng tổ chức chương trình quản lý nhiều dạng đối tượng khác nhau.

Chương 7 trình bày các thao tác trên tệp như: tạo một tệp mới, ghi dữ liệu từ bộ nhớ lên tệp, đọc dữ liệu từ tệp vào bộ nhớ, ...

Chương 8 nói về việc tổ chức vào/ra trong C++. C++ đưa vào một khái niệm mới gọi là các dòng tin (Stream). Các thao tác vào/ra sẽ thực hiện trao đổi dữ liệu giữa bộ nhớ với dòng tin: Vào là chuyển dữ liệu từ dòng nhập vào bộ nhớ, ra là chuyển dữ liệu từ bộ nhớ lên dòng xuất. Để nhập xuất dữ liệu trên một thiết bị cụ thể nào, ta chỉ cần gán dòng nhập xuất với thiết bị đó. Việc tổ chức vào ra theo cách như vậy là rất khoa học và tiện lợi vì nó có tính độc lập thiết bị.

Chương 9 trình bày các hàm đồ họa sử dụng trong C và C++. Các hàm này được sử dụng rải rác trong toàn bộ cuốn sách để xây dựng các đối tượng đồ họa.

Chương 10 trình bày các hàm truy xuất trực tiếp vào bộ nhớ của máy tính, trong đó có bộ nhớ màn hình. Các hàm này sẽ được sử dụng trong chương 11 để xây dựng các lớp menu và cửa sổ.

Chương 11 giới thiệu 5 chương trình tương đối hoàn chỉnh nhằm minh họa thêm khả năng và kỹ thuật lập trình hướng đối tượng trên C++.

Chương 12 trình bày thêm một số chương trình hướng đối tượng trên C++. Đây là các chương trình tương đối phức tạp, hữu ích và sử dụng các công cụ mạnh của C++.

Phụ lục 1 trình bày các phép toán trong C++ và thứ tự ưu tiên của chúng.

Phụ lục 2 liệt kê một danh sách các từ khoá của C++.

Phụ lục 3 trình bày bảng mã ASCII và mã quét của các ký tự.

Phụ lục 4 trình bày một vấn đề quan trọng nhưng còn ít được nói đến trong các tài liệu, đó là cách sử dụng con trỏ void để xây dựng các hàm với số đối không cố định giống như các hàm printf và scanf của C.

Phụ lục 5 sẽ giới thiệu tóm tắt hơn 200 hàm để bạn đọc tiện việc tra cứu. (Vì trong C++ vẫn sử dụng các hàm của C).

Phụ lục 6 trình bày một cách ngắn gọn phương pháp phân tích, thiết kế và lập trình hướng đối tượng trên bình diện chung.

Phụ lục 7 trình bày một số chương trình minh họa thuật toán đệ quy kiểu quay lui.

Khi viết chúng tôi đã hết sức cố gắng để cuốn sách được hoàn chỉnh, song chắc chắn không tránh khỏi thiếu sót, vì vậy rất mong nhận được sự góp ý của độc giả.

Nhân dịp này chúng tôi xin chân thành cảm ơn Thạc sĩ Nguyễn Văn Phác đã tận tình giúp đỡ trong việc hiệu đính và biên tập cuốn sách này.

CÁC TÁC GIẢ

C++ & LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Trong chương này trình bày các vấn đề sau:

- Cách sử dụng phần mềm TC++ 3.0
- Những sửa đổi cần thiết một chương trình C để biến nó thành một chương trình C++ (chạy được trong môi trường C++)
- Tóm lược về các phương pháp lập trình cấu trúc và lập trình hướng đối tượng
- Những mở rộng của C++ so với C

§1. TURBO C++ 3.0

Các ví dụ trong cuốn sách này sẽ viết và thực hiện trên môi trường TC++ 3.0. Bộ cài đặt TC++ 3.0 gồm 5 đĩa. Sau khi cài đặt (giả sử vào thư mục C:\TC) thì trong thư mục TC sẽ gồm các thư mục con sau:

C:\TC\BGI chứa các tệp đuôi BGI và CHR

C:\TC\BIN chứa các tệp chương trình (đuôi EXE) như TC, TCC, TLIB, TLINK

C:\TC\INCLUDE chứa các tệp tiêu đề đuôi H

C:\TC\LIB chứa các tệp đuôi LIB, OBJ

Để vào môi trường của TC++ chỉ cần thực hiện tệp chương trình TC trong thư mục C:\TC\BIN. Kết quả nhận được hệ menu chính của TC++ với màu nền xanh gần giống như hệ menu quen thuộc của TC (Turbo C). Hệ menu của TC++ gồm các menu: File, Edit, Search, Run, Compile, Debug, Project, Options, Window, Help.

Cách soạn thảo, biên dịch và chạy chương trình trong TC++ cũng giống như trong TC, ngoại trừ điểm sau: Tệp chương trình trong hệ soạn thảo của TC++ có đuôi mặc định là CPP còn trong TC thì tệp chương trình luôn có đuôi C.

Trong TC++ có thể thực hiện cả chương trình C và C++. Để thực hiện chương trình C cần dùng đuôi C để đặt tên cho tệp chương trình, để thực hiện chương trình C++ cần dùng đuôi CPP để đặt tên cho tệp chương trình.

§2. C++ VÀ C

- Có thể nói C++ là sự mở rộng (đáng kể) của C. Điều đó có nghĩa là mọi khả năng, mọi khái niệm trong C đều dùng được trong C++.

- Vì trong C++ sử dụng gần như toàn bộ các khái niệm, định nghĩa, các kiểu dữ liệu, các cấu trúc lệnh, các hàm và các công cụ khác của C, nên yêu cầu bắt buộc đối với các đọc giả C++ là phải biết sử dụng tương đối thành thạo ngôn ngữ C.

- Vì C++ là sự mở rộng của C, nên bản thân một chương trình C đã là chương trình C++ (chỉ cần thay đuôi C bằng đuôi CPP). Tuy nhiên Trình biên dịch TC++ yêu cầu mọi hàm chuẩn dùng trong chương trình đều phải khai báo nguyên mẫu bằng một câu lệnh `#include`, trong khi điều này không bắt buộc đối với Trình biên dịch của TC.

Trong C có thể dùng một hàm chuẩn mà bỏ qua câu lệnh `#include` để khai báo nguyên mẫu của hàm được dùng. Điều này không báo lỗi khi biên dịch, nhưng có thể dẫn đến kết quả sai khi chạy chương trình.

Ví dụ khi biên dịch chương trình sau trong môi trường C sẽ không gặp các dòng cảnh báo (warning) và thông báo lỗi (error). Nhưng khi chạy sẽ nhận được kết quả sai.

```
#include <stdio.h>
void main()
{
    float a, b, c, p, s;
    printf("\nNhập a, b, c: ");
    scanf("%f%f%f", &a, &b, &c);
    p = (a + b + c) / 2;
    s = sqrt(p * (p - a) * (p - b) * (p - c));
    printf("\nDiện tích = %0.2f", s);
    getch();
}
```

Nếu biên dịch chương trình này trong TC++ sẽ nhận được các thông báo lỗi sau:

Error: Function 'sqrt' should have a prototype

Error: Function 'getch' should have a prototype

Để biến chương trình trên thành một chương trình C++ cần:

+ Đặt tên chương trình với đuôi CPP

+ Thêm 2 câu lệnh `#include` để khai báo nguyên mẫu cho các hàm `sqrt`, `getch`:

```
#include <math.h>
#include <conio.h>
```

§3. LẬP TRÌNH CẤU TRÚC & LẬP TRÌNH ĐỐI TƯỢNG

3.1. Phương pháp lập trình cấu trúc

- Tư tưởng chính của lập trình cấu trúc là tổ chức chương trình thành các chương trình con. Trong PASCAL có 2 kiểu chương trình con là thủ tục và hàm. Trong C chỉ có một loại chương trình con là hàm.

Hàm là một đơn vị chương trình độc lập dùng để thực hiện một phần việc nào đó như: Nhập số liệu, in kết quả hay thực hiện một số tính toán. Hàm cần có đối và các biến, mảng cục bộ dùng riêng cho hàm.

Việc trao đổi dữ liệu giữa các hàm thực hiện thông qua các đối và các biến toàn bộ.

Các ngôn ngữ như C, PASCAL, FORTRAN là các ngôn ngữ cho phép triển khai phương pháp lập trình cấu trúc.

Một chương trình cấu trúc gồm các cấu trúc dữ liệu (như biến, mảng, bản ghi) và các hàm, thủ tục.

Nhiệm vụ chính của việc tổ chức thiết kế chương trình cấu trúc là tổ chức chương trình thành các hàm, thủ tục: Chương trình sẽ bao gồm các hàm, thủ tục nào.

Ví dụ xét yêu cầu sau: Viết chương trình nhập tọa độ (x, y) của một dãy điểm, sau đó tìm một cặp điểm cách xa nhau nhất.

Trên tư tưởng của lập trình cấu trúc có thể tổ chức chương trình như sau:

+ Sử dụng 2 mảng thực toàn bộ x và y để chứa tọa độ dãy điểm.

+ Xây dựng 2 hàm:

Hàm nhapsl dùng để nhập tọa độ n điểm, hàm này có một đối là biến nguyên n và được khai báo như sau:

```
void nhapsl(int n);
```

Hàm do_dai dùng để tính độ dài đoạn thẳng đi qua 2 điểm có chỉ số là i và j, nó được khai báo như sau:

```
float do_dai(int i, int j);
```

Chương trình C cho bài toán trên được viết như sau:

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
float x[100], y[100];
float do_dai(int i, int j)
{
    return sqrt(pow(x[i] - x[j], 2) + pow(y[i] - y[j], 2));
}
```

```

void nhapsl(int n)
{
    int i;
    for (i=1; i<=n; ++i)
    {
        printf("\nNhap toa do x, y cua diem thu %d: ", i);
        scanf("%f%f", &x[i], &y[i]);
    }
}

void main()
{
    int n, i, j, imax, jmax;
    float d, dmax;
    printf("\nSo diem N = ");
    scanf("%d", &n);
    nhapsl(n);
    dmax = do_dai(1, 2); imax = 1; jmax = 2;
    for (i=1; i<=n-1; ++i)
        for (j=i+1; j<=n; ++j)
        {
            d = do_dai(i, j);
            if (d > dmax)
            {
                dmax = d;
                imax = i;
                jmax = j;
            }
        }
    printf("\nDoan thang lon nhat co do dai = %0.2f", dmax);
    printf("\nDi qua 2 diem co chi so la %d, %d", imax, jmax);
    getch();
}

```

3.2. Phương pháp lập trình hướng đối tượng

+ Khái niệm trung tâm của lập trình hướng đối tượng là lớp (class). Có thể xem lớp là sự kết hợp các thành phần dữ liệu và các hàm. Cũng có thể xem lớp là sự mở rộng của cấu trúc trong C (struct) bằng cách đưa thêm vào các phương thức (method) hay còn gọi là hàm thành viên (member function). Một lớp được định nghĩa như sau:

```

class Tên_lớp
{
    // Khai báo các thành phần dữ liệu
    // Khai báo các phương thức
};

```

+ Các phương thức có thể được viết (xây dựng) bên trong hoặc bên

ngoài (phía dưới) phần định nghĩa lớp. Cấu trúc (cách viết) phương thức tương tự như hàm ngoại trừ quy tắc sau: Khi xây dựng một phương thức bên ngoài định nghĩa lớp thì trong dòng đầu tiên cần dùng tên lớp và 2 dấu : đặt trước tên phương thức để chỉ rõ phương thức thuộc lớp nào (xem ví dụ bên dưới).

+ Sử dụng các thành phần dữ liệu trong phương thức: Vì phương thức và các thành phần dữ liệu thuộc cùng một lớp và vì phương thức được lập lên cốt để xử lý các thành phần dữ liệu, nên trong thân của phương thức có quyền truy nhập đến các thành phần dữ liệu (của cùng lớp).

+ Biến lớp: Sau khi định nghĩa một lớp, có thể dùng tên lớp để khai báo các biến kiểu lớp hay còn gọi là đối tượng. Mỗi đối tượng sẽ có các thành phần dữ liệu và các phương thức. Lời gọi một phương thức cần chứa tên đối tượng để xác định phương thức thực hiện từ đối tượng nào.

+ Một chương trình hướng đối tượng sẽ bao gồm các lớp có quan hệ với nhau.

+ Việc phân tích, thiết kế chương trình theo phương pháp hướng đối tượng nhằm thiết kế, xây dựng các lớp.

+ Từ khái niệm lớp nảy sinh hàng loạt khái niệm khác như: Thành phần dữ liệu, phương thức, phạm vi, sự đóng gói, hàm tạo, hàm huỷ, sự thừa kế, lớp cơ sở, lớp dẫn xuất, tương ứng bội, phương thức ảo, ...

+ Ưu điểm của việc thiết kế hướng đối tượng là tập trung xác định các lớp để mô tả các thực thể của bài toán. Mỗi lớp đưa vào các thành phần dữ liệu của thực thể và xây dựng luôn các phương thức để xử lý dữ liệu. Như vậy việc thiết kế chương trình xuất phát từ các nội dung, các vấn đề của bài toán.

+ Các ngôn ngữ thuần túy hướng đối tượng (như Smalltalk) chỉ hỗ trợ các khái niệm về lớp, không có các khái niệm hàm.

+ C++ là ngôn ngữ lai, nó cho phép sử dụng cả các công cụ của lớp và hàm.

Để minh họa các khái niệm vừa nêu về lập trình hướng đối tượng ta trở lại xét bài toán tìm độ dài lớn nhất đi qua 2 điểm. Trong bài toán này ta gặp một thực thể là dãy điểm. Các thành phần dữ liệu của lớp dãy điểm gồm:

- Biến nguyên n là số điểm của dãy
- Con trỏ x kiểu thực trỏ đến vùng nhớ chứa dãy hoành độ
- Con trỏ y kiểu thực trỏ đến vùng nhớ chứa dãy tung độ

Các phương thức cần đưa vào theo yêu cầu bài toán gồm:

- Nhập tọa độ một điểm
- Tính độ dài đoạn thẳng đi qua 2 điểm

Dưới đây là chương trình viết theo thiết kế hướng đối tượng. Để thực hiện chương trình này nhớ đặt tên tệp có đuôi CPP. Xem chương trình ta thấy thêm một điều mới trong C++ là:

Các khai báo biến, mảng có thể viết bất kỳ chỗ nào trong chương trình (tất nhiên phải trước khi sử dụng biến, mảng).

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <alloc.h>
class daydiem
{
public:
    int n;
    float *x, *y;
    float do dai(int i, int j)
    {
        return sqrt(pow(x[i] - x[j], 2) +
                      pow(y[i] - y[j], 2));
    }
    void nhapsl(void);
};
void daydiem::nhapsl(void)
{
    int i;
    printf("\nSo diem N = ");
    scanf("%d", &n);
    x = (float*)malloc((n+1)*sizeof(float));
    y = (float*)malloc((n+1)*sizeof(float));
    for (i=1; i<=n; ++i)
    {
        printf("\nNhap toa do x, y cua diem thu %d: ", i);
        scanf("%f%f", &x[i], &y[i]);
    }
}
void main()
{
    daydiem p;
    p.nhapsl();
    int n, i, j, imax, jmax;
    float d, dmax;
    n = p.n;
```

```

dmax = p.do_dai(1, 2); imax = 1; jmax = 2;
for (i=1; i<=n-1; ++i)
    for (j=i+1; j<=n; ++j)
    {
        d = p.do_dai(i, j);
        if (d > dmax)
        {
            dmax = d;
            imax = i;
            jmax = j;
        }
    }
printf("\nDoan thang lon nhat co do dai = %0.2f", dmax);
printf("\nDi qua 2 diem co chi so la %d, %d", imax, jmax);
getch();
}

```

§4. MỘT SỐ MỞ RỘNG ĐƠN GIẢN CỦA C⁺⁺ SO VỚI C

Trong mục này trình bày một số mở rộng của C⁺⁺, tuy đơn giản, ngắn gọn nhưng đem lại rất nhiều tiện lợi.

4.1. Viết các dòng ghi chú

Trong C⁺⁺ vẫn có thể viết các dòng ghi chú trong các dấu `/*` và `*/` như trong C. Cách này cho phép viết các ghi chú trên nhiều dòng hoặc trên một dòng. Ngoài ra trong C⁺⁺ còn cho phép viết ghi chú trên một dòng sau 2 dấu gạch chéo, ví dụ:

```
int x, y; // Khai báo 2 biến thực
```

4.2. Khai báo linh hoạt

Trong C tất cả các câu lệnh khai báo biến, mảng cục bộ phải đặt tại đầu khối. Do vậy nhiều khi, vị trí khai báo và vị trí sử dụng của biến khá xa nhau, gây khó khăn trong việc kiểm soát chương trình. C⁺⁺ đã khắc phục nhược điểm này bằng cách cho phép các lệnh khai báo biến, mảng có thể đặt bất kỳ chỗ nào trong chương trình trước khi các biến, mảng được sử dụng. Ví dụ chương trình nhập một dãy số thực rồi sắp xếp theo thứ tự tăng dần có thể viết trong C⁺⁺ như sau:

```

#include <stdio.h>
#include <conio.h>
#include <alloc.h>
void main()
{
    int n;

```

```

printf("\nSố phần tử của dãy N = ");
scanf("%d", &n);
float *x = (float*)malloc((n+1)*sizeof(float));
for (int i = 1; i <= n; ++i)
{
    printf("\nX[%d] = ", i);
    scanf("%f", &x[i]);
}
for (i = 1; i <= n-1; ++i)
    for (int j = i+1; j <= n; ++j)
        if (x[i] > x[j])
        {
            float tg = x[i];
            x[i] = x[j];
            x[j] = tg;
        }
printf("\nDãy sau khi sắp xếp\n");
for (i = 1; i <= n; ++i)
    printf("%0.2f", x[i]);
getch();
}

```

4.3. Toán tử ép kiểu

Toán tử này được viết trong C như sau:

(kiểu)biểu thức

Trong C++ vẫn có thể dùng cách viết này. Ngoài ra C++ cho phép viết một cách khác tiện lợi hơn như sau:

kiểu(biểu thức)

Ví dụ chương trình tính công thức:

$$S = 2/1 + 3/2 + \dots + (n+1)/n$$

với n là một số nguyên dương nhập từ bàn phím, có thể viết như sau:

```

#include <stdio.h>
#include <conio.h>
void main()
{
    int n;
    printf("\nSố phần tử của dãy N = ");
    scanf("%d", &n);
    float s = 0.0;
    for (int i = 1; i <= n; ++i)
        s += float(i+1)/float(i); // Ép kiểu theo C++
    printf("S = %0.2f", s);
    getch();
}

```

4.4. Hằng có kiểu

Để tạo ra một hằng có kiểu, ta sử dụng từ khoá const đặt trước một khai báo có khởi gán giá trị. Sau đây là một số ví dụ.

+ Hằng nguyên:

```
const int maxsize = 1000;  
int a[maxsize];
```

+ Cấu trúc hằng:

```
typedef struct  
{  
    int x, y; // Tọa độ của điểm  
    int mau; // Mã màu của điểm  
} DIEM;  
const DIEM d = {320, 240, 15};
```

Chương trình dưới đây minh họa cách dùng hằng có kiểu. Chương trình tạo một cấu trúc hằng (kiểu DIEM) mô tả điểm giữa màn hình đồ họa với màu trắng. Điểm này được hiển thị trên màn hình đồ họa.

```
#include <stdio.h>  
#include <conio.h>  
#include <graphics.h>  
#include <stdlib.h>  
typedef struct  
{  
    int x, y;  
    int mau;  
} DIEM;  
void main()  
{  
    int mh = 0, mode = 0;  
    initgraph(&mh, &mode, "");  
    int loi = graphresult();  
    if (loi)  
    {  
        printf("\nLỗi do họa: %s", grapherrormsg(loi));  
        getch(); exit(0);  
    }  
    const DIEM qmh = {getmaxx()/2, getmaxy()/2, WHITE};  
    putpixel(qmh.x, qmh.y, qmh.mau);  
    getch();  
    closegraph();  
}
```

Chú ý:

a. Có thể dùng các hàm để gán giá trị cho các hằng có kiểu (trong chương trình trên dùng các hàm getmaxx và getmaxy).

b. Mọi câu lệnh nhằm thay đổi giá trị hằng có kiểu đều bị báo lỗi khi biên dịch chương trình. Ví dụ nếu trong chương trình đưa vào câu lệnh:

```
gmh.x = 200;
```

thì khi dịch chương trình sẽ nhận được thông báo lỗi như sau:

```
Cannot modify a const object
```

4.5. Các kiểu char và int

Trong C một hằng ký tự được xem là nguyên do đó nó có kích thước 2 byte, ví dụ trong C:

```
sizeof('A') = sizeof(int) = 2
```

Còn trong C++ một hằng ký tự được xem là giá trị kiểu char và có kích thước một byte. Như vậy trong C++ thì:

```
sizeof('A') = sizeof(char) = 1
```

4.6. Lấy địa chỉ các phần tử mảng thực 2 chiều

Trong TC 2.0 không cho phép dùng phép & để lấy địa chỉ các phần tử mảng thực 2 chiều. Vì vậy khi nhập một ma trận thực (dùng scanf) ta phải nhập qua một biến trung gian sau đó mới gán cho các phần tử mảng.

Trong TC ++ 3.0 cho phép lấy địa chỉ các phần tử mảng thực 2 chiều, do đó có thể dùng scanf để nhập trực tiếp vào các phần tử mảng

Chương trình C++ dưới đây sẽ minh họa điều này. Chương trình nhập một ma trận thực cấp mxn và xác định phần tử có giá trị lớn nhất.

```
#include <conio.h>
#include <stdio.h>
void main()
{
    float a[20][20], smax;
    int m, n, i, j, imax, jmax;
    clrscr();
    puts("Cho biết số hàng và số cột của ma trận: ");
    scanf("%d%d", &m, &n);
    for (i=1; i<=m; ++i)
        for (j=1; j<=n; ++j)
        {
            printf("\na[%d][%d] = ", i, j);
            scanf("%f", &a[i][j]); // Lấy địa chỉ phần tử mảng
                                   // thực 2 chiều
        }
    smax = a[1][1]; imax=1; jmax=1;
    for (i=1; i<=m; ++i)
        for (j=1; j<=n; ++j)
            if (smax<a[i][j])
            {
```

```

        smax = a[i][j];
        imax = i;
        jmax = j;
    }
    puts( "\n\nMa tran");
    for (i=1; i<=m; ++i)
        for (j=1; j<=n; ++j)
        {
            if (j==1) puts("");
            printf("%6.1f", a[i][j]);
        }
    puts( "\n\nPhan tu max: ");
    printf("\ncò gia tri = %6.1f", smax);
    printf("\nTai hang %d cot %d ", imax, jmax);
    getch();
}

```

§5. VÀO RA TRONG C++

5.1. Các toán tử và phương thức xuất nhập

Để in dữ liệu ra màn hình và nhập dữ liệu từ bàn phím, trong C++ vẫn có thể dùng các hàm printf và scanf (như chỉ ra trong các chương trình C++ ở các mục trên).

Ngoài ra trong C++ còn dùng toán tử xuất:

```
cout << biểu thức << ... << biểu thức;
```

để đưa giá trị các biểu thức ra màn hình, dùng toán tử nhập:

```
cin >> biến >> ... >> biến;
```

để nhập các giá trị số (nguyên thực) từ bàn phím và gán cho các biến.

Để nhập một dãy không quá n ký tự và chứa vào mảng h (kiểu char) có thể dùng phương thức cin.get như sau:

```
cin.get(h, n);
```

Chú ý 1: Toán tử nhập cin >> sẽ để lại ký tự chuyển dòng '\n' trong bộ đệm, ký tự này có thể làm trôi phương thức cin.get. Để khắc phục tình trạng trên cần dùng phương thức cin.ignore để bỏ qua một ký tự chuyển dòng như sau:

```
cin.ignore(1);
```

Chú ý 2: Để sử dụng các toán tử và phương thức nói trên cần khai báo tệp tiêu đề:

```
#include <iostream.h>
```

Chương trình sau minh họa việc sử dụng các công cụ vào ra mới của C++ để nhập một danh sách n thí sinh. Dữ liệu mỗi thí sinh gồm họ

tên, các điểm toán, lý, hoá. Sau đó in danh sách thí sinh theo thứ tự giảm của tổng điểm.

```
#include <iostream.h>
#include <conio.h>
void main()
{
    struct
    {
        char ht[25];
        float t, l, h, td;
    } ts[50], tg;
    int n, i, j;
    clrscr();
    cout << "So thi sinh: ";
    cin >> n;
    for (i=1; i<=n; ++i)
    {
        cout << "\nThi sinh " << i;
        cout << "\nHo ten: ";
        cin.ignore(1);
        cin.get(ts[i].ht, 25);
        cout << "Cac diem toan, ly, hoa: ";
        cin >> ts[i].t >> ts[i].l >> ts[i].h;
        ts[i].td = ts[i].t + ts[i].l + ts[i].h;
    }
    for (i=1; i<=n-1; ++i)
        for (j=i+1; j<=n; ++j)
            if (ts[i].td < ts[j].td)
            {
                tg = ts[i];
                ts[i] = ts[j];
                ts[j] = tg;
            }
    cout << "\nDanh sach thi sinh sau khi sap xep ";
    for (i=1; i<=n; ++i)
    {
        cout << "\nHo ten: " << ts[i].ht;
        cout << "Tong diem: " << ts[i].td;
    }
    getch();
}
```

5.2. Định dạng khi in ra màn hình

+ Để quy định số thực (float, double) được in ra có đúng p chữ số sau dấu chấm thập phân, ta sử dụng đồng thời các hàm sau:

```
setiosflags(ios::showpoint); //Bật cờ hiệu showpoint
setprecision(p);
```

Các hàm này cần đặt trong toán tử xuất như sau:

```
cout << setiosflags(ios::showpoint) << setprecision(p);
```

Câu lệnh trên sẽ có hiệu lực đối với tất cả các toán tử xuất tiếp theo cho đến khi gặp một câu lệnh định dạng mới.

+ Để quy định độ rộng tối thiểu là w vị trí cho giá trị (nguyên, thực, chuỗi) được in trong các toán tử xuất, ta dùng hàm:

```
setw(w)
```

Hàm này cần đặt trong toán tử xuất và nó chỉ có hiệu lực cho một giá trị được in gần nhất. Các giá trị in ra tiếp theo sẽ có độ rộng tối thiểu mặc định là 0. Như vậy câu lệnh:

```
cout << setw(3) << "AB" << "CD"
```

Sẽ in ra 5 ký tự là: một dấu cách và 4 chữ cái A, B, C và D.

Chú ý: Muốn sử dụng các hàm trên cần đưa vào câu lệnh `#include` sau:

```
#include <iomanip.h>
```

Trở lại chương trình trên ta thấy danh sách thí sinh in ra sẽ không thẳng cột. Để khắc phục điều này cần viết lại đoạn chương trình in như sau:

```
cout << "\nDanh sach thi sinh sau khi sap xep " ;
cout << setiosflags(ios::showpoint) << setprecision(1) ;
for (i = 1; i <= n; i++)
{
    cout << "\nHo ten: " << setw(25) << ts[i].ht;
    cout << "Tong diem: " << setw(5) << ts[i].td;
}
getch();
```

Chương trình dưới đây là một minh họa khác về việc sử dụng các toán tử nhập xuất và cách định dạng trong C⁺⁺. Chương trình nhập một ma trận thực cấp $m \times n$. Sau đó in ma trận dưới dạng bảng và tìm một phần tử lớn nhất.

```
#include <iostream.h>
#include <iomanip.h>
#include <conio.h>
void main()
{
    float a[20][20], smax;
    int m, n, i, j, imax, jmax;
    clrscr();
    cout << "Cho biet so hang va so cot cua ma tran: ";
    cin >> m >> n;
    for (i=1; i < m; i++)
```

```

    for (j=1; j<=n; ++j)
    {
        cout << "a[" << i << ", " << j << "] = ";
        cin >> a[i][j];
    }
    smax = a[1][1];
    imax = 1;
    jmax = 1;
    for (i=1; i<=m; ++i)
        for (j=1; j<=n; ++j)
            if (smax < a[i][j])
            {
                smax = a[i][j];
                imax = i;
                jmax = j;
            }
    cout << "\n\nMa tran";
    cout << setiosflags(ios::showpoint) << setprecision(1);
    for (i=1; i<=m; ++i)
        for (j=1; j<=n; ++j)
        {
            if (j==1) cout << '\n';
            cout << setw(6) << a[i][j];
        }
    cout << "\n\n" << "Phan tu max:" << '\n';
    cout << "co gia tri = " << setw(6) << smax;
    cout << "\nTai hang " << imax << " cot " << jmax;
    getch();
}

```

§6. CẤU TRÚC, HỢP VÀ KIỂU LIỆT KÊ

6.1. Tên sau từ khoá struct được xem như tên kiểu cấu trúc

Trong C++ một kiểu cấu trúc cũng được định nghĩa như C theo mẫu:

```

struct Tên_kiểu_ct
{
    // Khai báo các thành phần của cấu trúc
};

```

Sau đó để khai báo các biến, mảng cấu trúc, trong C dùng mẫu sau:

```

struct Tên_kiểu_ct danh_sách_biến, mảng_cấu_trúc;

```

Như vậy trong C, tên viết sau từ khoá struct chưa phải là tên kiểu và chưa có thể dùng để khai báo.

Trong C++ xem tên viết sau từ khoá struct là tên kiểu cấu trúc và có thể dùng nó để khai báo. Như vậy để khai báo các biến, mảng cấu trúc trong C++, ta có thể dùng mẫu sau:

Tên_kiểu ct danh sách biến, mảng cấu trúc;

Ví dụ sau sẽ: Định nghĩa kiểu cấu trúc TS (thí sinh) gồm các thành phần: ht (họ tên), sobd (số báo danh), dt (điểm toán), dl (điểm lý), dh (điểm hoá) và td (tổng điểm), sau đó khai báo biến cấu trúc h và mảng cấu trúc ts.

```
struct TS
{
    char ht [25];
    long sobd;
    float dt, dl, dh, td;
};
TS h, ts[1000];
```

6.2. Tên sau từ khoá union được xem như tên kiểu hợp

Trong C⁺⁺ một kiểu hợp (union) cũng được định nghĩa như C theo mẫu:

```
union Tên_kiểu_hợp
{
    // Khai báo các thành phần của hợp
};
```

Sau đó để khai báo các biến, mảng kiểu hợp, trong C dùng mẫu sau:
union Tên_kiểu_hợp danh_sách_biến, mảng_kiểu_hợp;

Như vậy trong C, tên viết sau từ khoá union chưa phải là tên kiểu và chưa có thể dùng để khai báo.

Trong C⁺⁺ xem tên viết sau từ khoá union là tên kiểu hợp và có thể dùng nó để khai báo. Như vậy để khai báo các biến, mảng kiểu hợp, trong C⁺⁺ có thể dùng mẫu sau:

```
Tên_Kiểu_hợp danh_sách_biến, mảng_kiểu_hợp;
```

6.3. Các union không tên

Trong C⁺⁺ cho phép dùng các union không tên dạng:

```
union
{
    // Khai báo các thành phần
};
```

Khi đó các thành phần (khai báo trong union) sẽ dùng chung một vùng nhớ. Điều này cho phép tiết kiệm bộ nhớ và cho phép dễ dàng tách các byte của một vùng nhớ.

Ví dụ nếu các biến nguyên i, biến ký tự ch và biến thực x không đồng thời sử dụng thì có thể khai báo chúng trong một union không tên như sau:

```
union
{
    int i;
```

```
char ch;
float x;
};
```

Khi đó các biến `i`, `ch` và `f` sử dụng chung một vùng nhớ 4 byte.

Xét ví dụ khác, để tách các byte của một biến `unsigned long` ta dùng `union` không tên sau:

```
union
{
    unsigned long u;
    unsigned char b[4];
};
```

Khi đó nếu gán:

```
u = 0xDDCCBBAA; // Số hệ 16
```

thì:

```
b[0] = 0xAA
b[1] = 0xBB
b[2] = 0xCC
b[3] = 0xDD
```

6.4. Kiểu liệt kê (enum)

+ Cũng giống như cấu trúc và hợp, tên viết sau từ khoá `enum` được xem là kiểu liệt kê và có thể dùng để khai báo, ví dụ:

```
enum MAU { xanh, do, tim, vang }; // Định nghĩa kiểu MAU
MAU m, dsu[10]; // Khai báo các biến, mảng kiểu MAU
```

+ Các giá trị kiểu liệt kê (enum) là các số nguyên. Do đó có thể thực hiện các phép tính trên các giá trị enum, có thể in các giá trị enum, có thể gán giá trị enum cho biến nguyên, ví dụ:

```
MAU m1, m2;
int n1, n2;
m1 = tim;
m2 = vang;
n1 = m1; // n1 = 2
n2 = m1 + m2; // n2 = 5
printf("\n %d ", m2 ); // in ra số 3
```

+ Không thể gán trực tiếp một giá trị nguyên cho một biến enum mà phải dùng phép ép kiểu, ví dụ:

```
m1 = 2; // lỗi
m1 = MAU(2); // đúng
```

§7. CẤP PHÁT BỘ NHỚ

7.1. Trong C++ có thể sử dụng các hàm cấp phát bộ nhớ động của C như: hàm *malloc* để cấp phát bộ nhớ, hàm *free* để giải phóng bộ nhớ được cấp phát.

7.2. Ngoài ra trong C++ còn đưa thêm toán tử *new* để cấp phát bộ nhớ và toán tử *delete* để giải phóng bộ nhớ được cấp phát bởi *new*.

7.3. Cách dùng toán tử *new* để cấp phát bộ nhớ như sau:

+ Trước hết cần khai báo một con trỏ để chứa địa chỉ vùng nhớ sẽ được cấp phát:

Kiểu *p;

ở đây Kiểu có thể là:

- các kiểu dữ liệu chuẩn của C++ như int, long, float, double, char, ...

- các kiểu do lập trình viên định nghĩa như: mảng, hợp, cấu trúc, lớp, ...

+ Sau đó dùng toán tử *new* theo mẫu:

p = new Kiểu; // Cấp phát bộ nhớ cho một biến (một phần tử)

p = new Kiểu[n]; // Cấp phát bộ nhớ cho n phần tử

Ví dụ để cấp phát bộ nhớ cho một biến thực ta dùng câu lệnh sau:

```
float *px = new float;
```

Để cấp phát bộ nhớ cho 100 phần tử nguyên ta dùng các câu lệnh:

```
int *pn = new int[100];
```

```
for (int i 0; i<100 ; ++i)
```

```
    pn[i] = 20 * i; // Gán cho phần tử thứ i
```

7.4. Hai cách kiểm tra sự thành công của *new*

Khi dùng câu lệnh:

```
Kiểu *p = new Kiểu[n];
```

hoặc câu lệnh:

```
Kiểu *p = new Kiểu;
```

để cấp phát bộ nhớ sẽ xuất hiện một trong 2 trường hợp: thành công hoặc không thành công.

Nếu thành công thì p sẽ chứa địa chỉ đầu vùng nhớ được cấp phát.

Nếu không thành công thì p = NULL.

Đoạn chương trình sau minh họa cách kiểm tra lỗi cấp phát bộ nhớ:

```
double *pd;
```

```
int n;
```

```
cout << "\n Số phần tử : ";
```

```
cin >> n;
```

```
pd = new double[n];
```

```
if (pd==NULL)
```

```
{
```

```
    cout << "Lỗi cấp phát bộ nhớ"
```

```
    exit (0);
```

```
}
```

Cách thứ 2 để kiểm tra sự thành công của toán tử new là dùng con trỏ hàm:

`_new_handler`

được định nghĩa trong tệp “new.h”. Khi gặp lỗi trong toán tử new (cấp phát không thành công) thì chương trình sẽ thực hiện một hàm nào đó do con trỏ `_new_handler` trỏ tới. Cách dùng con trỏ này như sau:

- + Xây dựng một hàm dùng để kiểm tra sự thành công của new
- + Gán tên hàm này cho con trỏ `_new_handler`

Như vậy hàm kiểm tra sẽ được gọi mỗi khi có lỗi xảy ra trong toán tử new.

Đoạn chương trình kiểm tra theo cách thứ nhất có thể viết theo cách thứ hai như sau:

```
void kiem_tra_new(void) // Lập hàm kiểm tra
{
    cout << " Lỗi cấp phát bộ nhớ "
    exit (0);
}
```

```
_new_handler = kiem_tra_new // Gán tên hàm cho con trỏ
// double *pd;
```

```
int n;
```

```
cout << "\n Số phần tử : ";
```

```
cin >> n;
```

```
pd = new double[n]; // Khi xảy ra lỗi sẽ gọi hàm kiem_tra_new
```

Chú ý: Có thể dùng lệnh gán để gán tên hàm xử lý lỗi cho con trỏ `_new_handler` như trong đoạn chương trình trên, hoặc dùng hàm:

```
set_new_handler(Tên hàm);
```

(Xem các chương trình minh họa bên dưới)

7.5. Toán tử delete dùng để giải phóng vùng nhớ được cấp phát bởi new

Cách dùng như sau:

```
delete p; // p là con trỏ dùng trong new
```

Ví dụ:

```
float *px;
```

```
px = new float[2000]; //Cấp phát bộ nhớ cho 2000 phần tử thực
// Sử dụng bộ nhớ được cấp phát
```

```
delete px ; // Giải phóng bộ nhớ
```

7.6. Hai chương trình minh họa

Chương trình thứ nhất minh họa cách dùng new để cấp phát bộ nhớ chứa n thí sinh. Mỗi thí sinh là một cấu trúc gồm các trường ht (họ tên), sobd (số báo danh) và td (tổng điểm). Chương trình sẽ nhập n, cấp phát bộ

nhớ chứa n thí sinh, kiểm tra lỗi cấp phát bộ nhớ (dùng cách 1), nhập n thí sinh, sắp xếp thí sinh theo thứ tự giảm của tổng điểm, in danh sách thí sinh sau khi sắp xếp, và cuối cùng là giải phóng bộ nhớ đã cấp phát.

```
#include <iomanip.h>
#include <iostream.h>
#include <stdlib.h>
#include <conio.h>
struct TS
{
    char ht[20];
    long sobd;
    float td;
};
void main(void)
{
    TS *ts;
    int n;
    cout << "\n So thi sinh n = ";
    cin >> n;
    ts = new TS[n+1];
    if (ts==NULL)
    {
        cout << "\nLoi cap phat bo nho ";
        getch();
        exit(0);
    }
    for (int i=1; i<=n; ++i)
    {
        cout << "\nThi sinh thu " << i;
        cout << "\nHo ten: ";
        cin.ignore(1) ;
        cin.get(ts[i].ht,20);
        cout << "So bao danh: ";
        cin >> ts[i].sobd;
        cout << "Tong diem: ";
        cin >> ts[i].td;
    }
    for (i=1; i<=n-1; ++i)
        for (int j=i+1; j<=n; ++j)
            if (ts[i].td < ts[j].td)
            {
                TS tg = ts[i];
                ts[i] = ts[j];
                ts[j] = tg;
            }
    cout << setiosflags(ios::showpoint) << setprecision(1);
    for (i=1; i<=n; ++i)
        cout << "\n" << setw(20) << ts[i].ht <<
```

```

        setw(6)<< ts[i].sobd <<setw(6)<< ts[i].td;
delete ts;
getch();
}

```

Chương trình thứ hai minh họa cách dùng con trỏ `_new_handler` để kiểm tra sự thành công của toán tử `new`. Chương trình sẽ cấp phát bộ nhớ cho một mảng con trỏ và sẽ theo dõi khi nào thì không đủ bộ nhớ để cấp phát.

```

#include <new.h>
#include <iostream.h>
#include <stdlib.h>
#include <conio.h>
int k;
void loi_bo_nho(void)
{
    cout << "\nloi bo nho khi cap phat
           bo nho cho q[" << k << "]\n";
    getch();
    exit(0);
}
void main()
{
    double *q[100];
    long n;
    clrscr();
    set_new_handler(loi_bo_nho);
    // _new_handler=loi_bo_nho;
    n=10000;
    for ( k=0;k<100;i+=k)
        q[k] = new double[n];
    cout << "Khong loi\n";
    getch();
}

```

§8. CÁC HÀM TRONG C++

Trong C++ có rất nhiều mở rộng, cải tiến về hàm làm cho việc xây dựng và sử dụng hàm rất tiện lợi. Điều này sẽ trình bày kỹ trong chương sau. Trong mục này chỉ thống kê một số điểm mới về hàm mà C++ đưa vào.

8.1. Đối kiểu tham chiếu

Trong C, để nhận kết quả của hàm cần dùng đối con trỏ, làm cho việc xây dựng cũng như sử dụng hàm khá phiền phức. Trong C++ đưa vào đối kiểu tham chiếu (giống như PASCAL) dùng để chứa kết quả của hàm, khiến cho việc tạo lập cũng như sử dụng hàm đơn giản hơn.

8.2. Đối tham chiếu const

Đối tham chiếu có đặc điểm là các câu lệnh trong thân hàm có thể truy nhập tới và dễ dàng làm cho giá trị của nó thay đổi. Nhiều khi ta muốn dùng đối kiểu tham chiếu chỉ để tăng tốc độ trao đổi dữ liệu giữa các hàm, không muốn dùng nó để chứa kết quả của hàm. Khi đó có thể dùng đối tham chiếu const để bảo toàn giá trị của đối trong thân hàm.

8.3. Đối có giá trị mặc định

Trong nhiều trường hợp người dùng viết một lời gọi hàm nhưng còn chưa biết nên chọn giá trị nào cho các đối. Để khắc phục khó khăn này, C⁺⁺ đưa ra giải pháp đối có giá trị mặc định. Khi xây dựng hàm, ta gán giá trị mặc định cho một số đối. Người dùng nếu không cung cấp giá trị cho các đối này, thì hàm sẽ dùng giá trị mặc định.

8.4. Hàm inline

Đối với một đoạn chương trình nhỏ (số lệnh không lớn) thì việc thay các đoạn chương trình này bằng các lời gọi hàm sẽ làm cho chương trình gọn nhẹ đôi chút nhưng làm tăng thời gian máy. Trong các trường hợp này có thể dùng hàm trực tuyến (inline) vừa giảm kích thước chương trình nguồn, vừa không làm tăng thời gian chạy máy.

8.5. Các hàm trùng tên (định nghĩa chồng các hàm)

Để lấy giá trị tuyệt đối của một số, trong C cần lập ra nhiều hàm với tên khác nhau, ví dụ abs cho số nguyên, fabs cho số thực, labs cho số nguyên dài, cabs cho số phức. Điều này rõ ràng gây phiền toái cho người sử dụng. Trong C⁺⁺ cho phép xây dựng các hàm trùng tên nhưng khác nhau về kiểu đối. Như vậy chỉ cần lập một hàm để lấy giá trị tuyệt đối cho nhiều kiểu dữ liệu khác nhau.

8.6. Định nghĩa chồng toán tử

Việc dùng các phép toán thay cho một lời gọi hàm rõ ràng làm cho chương trình ngắn gọn, sáng sủa hơn nhiều. Ví dụ để thực hiện phép cộng 2 ma trận nếu dùng phép cộng và viết:

$$C = A + B;$$

thì rất gần với toán học. Trong C⁺⁺ cho phép dùng các phép toán chuẩn để đặt tên cho các hàm (gọi là định nghĩa chồng toán tử). Sau đó có thể thay lời gọi hàm bằng các phép toán như nói ở trên. Như vậy một phép toán mang nhiều ý nghĩa, ví dụ phép + có thể hiểu là cộng 2 số nguyên, 2 số thực hoặc 2 ma trận. C⁺⁺ sẽ căn cứ vào kiểu của các số hạng mà quyết định chọn phép cộng cụ thể.

HÀM TRONG C++

Chương này trình bày những khả năng mới của C++ trong việc xây dựng và sử dụng hàm. Đó là:

- Kiểu tham chiếu và việc truyền dữ liệu cho hàm bằng tham chiếu.
- Đối tượng hằng (const)
- Đối tượng giá trị mặc định
- Hàm trực tuyến
- Việc định nghĩa chồng các hàm
- Việc định nghĩa chồng các toán tử

§1. BIẾN THAM CHIẾU (REFERENCE VARIABLE)

1.1. Hai loại biến dùng trong C

Trước khi nói đến biến tham chiếu, chúng ta nhắc lại 2 loại biến gặp trong C là:

Biến giá trị dùng để chứa dữ liệu (nguyên, thực, ký tự, ...)

Biến con trỏ dùng để chứa địa chỉ

Các biến này đều được cung cấp bộ nhớ và có địa chỉ. Ví dụ câu lệnh khai báo:

```
double x, *px;
```

sẽ tạo ra biến giá trị kiểu double x và biến con trỏ kiểu double px. Biến x có vùng nhớ 8 byte, biến px có vùng nhớ 4 byte (nếu dùng mô hình Large). Biến x dùng để chứa giá trị kiểu double, ví dụ lệnh gán:

```
x = 3.14;
```

sẽ chứa giá trị 3.14 vào biến x. Biến px dùng để chứa địa chỉ của một biến thực, ví dụ câu lệnh:

```
px = &x;
```

sẽ lưu trữ địa chỉ của biến x vào con trỏ px.

1.2. Biến tham chiếu

Trong C++ cho phép sử dụng loại biến thứ ba là biến tham chiếu. So với 2 loại biến quen biết nói trên, thì biến này có những đặc điểm sau:

+ Biến tham chiếu không được cấp phát bộ nhớ, không có địa chỉ riêng.

+ Nó dùng làm bí danh cho một biến (kiểu giá trị) nào đó và nó sử dụng vùng nhớ của biến này. Ví dụ câu lệnh:

```
float u, v, &r = u;
```

tạo ra các biến thực u, v và biến tham chiếu thực r. Biến r không được cấp phát bộ nhớ, nó là một tên khác (bí danh) của u và nó dùng chung vùng nhớ của biến u.

Thuật ngữ: Khi r là bí danh (alias) của u thì ta nói r tham chiếu đến biến u. Như vậy 2 thuật ngữ trên được hiểu như nhau.

Ý nghĩa: Khi r là bí danh của u thì r dùng chung vùng nhớ của u, do đó :

+ Trong mọi câu lệnh, viết u hay viết r đều có ý nghĩa như nhau, vì đều truy nhập đến cùng một vùng nhớ.

+ Có thể dùng biến tham chiếu để truy nhập đến một biến kiểu giá trị.

Ví dụ:

```
int u, v, &r = u;  
r = 10; // u = 10  
cout << u; // in ra số 10  
r++; // u = 11  
+u; // r = 12  
cout << r; // in ra số 12  
v = r; // v = 12  
&r; // Cho địa chỉ của u
```

Công dụng: Biến tham chiếu thường được sử dụng làm đối của hàm để cho phép hàm truy nhập đến các tham số biến trong lời gọi hàm.

Vài chú ý về biến tham chiếu:

a. Vì biến tham chiếu không có địa chỉ riêng, nó chỉ là bí danh của một biến kiểu giá trị nên trong khai báo phải chỉ rõ nó tham chiếu đến biến nào.

Ví dụ nếu khai báo:

```
double &x;
```

thì trình biên dịch sẽ báo lỗi:

Reference variable 'x' must be initialized

b. Biến tham chiếu có thể tham chiếu đến một phần tử mảng, ví dụ:

```
int a[10], &r = a[5];  
r = 25; // a[5] = 25
```

c. Không cho phép khai báo mảng tham chiếu

d. Biến tham chiếu có thể tham chiếu đến một hằng. Khi đó nó sẽ sử dụng vùng nhớ của hằng và nó có thể làm thay đổi giá trị chứa trong vùng nhớ này.

Ví dụ nếu khai báo:

```
int &s = 23;
```

thì trình biên dịch đưa ra cảnh báo (warning):

Temporary used to initialize 's'

Tuy nhiên chương trình vẫn làm việc. Các câu lệnh dưới đây vẫn thực hiện và cho kết quả như sau:

```
s++;  
cout << "\ns = " << s; // In ra s=24
```

Chương trình dưới đây minh họa cách dùng biến tham chiếu đến một phần tử mảng cấu trúc để nhập dữ liệu và thực hiện các phép tính trên các trường của phần tử mảng cấu trúc.

```
#include <iostream.h>  
#include <conio.h>  
struct TS  
{  
    char ht[25];  
    float t, l, h, td;  
};  
void main()  
{  
    TS ts[10], &h = ts[1]; // h tham chiếu đến ts[1]  
    cout << "\nIlo ten: ";  
    cin.get(h.ht, 25);  
    cout << "Cac diem toan, ly, hoa: ";  
    cin >> h.t >> h.l >> h.h;  
    h.td = h.t + h.l + h.h;  
    cout << "\nIlo ten: " << ts[1].ht;  
    cout << "\nTong diem: " << ts[1].td;  
    getch();  
}
```

1.3. Hằng tham chiếu (const)

Hằng tham chiếu được khai báo theo mẫu:

```
int n = 10;  
const int &r = n;
```

Cũng giống như biến tham chiếu, hằng tham chiếu có thể tham chiếu đến một biến hoặc một hằng. Ví dụ:

```
int n = 10;  
const int &r = n; // Hằng tham chiếu r tham chiếu đến biến n  
const int &s = 123; // Hằng tham chiếu s tham chiếu  
// đến hằng 123
```

Sự khác nhau giữa biến và hằng tham chiếu ở chỗ: Không cho phép dùng hằng tham chiếu để làm thay đổi giá trị của vùng nhớ mà nó tham chiếu.

Ví dụ:

```
int y = 12, z;  
const int &py = y; // Hằng tham chiếu py tham  
                    // chiếu đến biến y  
y++; // Đúng  
z = 2 * py; // Đúng z = 26  
cout << y << " " << py; // In ra: 13 13  
    py py+1; // Sai. Trình biên dịch thông báo lỗi:  
    // Cannot modify a const object
```

Cách dùng: Hằng tham chiếu cho phép sử dụng giá trị chứa trong một vùng nhớ, nhưng không cho phép thay đổi giá trị này.

Hằng tham chiếu thường được sử dụng làm đối của hàm để cho phép hàm sử dụng giá trị của các tham số trong lời gọi hàm, nhưng tránh không làm thay đổi giá trị của các tham số.

§2. TRUYỀN GIÁ TRỊ CHO HÀM THEO THAM CHIẾU

2.1. Hàm trong C

Trong C chỉ có một cách truyền dữ liệu cho hàm theo giá trị :

+ Cấp phát vùng nhớ cho các đối.

+ Gán giá trị các tham số trong lời gọi hàm cho các đối sau đó hàm làm việc trên vùng nhớ của các đối chứ không liên quan gì đến các tham số.

Như vậy chương trình sẽ tạo ra các bản sao (các đối) của các tham số và hàm sẽ thao tác trên các bản sao này, chứ không làm việc trực tiếp với các tham số. Phương pháp này có 2 nhược điểm chính:

Tốn kém về thời gian và bộ nhớ vì phải tạo ra các bản sao. Không thao tác trực tiếp trên các tham số, vì vậy không làm thay đổi được giá trị các tham số.

2.2. Truyền giá trị cho hàm theo tham chiếu

Trong C++ cung cấp thêm cách truyền dữ liệu cho hàm theo tham chiếu bằng cách dùng đối là biến tham chiếu hoặc đối là hằng tham chiếu. Cách này có ưu điểm:

Không cần tạo ra các bản sao của các tham số, do đó tiết kiệm bộ nhớ và thời gian chạy máy.

Hàm sẽ thao tác trực tiếp trên vùng nhớ của các tham số, do đó dễ dàng thay đổi giá trị các tham số khi cần.

2.3. Mối quan hệ giữa đối và tham số trong lời gọi hàm

Nếu đối là biến hoặc hằng tham chiếu kiểu K thì tham số (trong lời gọi hàm) phải là biến hoặc phần tử mảng kiểu K. Ví dụ:

+ Đối là biến hoặc hằng tham chiếu kiểu double, thì tham số là biến hoặc phần tử mảng kiểu double

+ Đối là biến hoặc hằng tham chiếu kiểu cấu trúc, thì tham số là biến hoặc phần tử mảng kiểu cấu trúc

2.4. Các chương trình minh họa

/*

Chương trình sau được tổ chức thành 3 hàm:

Nhập dãy số double

Hoán vị 2 biến double

Sắp xếp dãy số double theo thứ tự tăng dần

Chương trình sẽ nhập một dãy số và in dãy sau khi sắp xếp

*/

```
#include <iostream.h>
```

```
#include <conio.h>
```

```
#include <stdio.h>
```

```
void nhapds(double *a, int n)
```

```
{
```

```
    for (int i=1; i<= n; ++i)
```

```
    {
```

```
        cout << "\nPhan tu thu " << i << ": ";
```

```
        cin >> a[i];
```

```
    }
```

```
}
```

```
void hv(double &x, double &y)
```

```
{
```

```
    double tg = x; x = y; y = tg;
```

```
}
```

```
void sapxep(double *a, int n)
```

```
{
```

```
    for (int i=1; i<=n-1; ++i)
```

```
        for (int j=i+1; j<=n; ++j)
```

```
            if (a[i] > a[j])
```

```
                hv(a[i],a[j]);
```

```
}
```

```
void main()
```

```
{
```

```
    double x[100];
```

```
    int i, n;
```

```
    cout << "\nN = ";
```

```
    cin >> n;
```

```

nhapds(x, n);
sapxep(x, n);
for (i=1; i<=n; ++i)
    printf("\n%0.11f", x[i]);
getch();
}
/*

```

Chương trình sau gồm các hàm:

- Nhập dãy cấu trúc (mỗi cấu trúc chứa dữ liệu một thí sinh)
- Hoán vị 2 biến cấu trúc
- Sắp xếp dãy thí sinh theo thứ tự giảm của tổng điểm
- In một cấu trúc (in họ tên và tổng điểm)

Chương trình sẽ nhập dữ liệu một danh sách thí sinh, nhập điểm chuẩn và in danh sách thí sinh trúng tuyển

```

*/
#include <iostream.h>
#include <iomanip.h>
#include <conio.h>
struct TS
{
    char ht[20];
    float t, l, h, td;
};
void ints(const TS &ts)
{
    cout << setiosflags(ios::showpoint) << setprecision(1);
    cout << "\nHọ tên: " << setw(20) << ts.ht
        << setw(6) << ts.td;
}
void nhaps!(TS *ts, int n)
{
    for (int i=1; i<=n; ++i)
    {
        cout << "\nThí sinh " << i;
        cout << "\nHọ tên: ";
        cin.ignore(1);
        cin.get(ts[i].ht, 25) ;
        cout << "Các điểm toán, lý, hóa: ";
        cin >> ts[i].t >> ts[i].l >> ts[i].h;
        ts[i].td = ts[i].t + ts[i].l + ts[i].h;
    }
}
void hvts(TS &ts1, TS &ts2)
{
    TS tg = ts1;
    ts1 = ts2;
}

```

```

    ts2 = tg;
}
void sapxep(TS *ts, int n)
{
    for (int i=1; i<=n-1; ++i)
        for (int j=i+1; j<=n; ++j)
            if (ts[i].td < ts[j].td)
                hvts(ts[i], ts[j]);
}
void main()
{
    TS ts[100];
    int n, i;
    clrscr();
    cout << "So thi sinh: ";
    cin >> n;
    nhapst(ts, n);
    sapxep(ts, n);
    float dc;
    cout << "Diem chuan: ";
    cin >> dc;
    cout << "\n\nDanh sach trung tuyen\n";
    for (i=1; i<=n; ++i)
        if (ts[i].td >= dc)
            ints(ts[i]);
        else
            break;
    getch();
}
/* Chương trình sau gồm các hàm:
Nhập một ma trận thực cấp m * n
In một ma trận thực dưới dạng bảng
Tìm phần tử lớn nhất và phần tử nhỏ nhất của dãy số thực;
Chương trình sẽ nhập một ma trận, in ma trận vừa nhập và in
các phần tử lớn nhất và nhỏ nhất trên mỗi hàng của ma trận */
#include <iostream.h>
#include <iomanip.h>
#include <conio.h>
#include <stdio.h>
void nhapmt(float a[20][20], int m, int n)
{
    for (int i=1 ; i<= m ; ++i)
        for (int j=1; j<= n; ++j)
        {
            cout << "\na[" << i << ", " << j << "] = ";
            cin >> a[i][j];
        }
}
void inmt(float a[20][20], int m, int n)
{
    cout << setiosflags(ios::showpoint) << setprecision(1);

```

```

    for (int i=1 ; i< m ; ++i)
        for (int j=1; j<= n ; ++j)
        {
            if (j==1) cout << "\n";
            cout << setw(6) << a[i][j];
        }
}

void maxminds(float *x, int n, int &vtmax, int &vtmin)
{
    vtmax = vtmin = 1;
    for (int i=2; i<=n ; ++i)
    {
        if (x[i] > x[vtmax]) vtmax = i;
        if (x[i] < x[vtmin]) vtmin = i;
    }
}

void main()
{
    float a[20][20]; int m, n;
    cout << "\n So hang va so cot ma tran: ";
    cin >> m >> n;
    nhapmt(a, m, n);
    clrscr();
    inmt(a, m, n);
    float *p = (float*)a;
    int vtmax, vtmin;
    for (int i=1; i<=m; ++i)
    {
        p = ((float*)a) + i*20;
        maxminds(p, n, vtmax, vtmin);
        printf("\nhang %d Phan tu max = %6.1f tai cot\n", i, p[vtmax], vtmax);
        printf("\n Phan tu min =\n %6.1f tai cot %d", p[vtmin], vtmin);
    }
    getch();
}

```

S3. HÀM TRẢ VỀ CÁC THAM CHIẾU

Hàm có thể có kiểu tham chiếu và trả về giá trị tham chiếu. Khi đó có thể dùng hàm để truy nhập đến một biến hoặc một phần tử mảng nào đó. Dưới đây là một số ví dụ.

Ví dụ 1 trình bày một hàm trả về một tham chiếu đến một biến toàn bộ. Do đó có thể dùng hàm để truy nhập đến biến này.

```

#include <iostream.h>
#include <conio.h>
int z;

```

```

int &f() // Hàm trả về một bí danh của biến toàn bộ z
{
    return z;
}
void main(void)
{
    f() = 50; // z = 50
    cout << "\nz = " << z;
    getch();
}

```

Ví dụ 2 trình bày một hàm trả về bí danh của một biến cấu trúc toàn bộ. Khác với ví dụ trên, ở đây không dùng hàm một cách trực tiếp mà gán hàm cho một biến tham chiếu, sau đó dùng biến tham chiếu này để truy nhập đến biến cấu trúc toàn bộ.

```

#include <iostream.h>
#include <conio.h>
struct TS
{
    char ht[25];
    float t, l, h, td;
};
TS ts;
TS &f()
{
    return ts;
}
void main()
{
    TS &h = f(); // h tham chiếu đến biến ts
    cout << "\nHo ten: ";
    cin.get(h.ht, 25);
    cout << "Cac diem toan, ly, hoa: ";
    cin >> h.t >> h.l >> h.h;
    h.td = h.t + h.l + h.h;
    cout << "\nHo ten: " << ts.ht;
    cout << "\nTong diem: " << ts.td;
    getch();
}

```

Ví dụ 3 trình bày một hàm trả về bí danh của một phần tử mảng cấu trúc toàn bộ.

Hàm sẽ kiểm tra xem chỉ số mảng có vượt ra ngoài miền quy định hay không. Sau đó dùng hàm này để truy nhập đến các phần tử mảng cấu trúc.

```

#include <iostream.h>
#include <conio.h>
#include <stdlib.h>
struct TS
{
    char ht[25];

```

```

    float t, l, h, td;
};
TS *ts;
void cap_phat_bo_nho_nhapsl(int n)
{
    ts = new TS[n+1];
    if (ts==NULL)
    {
        cout << "Loi cap phat bo nho ";
        exit(1);
    }
    for (int i=1; i<=n; ++i)
    {
        TS &h = ts[i];
        cout << "\nThi sinh thu " << i;
        cout << "\n Ho ten: " ;
        cin.ignore(1);
        cin.get(h.ht,25) ;
        cout << "Cac diem toan, ly, hoa: ";
        cin >> h.t >> h.l >> h.h ;
        h.td = h.t + h.l + h.h ;
    }
}
TS &f(int i, int n) // Cho bi danh ts[i]
{
    if (i<1 || i>n)
    {
        cout << "Chi so mang khong hop le ";
        exit(1);
    }
    return ts[i];
}
void main()
{
    int n, i ;
    cout << "\nSo thi sinh : ";
    cin >> n;
    cap_phat_bo_nho_nhapsl(n);
    while (1)
    {
        cout << "\nCan xem thi sinh thu may: ";
        cout << "\nChon so tu 1 den " << n <<
            "\n(bam sai ket thuc CT)";
        cin >> i;
        TS &h = f(i, n);
    }
}

```

```

    cout << "\alto ten: " << h.ht;
    cout << "\nTong diem: " << h.td;
}
}

```

§4. ĐỐI CÓ GIÁ TRỊ MẶC ĐỊNH

4.1. Thế nào là đối mặc định

Một trong các khả năng mạnh của C++ là nó cho phép xây dựng hàm với các đối có giá trị mặc định. Thông thường số tham số trong lời gọi hàm phải bằng số đối của hàm. Mỗi đối sẽ được khởi gán giá trị theo tham số tương ứng của nó. Trong C++ cho phép tạo giá trị mặc định cho các đối. Các đối này có thể có hoặc không có tham số tương ứng trong lời gọi hàm. Khi không có tham số tương ứng, đối được khởi gán bởi giá trị mặc định.

Ví dụ hàm delay với đối số mặc định được viết theo một trong 2 cách sau:

Cách 1 (Không khai báo nguyên mẫu):

```

void delay(int n = 1000)
{
    for (int i=0; i<n; ++i)
        ;
}

```

Cách 2 (Có khai báo nguyên mẫu):

```

void delay(int n = 1000);
void delay(int n)
{
    for (int i=0; i<n; ++i)
        ;
}

```

Cách dùng:

+ Cung cấp giá trị cho đối n (Có tham số trong lời gọi hàm)

```
delay(5000); // Đối n = 5000
```

+ Sử dụng giá trị mặc định của đối (Không có tham số trong lời gọi)

```
delay(); // Đối n = 1000
```

4.2. Quy tắc xây dựng hàm với đối mặc định

+ Các đối mặc định cần phải là các đối cuối cùng tính từ trái sang phải.

Giả sử có 5 đối theo thứ tự từ trái sang phải là

d1, d2, d3, d4, d5

Khi đó:

nếu một đối mặc định thì phải là d5
nếu hai đối mặc định thì phải là d4, d5
nếu ba đối mặc định thì phải là d3, d4, d5

...

Các ví dụ sai:

d3 và d5 mặc định (khi đó d4 cũng phải mặc định)

d3 và d4 mặc định (khi đó d5 cũng phải mặc định)

+ Khi xây dựng hàm, nếu sử dụng khai báo nguyên mẫu, thì các đối mặc định cần được khởi gán trong nguyên mẫu, ví dụ:

```
// Khởi gán giá trị cho 3 đối mặc định d3, d4 và d5
void f(int d1, float d2, char *d3 = "HA NOI",
      int d4 = 100, double d5 = 3.14);
void f(int d1, float d2, char *d3, int d4, double d5)
{
    // Các câu lệnh trong thân hàm
}
```

Không được khởi gán lại cho các đối mặc định trong dòng đầu của định nghĩa hàm. Nếu vi phạm điều này thì chương trình dịch sẽ thông báo lỗi.

+ Khi xây dựng hàm, nếu không khai báo nguyên mẫu, thì các đối mặc định được khởi gán trong dòng đầu của định nghĩa hàm, ví dụ:

```
// Khởi gán giá trị cho 3 đối mặc định d3, d4 và d5
void f(int d1, float d2, char *d3 = "HA NOI",
      int d4 = 100, double d5 = 3.14)
{
    // Các câu lệnh trong thân hàm
}
```

+ Giá trị dùng để khởi gán cho đối mặc định

Có thể dùng các hằng, các biến toàn bộ, các hàm để khởi gán cho đối mặc định, ví dụ:

```
int MAX = 10000;
void f(int n, int m = MAX, int xmax = getmaxx(),
      int ymax = getmaxy());
```

4.3. Cách sử dụng hàm có đối mặc định

Lời gọi hàm cần viết theo quy định sau:

Các tham số thiếu vắng trong lời gọi hàm phải tương ứng với các đối mặc định cuối cùng (tính từ trái sang phải).

Nói cách khác: Đã dùng giá trị mặc định cho một đối (tất nhiên phải là đối mặc định) thì cũng phải sử dụng giá trị mặc định cho các đối còn lại.

Ví dụ với hàm có 3 đối mặc định:

```
void f(int d1, float d2, char *d3 = "HA NOI",  
      int d4 = 100, double d5 = 3.14);
```

Thì các lời gọi sau là đúng:

```
f(3, 3.4, "ABC", 10, 1.0); // Đây đủ tham số  
f(3, 3.4, "ABC"); // Thiếu 2 tham số cuối  
f(3, 3.4); // Thiếu 3 tham số cuối
```

Các lời gọi sau là sai:

```
f(3); // Thiếu tham số cho đối không mặc định d2  
f(3, 3.4,, 10); // Đã dùng giá trị mặc định cho d3, thì cũng  
                // phải dùng giá trị mặc định cho d4 và d5
```

4.4. Các ví dụ

Hàm `ht` (bên dưới) dùng để hiển thị chuỗi ký tự `dc` trên `n` dòng màn hình. Các đối `dc` và `n` đều có giá trị mặc định.

```
#include <conio.h>  
#include <iostream.h>  
void ht(char *dc = "HA NOI", int n = 10);  
void ht(char *dc, int n)  
{  
    for (int i=0; i<n; ++i)  
        cout << "\n" << dc;  
}  
void main()  
{  
    ht(); // In dòng chữ "HA NOI" trên 10 dòng  
    ht("ABC", 3); // In dòng chữ "ABC" trên 3 dòng  
    ht("DEF"); // In dòng chữ "DEF" trên 10 dòng  
    getch();  
}
```

Ví dụ dưới đây trình bày hàm hiển thị một chuỗi `str` trên màn hình đồ họa, tại vị trí `(x, y)` và có màu `m`. Các đối `x`, `y` và `m` là mặc định. Dùng các hàm `getmaxx()` và `getmaxy()` để khởi gán cho `x`, `y`. Dùng hằng `RED` gán cho `m`.

```
#include <conio.h>  
#include <graphics.h>  
void hiendc(char *str, int x = getmaxx() / 2,  
            int y = getmaxy() / 2, int m = RED);  
void hiendc(char *str, int x, int y, int m)  
{  
    int mau_ht = getcolor(); // Lưu màu hiện tại  
    setcolor(m);  
    outtextxy(x, y, str);  
    setcolor(mau_ht); // Khôi phục màu hiện tại  
}
```

```

void main()
{
    int mh = 0, mode = 0;
    initgraph(&mh, &mode, " ");
    setbkcolor(BLUE);
    hiendc("HELLO"); // HELLO màu đỏ giữa màn hình
    hiendc("CHUC MUNG", 1, 1); // CHUC MUNG màu đỏ tại
                                // vị trí (1, 1)
    hiendc("CHAO", 1, 400, YELLOW); // CHAO màu vàng tại
                                // vị trí (1, 400)
    getch();
}

```

Ví dụ dưới đây trình bày hàm tính tích phân xác định gồm 3 đối: f là hàm cần tính tích phân, a và b là các cận dưới và trên ($a < b$). Cả 3 đối f , a và b đều mặc định. Giá trị mặc định của con trỏ hàm f là địa chỉ của hàm bp (bình phương), của a bằng 0, của b bằng 1.

```

#include <conio.h>
#include <iostream.h>
#include <iomanip.h>
#include <math.h>
double bp(double x);
double tp(double (*f)(double) = bp, double a = 0.0,
           double b = 1.0);
double bp(double x)
{
    return x * x;
}
double tp(double (*f)(double), double a, double b)
{
    int n = 1000;
    double s = 0.0, h = (b - a) / n;
    for (int i = 0; i < n; ++i)
        s += f(a + i * h + h) + f(a + i * h);
    return s * h / 2;
}
void main()
{
    clrscr();
    cout << setiosflags(ios::showpoint) << setprecision(2);
    cout << "\nTích phân từ 0 đến 1 của  $x * x =$  " << tp();
    cout << "\nTích phân từ 0 đến 1 của  $\exp(x) =$  " << tp(exp);
    cout << "\nTích phân từ 0 đến  $\pi/2$  của  $\sin(x)$  " <<
        tp(sin, 0, 3.14/2);
    getch();
}

```

§5. CÁC HÀM TRỰC TUYẾN (INLINE)

5.1. Ưu, nhược điểm của hàm

Việc tổ chức chương trình thành các hàm có 2 ưu điểm rõ rệt: Thứ nhất là chia chương trình thành các đơn vị độc lập, làm cho chương trình được tổ chức một cách khoa học dễ kiểm soát để phát hiện lỗi, dễ phát triển, mở rộng.

Thứ hai là giảm được kích thước chương trình, vì mỗi đoạn chương trình thực hiện nhiệm vụ của hàm được thay bằng một lời gọi hàm.

Tuy nhiên hàm cũng có nhược điểm là làm chậm tốc độ chương trình do phải thực hiện một số thao tác có tính thủ tục mỗi khi gọi hàm như: Cấp phát vùng nhớ cho các đối và biến cục bộ, truyền dữ liệu của các tham số cho các đối, giải phóng vùng nhớ trước khi thoát khỏi hàm.

Các hàm trực tuyến trong C++ cho khả năng khắc phục được nhược điểm nói trên.

5.2. Các hàm trực tuyến

Để biến một hàm thành trực tuyến ta viết thêm từ khoá:

`inline`

vào trước khai báo nguyên mẫu hàm. Nếu không dùng nguyên mẫu thì viết từ khoá này trước dòng đầu tiên của định nghĩa hàm. Ví dụ:

```
inline float f(int n, float x);  
float f(int n, float x)  
{  
    // Các câu lệnh trong thân hàm  
}
```

hoặc:

```
inline float f(int n, float x)  
{  
    // Các câu lệnh trong thân hàm  
}
```

Chú ý: Trong mọi trường hợp, từ khoá `inline` phải xuất hiện trước các lời gọi hàm thì trình biên dịch mới biết cần xử lý hàm theo kiểu `inline`.

Ví dụ hàm `f` trong chương trình sau sẽ không phải là hàm trực tuyến vì từ khoá `inline` viết sau lời gọi hàm:

```
#include <conio.h>  
#include <iostream.h>  
void main()  
{  
    int s;  
    s = f(5, 6);  
}
```

```
cout << s;
getch();
}
inline int f(int a, int b)
{
    return a * b;
}
```

Chú ý: Trong C⁺⁺, nếu hàm được xây dựng sau lời gọi hàm thì bắt buộc phải khai báo nguyên mẫu hàm trước lời gọi. Trong ví dụ trên, trình biên dịch C⁺⁺ sẽ bắt lỗi vì thiếu khai báo nguyên mẫu hàm f.

5.3. Cách biên dịch hàm trực tuyến

Chương trình dịch xử lý các hàm inline như các macro (được định nghĩa trong lệnh #define), nghĩa là nó sẽ thay mỗi lời gọi hàm bằng một đoạn chương trình thực hiện nhiệm vụ của hàm. Cách này làm cho chương trình dài ra, nhưng tốc độ chương trình tăng lên do không phải thực hiện các thao tác có tính thủ tục khi gọi hàm.

5.4. So sánh macro và hàm trực tuyến

Dùng macro và hàm trực tuyến đều dẫn đến hiệu quả tương tự, tuy nhiên người ta thích dùng hàm trực tuyến hơn, vì cách này đảm bảo tính cấu trúc của chương trình, dễ sử dụng và tránh được các sai sót lặt vặt thường gặp khi dùng #define (như thiếu các dấu ngoặc, dấu chấm phẩy).

5.5. Khi nào thì nên dùng hàm trực tuyến

Phương án dùng hàm trực tuyến rút ngắn được thời gian chạy máy nhưng lại làm tăng khối lượng bộ nhớ chương trình (nhất là đối với các hàm trực tuyến có nhiều câu lệnh). Vì vậy chỉ nên dùng phương án trực tuyến đối với các hàm nhỏ.

5.6. Sự hạn chế của Trình biên dịch

Không phải khi gặp từ khoá inline là trình biên dịch nhất thiết phải xử lý hàm theo kiểu trực tuyến.

Chú ý rằng từ khoá inline chỉ là một sự gợi ý cho trình biên dịch chứ không phải là một mệnh lệnh bắt buộc.

Có một số hàm mà các trình biên dịch thường không xử lý theo cách inline như các hàm chứa biến static, hàm chứa các lệnh chu trình hoặc lệnh goto hoặc lệnh switch, hàm đệ quy. Trong trường hợp này từ khoá inline lẽ dĩ nhiên bị bỏ qua.

Thậm chí từ khoá inline vẫn bị bỏ qua ngay cả đối với các hàm không có những hạn chế nêu trên nếu như trình biên dịch thấy cần thiết (ví dụ đã có quá nhiều hàm inline làm cho bộ nhớ chương trình quá lớn).

Ví dụ: Chương trình sau sử dụng hàm inline tính chu vi và diện tích của hình chữ nhật:

Phương án 1: Không khai báo nguyên mẫu. Khi đó hàm dtcvhcn phải đặt trên hàm main.

```
#include <conio.h>
#include <iostream.h>
inline void dtcvhcn(int a, int b, int &dt, int &cv)
{
    dt = a * b;
    cv = 2 * (a + b);
}
void main()
{
    int a[20], b[20], cv[20], dt[20], n;
    cout << "\nSố hình chữ nhật: ";
    cin >> n;
    for (int i=1; i<=n; ++i)
    {
        cout << "\nNhập 2 cạnh của hình chữ nhật thứ "
              << i << ": ";
        cin >> a[i] >> b[i];
        dtcvhcn(a[i], b[i], dt[i], cv[i]);
    }
    clrscr();
    for (i=1; i<=n; ++i)
    {
        cout << "\nHình chữ nhật thứ " << i << ": ";
        cout << "\nĐộ dài 2 cạnh = " << a[i] << " và " << b[i];
        cout << "\nDiện tích = " << dt[i];
        cout << "\nChu vi = " << cv[i];
    }
    getch();
}
```

Phương án 2: Sử dụng khai báo nguyên mẫu. Khi đó từ khoá inline đặt trước nguyên mẫu.

Chú ý: Không được đặt inline trước định nghĩa hàm. Trong chương trình dưới đây nếu đặt inline trước định nghĩa hàm thì hậu quả như sau: Chương trình vẫn dịch thông, nhưng khi chạy thì chương trình bị quán, không thoát được.

```
#include <conio.h>
#include <iostream.h>
inline void dtcvhcn(int a, int b, int &dt, int &cv);
void main()
{
    int a[20], b[20], cv[20], dt[20], n;
    cout << "\nSố hình chữ nhật: ";
    cin >> n;
```

```

for (int i=1; i<=n; ++i)
{
    cout << "\nNhap 2 canh cua hinh chu nhat thu "
        <<i<< ": ";
    cin >> a[i] >> b[i];
    dtcvhn(a[i], b[i], dt[i], cv[i]);
}
clrscr();
for (i=1; i<=n; ++i)
{
    cout << "\n Hinh chu nhat thu " << i << ": ";
    cout << "\nDo dai 2 canh = " << a[i] << " va " << b[i];
    cout << "\nDien tich = " << dt[i];
    cout << "\nChu vi = " << cv[i];
}
getch();
}
void dtcvhn(int a, int b, int &dt, int &cv)
{
    dt = a * b;
    cv = 2 * (a + b);
}

```

§6. ĐỊNH NGHĨA CHỒNG CÁC HÀM (OVERLOADING)

6.1. Khái niệm về định nghĩa chồng

Định nghĩa chồng (hay còn gọi sự tải bội) các hàm là dùng cùng một tên để định nghĩa các hàm khác nhau. Đây là một mở rộng rất có ý nghĩa của C++.

Như đã biết, trong C và các ngôn ngữ khác (như PASCAL, FORTRAN, ...) mỗi hàm đều phải có một tên phân biệt. Đôi khi đây là một sự hạn chế lớn, vì phải dùng nhiều hàm khác nhau để thực hiện cùng một công việc. Ví dụ để lấy giá trị tuyệt đối trong C cần dùng tới 3 hàm khác nhau:

```

int abs(int i); // Lấy giá trị tuyệt đối giá trị kiểu int
longt labs(longt l); // Lấy giá trị tuyệt đối
// giá trị kiểu long
double fabs(double d); // Lấy giá trị tuyệt đối
// giá trị kiểu double

```

Nhờ khả năng định nghĩa chồng, trong C++ có thể dùng chung một tên cho cả 3 hàm trên như sau:

```

int abs(int i); // Lấy giá trị tuyệt đối giá trị kiểu int
longt abs(longt l); // Lấy giá trị tuyệt đối
// giá trị kiểu long
double abs(double d); // Lấy giá trị tuyệt đối
// giá trị kiểu double

```

6.2. Yêu cầu về các hàm định nghĩa chồng

Khi dùng cùng một tên để định nghĩa nhiều hàm, trình biên dịch C++ sẽ dựa vào sự khác nhau về tập đối của các hàm này để đổi tên các hàm. Như vậy, sau khi biên dịch mỗi hàm sẽ có một tên khác nhau.

Từ đó cho thấy: các hàm được định nghĩa trùng tên phải có tập đối khác nhau (về số lượng hoặc kiểu). Nếu 2 hàm hoàn toàn trùng tên và trùng đối thì trình biên dịch sẽ không có cách nào phân biệt được. Ngay cả khi 2 hàm này có kiểu khác nhau thì trình biên dịch vẫn báo lỗi. Ví dụ sau xây dựng 2 hàm cùng có tên là `f` và cùng có một đối nguyên `a`, nhưng kiểu hàm khác nhau. Hàm thứ nhất kiểu nguyên (trả về `a*a`), hàm thứ hai kiểu void (in giá trị `a`). Chương trình sẽ bị thông báo lỗi khi biên dịch (bạn hãy thử xem sao).

```
#include <conio.h>
#include <iostream.h>
int f(int a);
void f(int a);
int f(int a)
{
    return a * a;
}
void f(int a)
{
    cout << "\n " << a;
}
void main()
{
    int b = f(5);
    f(b);
    getch();
}
```

6.3. Sử dụng các hàm định nghĩa chồng

Khi gặp một lời gọi, trình biên dịch sẽ căn cứ vào số lượng và kiểu của các tham số để gọi hàm có đúng tên và đúng bộ đối số tương ứng. Ví dụ:

```
abs(123); // Tham số kiểu int, gọi hàm int abs(int i) ;
abs(123L); // Tham số kiểu long, gọi hàm long abs(long l);
abs(3.14); // Tham số kiểu double, gọi hàm
           // double abs(double d);
```

Khi không có hàm nào có bộ đối cùng kiểu với bộ tham số (trong lời gọi), thì trình biên dịch sẽ chọn hàm nào có bộ đối gần kiểu nhất (phép chuyển kiểu dễ dàng nhất).

Ví dụ:

```
abs('A'); // Tham số kiểu char, gọi hàm int abs(int i) ;  
abs(3.14F); // Tham số kiểu float, gọi hàm  
           // double abs(double d);
```

6.4. Nên sử dụng phép định nghĩa chồng các hàm như thế nào?

Như đã nói ở trên, khi xây dựng cũng như sử dụng các hàm trùng tên, trình biên dịch C++ đã phải suy đoán và giải quyết nhiều trường hợp khá nhập nhằng. Vì vậy không nên lạm dụng quá đáng khả năng định nghĩa chồng, vì điều đó làm cho chương trình khó kiểm soát và dễ dẫn đến sai sót. Việc định nghĩa chồng sẽ hiệu quả hơn nếu được sử dụng theo các lời khuyên sau:

- + Chỉ nên định nghĩa chồng các hàm thực hiện những công việc như nhau nhưng trên các đối tượng có kiểu khác nhau. Ví dụ trong chương trình cần xây dựng các hàm: cộng 2 ma trận vuông kiểu double, cộng 2 ma trận vuông kiểu int, cộng 2 ma trận chữ nhật kiểu double, cộng 2 ma trận chữ nhật kiểu int, thì 4 hàm trên nên định nghĩa chồng (đặt cùng tên).

- + Nên dùng các phép chuyển kiểu (nếu cần) để bộ tham số trong lời gọi hoàn toàn trùng kiểu với bộ đối số của một hàm được định nghĩa chồng. Vì như thế mới tránh được sự nhập nhằng cho trình biên dịch và trình biên dịch sẽ chọn đúng hàm cần gọi.

6.5. Lấy địa chỉ các hàm trùng tên

Giả sử có 4 hàm đều có tên là `tính_max` được khai báo như sau:

```
int tính_max(int a, int b, int c); // Max của 3 số nguyên  
double tính_max(double a, double b, double c); // Max của  
                                                // 3 số thực  
int tính_max(int *a, int n); // Max của một dãy số nguyên  
double tính_max(double *a, int n); //Max của một dãy số thực
```

Vấn đề đặt ra là làm thế nào lấy được địa chỉ của mỗi hàm. Câu trả lời như sau:

Để lấy địa chỉ của một hàm, ta khai báo một con trỏ hàm có kiểu và bộ đối như hàm cần lấy địa chỉ. Sau đó gán tên hàm cho con trỏ hàm. Ví dụ:

```
int (*f1)(int, int, int);  
f1 = tính_max; // lấy địa chỉ của hàm thứ nhất  
double (*f2)(double, double, double);  
f2 = tính_max; // lấy địa chỉ của hàm thứ hai  
int (*f3)(int *, int);  
f3 = tính_max; // Lấy địa chỉ của hàm thứ ba  
double (*f4)(double *, int);  
f4 = tính_max; // Lấy địa chỉ của hàm thứ tư
```

6.6. Các ví dụ

Ví dụ 1: Chương trình giải bài toán tìm max của một dãy số nguyên và max của một dãy số thực. Trong chương trình có 6 hàm. Hai hàm dùng để nhập dãy số nguyên và dãy số thực có tên chung là `nhapds`. Bốn hàm: tính max 2 số nguyên, tính max 2 số thực, tính max của dãy số nguyên, tính max của dãy số thực được đặt chung một tên là `max`.

```
#include <conio.h>
#include <iostream.h>
#include <iomanip.h>
void nhapds(int *x, int n);
void nhapds(double *x, int n);
int max(int x, int y);
double max(double x, double y);
int max(int *x, int n);
double max(double *x, int n);
void nhapds(int *x, int n)
{
    for (int i=1; i<=n; ++i)
    {
        cout << "Phan tu " << i << " = ";
        cin >> x[i];
    }
}
void nhapds(double *x, int n)
{
    for (int i=1; i<=n; ++i)
    {
        cout << "Phan tu " << i << " = ";
        cin >> x[i];
    }
}
int max(int x, int y)
{
    return x > y ? x : y;
}
double max(double x, double y)
{
    return x > y ? x : y;
}
int max(int *x, int n)
{
    int s = x[1];
    for (int i=2; i<=n; ++i)
        s = max(s, x[i]);
    return s;
}
double max(double *x, int n)
{

```

```

double s = x[1];
for (int i=2; i<=n; ++i)
    s = max(s, x[i]);
return s;
}

void main()
{
    int a[20], n, ni, nd, maxi;
    double x[20], maxd;
    clrscr();
    cout << "\nSo phan tu nguyen ni = ";
    cin >> ni;
    cout << "Nhap day so nguyen\n ";
    nhapds(a, ni);
    cout << "\nSo phan tu thuc nd = ";
    cin >> nd;
    cout << "Nhap day so thuc\n ";
    nhapds(x, nd);
    maxi = max(a, ni);
    maxd = max(x, nd);
    cout << "\nMax cua day nguyen = " << maxi;
    cout << "\nMax cua day thuc = " << maxd;
    getch();
}

```

Ví dụ 2:

Chương trình sau thực hiện phép nhân ma trận:

$$D = A * B * C$$

trong đó A, B là các ma trận vuông, C là ma trận chữ nhật. Trong chương trình có 3 cặp hàm trùng tên để thực hiện 3 nhiệm vụ (nhưng trên 2 đối tượng khác nhau là ma trận vuông và chữ nhật): Nhập ma trận, nhân 2 ma trận và in ma trận.

```

#include <conio.h>
#include <iostream.h>
#include <iomanip.h>
typedef int MT[20][20];
void nhapmt(MT a, char *ten, int m, int n);
void inmt(MT a, char *ten, int m, int n);
void nhanmt(MT a, MT b, MT c, int m, int n, int p);
void nhapmt(MT a, char *ten, int m, int n);
void inmt(MT a, char *ten, int m, int n);
void nhanmt(MT a, MT b, MT c, int m, int n);
void nhapmt(MT a, char *ten, int m, int n)
{
    for (int i=1; i<=m; ++i)
        for (int j=1; j<=n; ++j)
        {
            cout << "\n" << ten << "[" << i << ", " << j << "] = ";

```

```

        cin >> a[i][j];
    }
}
void nhapmt(MT a, char *ten, int n)
{
    nhapmt(a, ten, n, n);
}
void inmt(MT a, char *ten, int m, int n)
{
    cout << "\nMa tran: " << ten;
    for (int i=1; i<=m; ++i)
    {
        cout << "\n";
        for (int j=1; j<=n; ++j)
            cout << setw(6) << a[i][j];
    }
}
void inmt(MT a, char *ten, int n)
{
    inmt(a, ten, n, n);
}
void nhanmt(MT a, MT b, MT c, int m, int n, int p)
{
    for (int i=1; i<=m; ++i)
        for (int j=1; j<=p; ++j)
        {
            c[i][j] = 0;
            for (int k=1; k<=n; ++k)
                c[i][j] += a[i][k] * b[k][j];
        }
}
void nhanmt(MT a, MT b, MT c, int n)
{
    nhanmt(a, b, c, n, n, n);
}
void main()
{
    MT a, b, c, d; // d = abc
    MT u;
    clrscr();
    nhapmt(a, "A", 2);
    nhapmt(b, "B", 2);
    nhapmt(c, "C", 2, 3);
    nhanmt(a, b, u, 2);
    nhanmt(u, c, d, 2, 2, 3);
    inmt(a, "A", 2);
    inmt(b, "B", 2);
    inmt(u, "U = A * B", 2);
    inmt(c, "C", 2, 3);
    inmt(d, "D = U * C", 2, 3);
    getch();
}

```

§7. ĐỊNH NGHĨA CHỐNG CÁC TOÁN TỬ

7.1. Các phép toán trong C⁺⁺ và C

Trong C⁺⁺ và C có khá nhiều các phép toán dùng để thực hiện các thao tác trên các kiểu dữ liệu chuẩn. Ví dụ các phép số học: + - * / áp dụng cho các kiểu dữ liệu nguyên, thực. Phép lấy phần dư % áp dụng đối với kiểu nguyên.

7.2. Thực hiện các phép toán trên các kiểu dữ liệu không chuẩn trong C

Việc thực hiện các phép toán trên các đối tượng tự định nghĩa (như mảng, cấu trúc) là nhu cầu bắt buộc của thực tế. Chẳng hạn cần thực hiện các phép số học trên số phức, trên phân số, trên đa thức, trên véc tơ, trên ma trận. Để đáp ứng yêu cầu này, ta sử dụng các hàm trong C. Ví dụ sau đây là một chương trình C gồm các hàm nhập phân số, in phân số và thực hiện các phép cộng trừ nhân chia phân số. Chương trình sẽ nhập 5 phân số: p, q, z, u, v và tính phân số s theo công thức:

$$s = (p \cdot q * z) / (u + v)$$

```
#include <conio.h>
#include <stdio.h>
#include <math.h>
typedef struct
{
    int a, b;
} PS;
void nhap(PS *p);
void in(PS p);
int uscln(int x, int y);
PS rutgon(PS p);
PS cong(PS p1, PS p2);
PS tru(PS p1, PS p2);
PS nhan(PS p1, PS p2);
PS chia(PS p1, PS p2);
void nhap(PS *p)
{
    int t, m;
    printf("\nTu va mau: ");
    scanf("%d%d", &t, &m);
    p->a = t; p->b = m;
}
void in(PS p)
{
    printf("%d/%d", p.a, p.b);
}
int uscln(int x, int y)
{

```

```

    x = abs(x); y = abs(y);
    if (x*y != 0) return 1;
        while (x!=y)
            if (x>y) x -= y;
            else y -= x;
    return x;
}
PS rutgon(PS p)
{
    PS q;
    int x;
    x=uscln(p.a, p.b);
    q.a = p.a / x;
    q.b = p.b / x;
    return q;
}
PS cong(PS p1, PS p2)
{
    PS q;
    q.a = p1.a * p2.b + p2.a * p1.b;
    q.b = p1.b * p2.b;
    return rutgon(q);
}
PS tru(PS p1, PS p2)
{
    PS q;
    q.a = p1.a * p2.b - p2.a * p1.b;
    q.b = p1.b * p2.b;
    return rutgon(q);
}
PS nhan(PS p1, PS p2)
{
    PS q;
    q.a = p1.a * p2.a;
    q.b = p1.b * p2.b;
    return rutgon(q);
}
PS chia(PS p1, PS p2)
{
    PS q;
    q.a = p1.a * p2.b;
    q.b = p1.b * p2.a;
    return rutgon(q);
}
void main()
{
    PS p, q, z, u, v ;
    PS tu, mau, s;
    printf("\n Nhap phan so p: "); nhap(&p);

```

```

printf("\n Nhap phan so q: "); nhap(&q);
printf("\n Nhap phan so z: "); nhap(&z);
printf("\n Nhap phan so u: "); nhap(&u);
printf("\n Nhap phan so v: "); nhap(&v);
tu = nhan(q, z);
tu = tru(p, tu);
mau = cong(u, v);
s = chia(tu, mau);
printf("\nPhan so s = "); in(s);
getch();
}

```

Nhận xét: Việc sử dụng các hàm để thực hiện các phép tính không được tự nhiên và tỏ ra dài dòng. Ví dụ để thực hiện một công thức:

$$s = (p - q * z) / (u + v)$$

phải dùng 2 biến trung gian và 4 lời gọi hàm. Câu hỏi đặt ra là có cách nào để chỉ cần viết đúng công thức toán học, mà vẫn nhận được kết quả mong muốn hay không?

Trong C++ có thể đáp ứng được mong muốn này bằng cách sử dụng các phép toán chuẩn của nó cho các kiểu dữ liệu tự định nghĩa (mảng, cấu trúc,...). Nói cách khác C++ cho phép dùng các phép toán để định nghĩa các hàm, mà ta thường gọi là định nghĩa chồng các toán tử (hay còn gọi: Sự tải bội các toán tử).

7.3. Cách định nghĩa chồng các toán tử

7.3.1. Tên hàm toán tử: Gồm từ khoá operator và tên phép toán, ví dụ:

operator+ (định nghĩa chồng phép +)

operator- (định nghĩa chồng phép -)

7.3.2. Các đối của hàm toán tử:

a. Với các phép toán có 2 toán hạng, thì hàm toán tử cần có 2 đối. Đối thứ nhất ứng với toán hạng thứ nhất, đối thứ hai ứng với toán hạng thứ hai. Do vậy, với các phép toán không giao hoán (như phép-) thì thứ tự đối là rất quan trọng.

Ví dụ các hàm toán tử cộng, trừ phân số được khai báo như sau:

```

struct PS
{
    int a; // Tử số
    int b; // Mẫu số
};
PS operator+(PS p1, PS p2); // p1 + p2
PS operator-(PS p1, PS p2); // p1 - p2
PS operator*(PS p1, PS p2); // p1 * p2
PS operator/(PS p1, PS p2); // p1 / p2

```

b. Với các phép toán có một toán hạng, thì hàm toán tử có một đối. Ví dụ hàm toán tử đổi dấu ma trận (đổi dấu tất cả các phần tử của ma trận) được khai báo như sau:

```
struct MT
{
    double a[20][20]; // Mảng chứa các phần tử ma trận
    int m; // Số hàng ma trận
    int n; // Số cột ma trận
};
MT operator-(MT x);
```

7.3.3. Thân của hàm toán tử: Viết như thân của hàm thông thường. Ví dụ hàm đổi dấu ma trận có thể được định nghĩa như sau:

```
struct MT
{
    double a[20][20]; // Mảng chứa các phần tử ma trận
    int m; // Số hàng ma trận
    int n; // Số cột ma trận
};
MT operator-(MT x)
{
    MT y;
    for (int i=1; i<= m ; ++i)
        for (int j=1; j<= n; ++j)
            y[i][j] = -x[i][j];
    return y;
}
```

7.4. Cách dùng hàm toán tử

Có 2 cách dùng:

Cách 1: Dùng như một hàm thông thường bằng cách viết lời gọi.

Ví dụ:

```
PS p, q, u, v;
u = operator+(p, q); // u = p + q
v = operator-(p, q); // v = p - q
```

Cách 2: Dùng như phép toán của C⁺⁺.

Ví dụ:

```
PS p, q, u, v;
u = p + q; // u = p + q
v = p - q; // v = p - q
```

Chú ý: Khi dùng các hàm toán tử như phép toán của C⁺⁺ ta có thể kết hợp nhiều phép toán để viết các công thức phức tạp. Cũng cho phép dùng dấu ngoặc tròn để quy định thứ tự thực hiện các phép tính. Thứ tự ưu tiên của các phép tính vẫn tuân theo các quy tắc ban đầu của C⁺⁺. Chẳng hạn các phép * và / có thứ ưu tiên cao hơn so với các phép + và -

Ví dụ:

```
PS p, q, u, v, s1, s2;  
s1 = p * q - u / v; // s1 = (p * q)  
s2 = (p - q) / (u + v); // s2 = (p - q) / (u + v)
```

§8. CÁC VÍ DỤ VỀ ĐỊNH NGHĨA CHỒNG TOÁN TỬ

Ví dụ 1: Trong ví dụ này ngoài việc sử dụng các hàm toán tử để thực hiện 4 phép tính trên phân số, còn định nghĩa chồng các phép toán << và >> để xuất và nhập phân số (xem chi tiết trong *Chương 7*).

Hàm `operator<<` có 2 đối kiểu `ostream&` và PS (Phân số). Hàm trả về giá trị kiểu `ostream&`. Hàm được khai báo như sau:

```
ostream& operator<< (ostream& os, PS p);
```

Tương tự hàm `operator>>` được khai báo như sau:

```
istream& operator>> (istream& is, PS &p);
```

Dưới đây sẽ chỉ ra cách xây dựng và sử dụng các hàm toán tử. Chúng ta cũng sẽ thấy việc sử dụng các hàm toán tử rất tự nhiên, ngắn gọn và tiện lợi.

Chương trình dưới đây có nội dung như chương trình trong §6.2, nhưng thay các hàm bằng các hàm toán tử.

```
#include <conio.h>  
#include <iostream.h>  
#include <math.h>  
typedef struct  
{  
    int a,b;  
} PS;  
ostream& operator<< (ostream& os, PS p);  
istream& operator>> (istream& is, PS &p);  
int uscln(int x, int y);  
PS rutgon(PS p);  
PS operator+(PS p1, PS p2);  
PS operator-(PS p1, PS p2);  
PS operator*(PS p1, PS p2);  
PS operator/(PS p1, PS p2);  
ostream& operator<< (ostream& os, PS p)  
{  
    os << p.a << '/' << p.b;  
    return os;  
}  
istream &operator>> (istream &is, PS &p)  
{  
    cout << "Nhap tu va mau: ";  
    is >> p.a >> p.b;  
    return is;  
}
```

```

int uscln(int x, int y)
{
    x = abs(x); y = abs(y);
    if (x*y==0) return 1;
    while (x!=y)
        if (x>y) x -= y;
        else y -= x;
    return x;
}
PS rutgon(PS p)
{
    PS q;
    int x;
    x = uscln(p.a,p.b);
    q.a = p.a / x;
    q.b = p.b / x;
    return q;
}
PS operator+(PS p1, PS p2)
{
    PS q;
    q.a = p1.a * p2.b + p2.a * p1.b;
    q.b = p1.b * p2.b;
    return rutgon(q);
}
PS operator-(PS p1, PS p2)
{
    PS q;
    q.a = p1.a * p2.b - p2.a * p1.b;
    q.b = p1.b * p2.b;
    return rutgon(q);
}
PS operator*(PS p1, PS p2)
{
    PS q;
    q.a = p1.a * p2.a ;
    q.b = p1.b * p2.b ;
    return rutgon(q);
}
PS operator/(PS p1, PS p2)
{
    PS q;
    q.a = p1.a * p2.b;
    q.b = p1.b * p2.a;
    return rutgon(q);
}
void main()
{
    PS p, q, z, u, v;
    PS s;
    cout <<"\nNhap cac PS p, q, z, u, v:\n ";

```

```

cin >> p >> q >> z >> u >> v;
s = (p - q * z) / (u + v) ;
cout << "\n Phan so s = " << s;
getch();
}

```

Ví dụ 2: Chương trình đưa vào các hàm toán tử:

operator- có một đối dùng để đảo dấu một đa thức

operator+ có 2 đối dùng để cộng 2 đa thức

operator- có 2 đối dùng để trừ 2 đa thức

operator* có 2 đối dùng để nhân 2 đa thức

operator^ có 2 đối dùng để tính giá đa thức tại x

operator<< có 2 đối dùng để in đa thức

operator>> có 2 đối dùng để nhập đa thức

Chương trình sẽ nhập 4 đa thức: p, q, r, s. Sau đó tính đa thức:

$$f = -(p + q) * (r - s)$$

Cuối cùng tính giá trị $f(x)$, với x là một số thực nhập từ bàn phím.

```

#include <conio.h>
#include <iostream.h>
#include <math.h>
struct DT
{
double a[20]; // Mang chua cac he so da thuc a0, a1,...
int n; // Bac da thuc
};
ostream& operator<< (ostream &os, DT d);
istream& operator>> (istream &is,DT &d);
DT operator-(const DT &d);
DT operator+(DT d1, DT d2);
DT operator-(DT d1, DT d2);
DT operator*(DT d1, DT d2);
double operator^(DT d, double x); // Tinh gia tri da thuc
ostream &operator<< (ostream &os, DT d)
{
os << "- Cac he so (tu ao): ";
for (int i=0 ; i<= d.n ; ++i)
os << d.a[i] <<" ";
return os;
}
istream& operator>> (istream& is, DT &d)
{
cout << "- Bac da thuc: ";
cin >> d.n;
cout << "Nhap cac he so da thuc:\n";
for (int i=0 ; i<= d.n ; ++i)
{

```

```

        cout << "He so bac " << i << " = ";
        is >> d.a[i];
    }
    return is;
}
DT operator-(const DT &d)
{
    DT p;
    p.n = d.n;
    for (int i=0 ; i<=d.n ; ++i)
        p.a[i] = -d.a[i];
    return p;
}
DT operator+(DT d1, DT d2)
{
    DT d;
    int k,i;
    k = d1.n > d2.n ? d1.n : d2.n;
    for (i=0; i<=k ; ++i)
        if (i<=d1.n && i<=d2.n)
            d.a[i] = d1.a[i] + d2.a[i];
        else if (i<=d1.n)
            d.a[i] = d1.a[i];
        else
            d.a[i] = d2.a[i];
    i = k;
    while (i>0 && d.a[i] == 0.0) --i;
    d.n = i;
    return d;
}
DT operator-(DT d1, DT d2)
{
    return (d1 + (-d2));
}
DT operator*(DT d1, DT d2)
{
    DT d;
    int k, i, j;
    k = d.n = d1.n + d2.n;
    for (i=0; i<=k; ++i) d.a[i] = 0;
    for (i=0 ; i<= d1.n ; ++i)
        for (j=0 ; j<= d2.n ; ++j)
            d.a[i+j] += d1.a[i] * d2.a[j];
    return d;
}
double operator^(DT d, double x)
{
    double s 0.0, t 1.0;
    for (int i=0 ; i<= d.n ; ++i)
    {
        s += d.a[i] * t;
    }

```

```

    t *= x;
}
return s;
}
void main()
{
    DT p, q, r, s, f;
    double x, g;
    clrscr();
    cout << "\nNhap da thuc P " ; cin >> p;
    cout << "\nNhap da thuc Q " ; cin >> q;
    cout << "\nNhap da thuc R " ; cin >> r;
    cout << "\nNhap da thuc S " ; cin >> s;
    cout << "\nNhap so thuc x: " ; cin >> x;
    f = -(p + q) * (r - s);
    g = f ^ x;
    cout << "\nDa thuc f " << f;
    cout << "\n x " << x;
    cout << "\nf(x) = " << g;
    getch();
}

```

59. CÁC BÀI TOÁN VỀ MA TRẬN VÀ VEC TƠ

Trong mục này sẽ xét các ma trận thực vuông cấp n và các véc tơ thực cấp n . Chúng được biểu diễn thông qua các kiểu cấu trúc MT và VT:

```

struct MT
{
    double a[20][20]; // Mảng a chứa các phần tử ma trận
    int n;             // Cấp ma trận
};

struct VT
{
    double b[20]; // Mảng chứa các phần tử của véc tơ
    int n; // Cấp véc tơ
};

```

Để xử lý ma trận và véc tơ, chúng ta xây dựng 9 hàm toán tử:

```

ostream& operator<< (ostream& os, const MT& x); // In ma trận
ostream& operator<< (ostream& os, const VT& v); // In véc tơ
istream& operator>> (istream& is, MT& x); // Nhập ma trận
istream& operator>> (istream& is, VT& v); // Nhập véc tơ
MT operator+(const MT& x1, const MT& x2); // Cộng 2 ma trận
MT operator-(const MT& x1, const MT& x2); // Trừ 2 ma trận
MT operator*(const MT& x1, const MT& x2); // Nhân 2 ma trận
VT operator*(const MT& x, const VT& v); // Nhân ma trận véc tơ
MT operator!(MT x); // Nghịch đảo ma trận

```

Thuật toán cho 8 hàm toán tử đầu tương đối quen thuộc không có gì phải bàn. Đề nghị đảo ma trận có nhiều cách, ở đây chúng ta dùng phương pháp Jordance như sau. Giả sử cần nghịch đảo ma trận x cấp n . Ta dùng thêm ma trận đơn vị y . Sau đó thực hiện đồng thời các phép tính trên cả x và y sao cho x trở thành đơn vị. Kết quả y chính là nghịch đảo của x . Thuật toán được tiến hành trên n bước. Nội dung của bước k ($k = 1, \dots, n$) như sau:

Tìm chỉ số r ($k \leq r \leq n$) sao cho

$$\text{abs}(x[r,k]) = \max \{ \text{abs}(x[i,k]) \text{ với } i = k, \dots, n \}$$

Nếu $\text{abs}(x[r,k]) = 0$ thì ma trận không có nghịch đảo và thuật toán kết thúc giữa chừng.

Hoán vị hàng k với hàng r trong cả 2 ma trận x và y .

Chia hàng k của cả x và y cho $\text{tg} = x[k,k]$ (mục đích làm cho $x[k,k] = 1$).

Biến đổi đề cột k của x trở thành véc tơ đơn vị bằng cách làm cho các phần tử $x[i,k] = 0$ (với i khác k). Muốn vậy ta thực hiện các phép tính sau trên cả x và y :

$$(\text{hàng } i) = (\text{hàng } i) - x[i,k](\text{hàng } k), \text{ với mọi } i \text{ khác } k$$

Nội dung chương trình là nhập 4 ma trận X, Y, R, S và véc tơ u . Sau đó tính véc tơ v theo công thức:

$$v = ((X + Y) * (R - S))^{-1} u$$

Như sẽ thấy trong hàm `main()` dưới đây, nhờ các hàm toán tử mà câu lệnh tính v được viết gần giống như công thức toán học nêu trên.

```
/* Chương trình */
#include <conio.h>
#include <iostream.h>
#include <iomanip.h>
#include <math.h>
struct MT
{
    double a[20][20]; // Mang chua cac phan tu ma tran
    int n; // Cap ma tran
};
struct VT
{
    double b[20]; // Mang chua cac phan tu cua vec to
    int n; // Cap vec to
};
ostream& operator<< (ostream& os, const MT& x);
ostream& operator<< (ostream& os, const VT& v);
istream& operator>> (istream& is, MT& x);
istream& operator>> (istream& is, VT &v);
```

```

MT operator+(const MT& x1, const MT& x2);
MT operator-(const MT& x1, const MT& x2);
MT operator*(const MT& x1, const MT& x2);
VT operator*(const MT& x, const VT& v);
MT operator!(MT x); // Tinh ma tran nghich dao
ostream& operator<< (ostream& os, const MT& x)
{
    os << setprecision(2) << setiosflags(ios::showpoint);
    for (int i=1 ; i<= x.n ; ++i)
    {
        os << "\n" ;
        for (int j=1; j<=x.n; ++j)
            os << setw(6) << x.a[i][j] ;
    }
    os << "\n" ;
    return os;
}
ostream& operator<< (ostream& os, const VT& v)
{
    os << setprecision(2) << setiosflags(ios::showpoint);
    for (int i=1 ; i<= v.n ; ++i)
        os << setw(6) << v.b[i] ;
    os << "\n" ;
    return os;
}
istream& operator>> (istream& is, MT& x)
{
    cout << " - Cap ma tran: " ;
    is >> x.n;
    cout << "Nhap cac phan tu :\n" ;
    for (int i=1 ; i<= x.n ; ++i)
        for (int j=1; j<=x.n; ++j)
        {
            cout << "PT" hang " << i << " cot " << j << " = " ;
            is >> x.a[i][j] ;
        }
    return is;
}
istream& operator>> (istream& is, VT& v)
{
    cout << " - Cap vec to: " ;
    is >> v.n;
    cout << "Nhap cac phan tu :\n" ;
    for (int i=1 ; i<= v.n ; ++i)
    {
        cout << "Phan tu thu " << i << " = " ;
        is >> v.b[i] ;
    }
    return is;
}
MT operator+(const MT& x1, const MT& x2)

```

```

{
    if (x1.n!=x2.n)
    {
        cout << "\nKhong thuc hien duoc phep cong vi 2 MT khong
cung cap";
        getch();
        return x1;
    }
    else
    {
        MT x;
        int i, j, n;
        n = x.n = x1.n ;
        for (i=1; i<=n; ++i)
            for (j=1; j<=n ;++j)
                x.a[i][j] = x1.a[i][j] + x2.a[i][j] ;
        return x;
    }
}

MT operator-(const MT& x1, const MT& x2)
{
    if (x1.n!= x2.n)
    {
        cout << "\nKhong thuc hien duoc phep tru vi 2 MT khong
cung cap";
        getch();
        return x1;
    }
    else
    {
        MT x;
        int i, j, n;
        n = x.n = x1.n;
        for (i=1; i<=n; ++i)
            for (j=1; j<=n ;++j)
                x.a[i][j] = x1.a[i][j] - x2.a[i][j] ;
        return x;
    }
}

MT operator*(const MT& x1, const MT& x2)
{
    if (x1.n!=x2.n)
    {
        cout << "\nKhong thuc hien duoc phep nhan vi 2 MT khong
cung cap";
        getch();
        return x1;
    }
    else
    {
        MT x;

```

```

    int n, i, j, k;
    n = x.n = x1.n;
    for (i=1; i<=n; ++i)
        for (j=1; j<=n; ++j)
        {
            x.a[i][j] = 0.0;
            for (k=1; k<=n; ++k)
                x.a[i][j] += x1.a[i][k]*x2.a[k][j];
        }
    return x;
}

VT operator*(const MT& x, const VT& v)
{
    if (x.n != v.n)
    {
        cout << "\n Cap ma tran khac cap vec to, phap nhan vo
nghia";
        getch();
        return v;
    }
    else
    {
        VT u; int n;
        n = u.n = v.n;
        for (int i=1; i <=n; ++i)
        {
            u.b[i] = 0;
            for (int j=1; j<=n; ++j)
                u.b[i] += x.a[i][j]*v.b[j];
        }
        return u;
    }
}

MT operator!(MT x)
{
    MT y;
    int i, j, k, r, n;
    double lg;
    n = y.n = x.n;
    for (i=1; i<=n; ++i)
        for (j=1; j<=n; ++j)
            if (i==j) y.a[i][j] = 1;
            else y.a[i][j] = 0;
    for (k=1; k<=n; ++k)
    {
        r=k;
        for (i=k+1; i<=n; ++i)
            if (abs(x.a[i][k]) > abs(x.a[r][k])) r = i;
        if (abs(x.a[r][k]) < 1.0E-8)
        {

```

```

        cout << "\n Ma tran suy bien, khong co nghich dao" ;
        getch(); return x;
    }
    /* Hoan vi hang r va hang k */
    for (j=1 ; j<=n ; ++j)
    {
        tg = x.a[k][j];
        x.a[k][j] = x.a[r][j];
        x.a[r][j] = tg;
        tg = y.a[k][j];
        y.a[k][j] = y.a[r][j];
        y.a[r][j] = tg;
    }
    /* Chia hang k cho a[k,k] */
    tg = x.a[k][k] ;
    for (j=1 ; j<=n ; ++j)
    {
        x.a[k][j] /= tg;
        y.a[k][j] /= tg;
    }
    /* Khu cot k : lam cho a[i,k] = 0 voi i != k */
    for (int i=1; i<= n ; ++i)
        if (i != k)
        {
            tg = x.a[i][k] ;
            for (j=1 ; j<= n ; ++j)
            {
                x.a[i][j] -= tg*x.a[k][j] ;
                y.a[i][j] -= tg*y.a[k][j] ;
            }
        }
    }
    return y;
}
void main()
{
    MT x, y, r, s;
    VT u, v;
    clrscr();
    cout << "\nNhap ma tran X " ; cin >> x;
    cout << "\nNhap ma tran Y " ; cin >> y;
    cout << "\nNhap ma tran R " ; cin >> r;
    cout << "\nNhap ma tran S " ; cin >> s;
    cout << "\nNhap vec to u " ; cin >> u;
    v = !((x + y) *(r - s)) * u;
    cout << "\nVec to v = xu " << v
    getch();
}

```

KHÁI NIỆM VỀ LỚP

Như đã nói ở trên, lớp là khái niệm trung tâm của lập trình hướng đối tượng, nó là sự mở rộng của các khái niệm cấu trúc (struct) của C và bản ghi (record) của PASCAL. Ngoài các thành phần dữ liệu (như cấu trúc), lớp còn chứa các thành phần hàm, còn gọi là phương thức (method) hay hàm thành viên (member function). Cũng giống như cấu trúc, lớp có thể xem như một kiểu dữ liệu. Vì vậy lớp còn gọi là kiểu đối tượng và lớp được dùng để khai báo các biến, mảng đối tượng (như thể dùng kiểu int để khai báo các biến mảng nguyên). Như vậy từ một lớp có thể tạo ra (bằng cách khai báo) nhiều đối tượng (biến, mảng) khác nhau. Mỗi đối tượng có vùng nhớ riêng của mình. Vì vậy cũng có thể quan niệm lớp là tập hợp các đối tượng cùng kiểu.

Chương này sẽ trình bày cách định nghĩa lớp, cách xây dựng phương thức, giải thích về phạm vi truy nhập, sử dụng các thành phần của lớp, cách khai báo biến, mảng cấu trúc, lời gọi tới các phương thức.

§1. ĐỊNH NGHĨA LỚP

1. Lớp được định nghĩa theo mẫu:

```
class Tên lớp
{
    // Khai báo các thuộc tính (peroperties)
    // Khai báo các phương thức (methods)
};
// Định nghĩa (xây dựng) các phương thức
```

Chú ý:

Thuộc tính của lớp có thể là các biến, mảng, con trỏ có kiểu chuẩn (int, float, char, char*, long,...) hoặc kiểu ngoài chuẩn đã định nghĩa trước (cấu trúc, hợp, lớp, ...). Thuộc tính của lớp không thể có kiểu của chính lớp đó, nhưng có thể là kiểu con trỏ lớp này, ví dụ:

```
class A
{
    A x; // Không cho phép, vì x có kiểu lớp A
```

```

    A *p; // Cho phép, vì p là con trỏ kiểu lớp A
    ...
};

```

2. Khi báo các thành phần của lớp (thuộc tính và phương thức) có thể dùng các từ khoá `private` và `public` để quy định phạm vi sử dụng của các thành phần. Nếu không quy định cụ thể (không dùng các từ khoá `private` và `public`) thì C++ hiểu đó là `private`.

Các thành phần `private` (riêng) chỉ được sử dụng bên trong lớp (trong thân của các phương thức của lớp). Các hàm không phải là phương thức của lớp không được phép sử dụng các thành phần này.

Các thành phần `public` (công cộng) được phép sử dụng ở cả bên trong và bên ngoài lớp.

3. Các thành phần dữ liệu thường (nhưng không bắt buộc) khai báo là `private` để bảo đảm tính giấu kín, bảo vệ an toàn dữ liệu của lớp, không cho phép các hàm bên ngoài xâm nhập vào dữ liệu của lớp.

4. Các phương thức thường khai báo là `public` để chúng có thể được gọi tới (sử dụng) từ các hàm khác trong chương trình.

5. Các phương thức có thể được xây dựng bên ngoài hoặc bên trong định nghĩa lớp. Thông thường, các phương thức ngắn được viết bên trong định nghĩa lớp, còn các phương thức dài thì viết bên ngoài định nghĩa lớp.

6. Trong thân phương thức của một lớp (giả sử lớp A) có thể sử dụng:

+ Các thuộc tính của lớp A

+ Các phương thức của lớp A

+ Các hàm tự lập trong chương trình. Vì phạm vi sử dụng của hàm là toàn chương trình.

7. Giá trị trả về của phương thức có thể có kiểu bất kỳ (chuẩn và ngoài chuẩn)

Ví dụ sau sẽ minh họa các điều nói trên. Chúng ta sẽ định nghĩa lớp để mô tả và xử lý các điểm trên màn hình đồ họa. Lớp được đặt tên là `DIEM`.

+ Các thuộc tính của lớp gồm:

`int x; // hoành độ (cột)`

`int y; // tung độ (hàng)`

`int m; // màu`

+ Các phương thức:

Nhập dữ liệu một điểm

Hiển thị một điểm

Ẩn một điểm

Lớp điểm được xây dựng như sau:

```
class DIEM
{
    private:
        int x, y, m;
    public:
        void nhapsl();
        void hien();
        void an()
        {
            putpixel(x, y, getbkcolor());
        }
};

void DIEM::nhap()
{
    cout << "\nNhập hoành độ (cột) và  
tung độ (hàng) của điểm: "
    cin >> x >> y;
    cout << "\nNhập mã màu của điểm: "
    cin >> m;
}

void DIEM::hien()
{
    int mau_ht;
    mau_ht = getcolor();
    putpixel(x, y, m);
    setcolor(mau_ht);
}
```

Qua ví dụ trên có thể rút ra một số điều cần nhớ sau:

- + Trong cả 3 phương thức (dù viết trong hay viết ngoài định nghĩa lớp) đều được phép truy nhập đến các thuộc tính x, y và m của lớp.
- + Các phương thức viết bên trong định nghĩa lớp (như phương thức an()) được viết như một hàm thông thường.
- + Khi xây dựng các phương thức bên ngoài lớp, cần dùng thêm tên lớp và toán tử phạm vi :: đặt ngay trước tên phương thức để quy định rõ đây là phương thức của lớp nào.

§2. BIẾN, MẢNG ĐỐI TƯỢNG

Như đã nói ở trên, một lớp (sau khi định nghĩa) có thể xem như một kiểu đối tượng và có thể dùng để khai báo các biến, mảng đối tượng. Cách khai báo biến, mảng đối tượng cũng giống như khai báo biến, mảng các kiểu khác (như int, float, cấu trúc, hợp, ...), theo mẫu sau:

```
Tên_lớp  danh_sách_đổi;  
Tên_lớp  danh_sách_mảng;
```

Ví dụ sử dụng lớp DIEM ở §1, có thể khai báo các biến, mảng DIEM như sau:

```
DIEM d1, d2, d3; // Khai báo 3 biến đối tượng d1, d2, d3  
DIEM d[20]; // Khai báo mảng đối tượng d gồm 20 phần tử
```

Mỗi đối tượng sau khi khai báo sẽ được cấp phát một vùng nhớ riêng để chứa các thuộc tính của chúng. Chú ý rằng sẽ không có vùng nhớ riêng để chứa các phương thức cho mỗi đối tượng. Các phương thức sẽ được sử dụng chung cho tất cả các đối tượng cùng lớp. Như vậy về bộ nhớ được cấp phát thì đối tượng giống cấu trúc. Trong trường hợp này:

```
sizeof(d1) = sizeof(d2) = sizeof(d3) = 3*sizeof(int) = 6  
sizeof(d) = 20*6 = 120
```

Thuộc tính của đối tượng:

Trong ví dụ trên, mỗi đối tượng d1, d2, d3 và mỗi phần tử d[i] đều có 3 thuộc tính là x, y, m. Chú ý là mỗi thuộc đều thuộc về một đối tượng, vì vậy không thể viết tên thuộc một cách riêng rẽ mà bao giờ cũng phải có tên đối tượng đi kèm, giống như cách viết trong cấu trúc của C hay bản ghi của PASCAL. Nói cách khác, cách viết thuộc tính của đối tượng như sau:

```
tên_đối_tượng.Tên_thuộc_tính
```

Với các đối tượng d1, d2, d3 và mảng d, có thể viết như sau:

```
d1.x    // Thuộc tính x của đối tượng d1  
d2.x    // Thuộc tính x của đối tượng d2  
d3.y    // Thuộc tính y của đối tượng d3  
d[2].m   // Thuộc tính m của phần tử d[2]  
d1.x = 100 ; // Gán 100 cho d1.x  
d2.y = d1.x; // Gán d1.x cho d2.y
```

Sử dụng các phương thức

Cũng giống như hàm, một phương thức được sử dụng thông qua lời gọi. Tuy nhiên trong lời gọi phương thức bao giờ cũng phải có tên đối tượng để chỉ rõ phương thức thực hiện trên các thuộc tính của đối tượng nào. Ví dụ lời gọi:

```
d1.nhap1();
```

sẽ thực hiện nhập số liệu vào các thành phần d1.x, d1.y và d1.m

Câu lệnh

```
d[3].nhap1() ;
```

sẽ thực hiện nhập số liệu vào các thành phần d[3].x, d[3].y và d[3].m

Chúng ta sẽ minh họa các điều nói trên bằng một chương trình đơn giản sử dụng lớp DIEM để nhập 3 điểm, hiện rồi ẩn các điểm vừa nhập. Trong chương trình đưa vào hàm kd_do_hoa() dùng để khởi động hệ đồ họa.

```
#include <conio.h>
#include <iostream.h>
#include <graphics.h>
class DIEM
{
private:
    int x, y, m ;
public:
    void nhapsl();
    void an()
    {
        putpixel(x,y,getbkcolor());
    }
    void hien();
};
void DIEM::nhapsl()
{
    cout << "\nNhap hoành đo (cot) va
            tung đo (hang) cua diem: " ;
    cin >> x >> y ;
    cout << " \nNhap ma mau cua diem: " ;
    cin >> m ;
}
void DIEM::hien()
{
    int mau_ht;
    mau_ht = getcolor() ;
    putpixel(x,y,m);
    setcolor(mau_ht);
}
void kd_do_hoa()
{
    int mh, mode ;
    mh=mode=0;
    initgraph(&mh, &mode, "");
}
void main()
{
    DIEM d1, d2, d3 ;
    d1.nhapsl();
    d2.nhapsl();
    d3.nhapsl();
    kd_do_hoa();
```

```

setbkcolor(BLACK);
d1.hien();
d2.hien();
d3.hien();
getch();
d1.an();
d2.an();
d3.an();
getch();
closegraph();
}

```

§3. CON TRỎ ĐỐI TƯỢNG

Con trỏ đối tượng dùng để chứa địa chỉ của biến, mảng đối tượng. Nó được khai báo như sau:

Tên_lớp *con trỏ ;

Ví dụ dùng lớp DIEM có thể khai báo:

```

DIEM *p1, *p2, *p3; // khai báo 3 con trỏ p1, p2, p3
DIEM d1, d2; // Khai báo 2 đối tượng d1, d2
DIEM d[20]; // Khai báo mảng đối tượng

```

và có thể thực hiện các câu lệnh:

```

p1 = &d2; // p1 chứa địa chỉ của d2, hay p1 trỏ tới d2
p2 = d; // p2 trỏ tới đầu mảng d
p3 = new DIEM // Tạo một đối tượng và chứa
               // địa chỉ của nó vào p3

```

Để sử dụng thuộc tính của đối tượng thông qua con trỏ, ta viết như sau:

Tên_con trỏ->Tên_thuộc_tính

Chú ý: Nếu con trỏ chứa địa chỉ đầu của mảng, có thể dùng con trỏ như tên mảng.

Như vậy sau khi thực hiện các câu lệnh trên thì:

```

p1->x và d2.x là như nhau
p2[i].y và d[i].y là như nhau

```

Tóm lại ta có quy tắc sau

Quy tắc sử dụng thuộc tính: Để sử dụng một thuộc tính của đối tượng ta phải dùng phép . hoặc phép -> . Trong chương trình, không cho phép viết tên thuộc tính một cách đơn độc mà phải đi kèm tên đối tượng hoặc tên con trỏ theo các mẫu sau:

```

Tên_đối_tượng.Tên_thuộc_tính
Tên_con trỏ->Tên_thuộc_tính

```

```
Tên_mảng_đối_tượng[chi_số].Tên_thuộc_tính
Tên_con_trò[chi_số].Tên_thuộc_tính
```

Chương trình dưới đây cũng sử dụng lớp DIEM (trong §1) để nhập một dãy điểm, hiển thị và ẩn các điểm vừa nhập. Chương trình dùng một con trỏ kiểu DIEM và dùng toán tử new để tạo ra một dãy đối tượng.

```
#include <conio.h>
#include <iostream.h>
#include <graphics.h>
class DIEM
{
private:
    int x, y, m ;
public:
    void nhapsl();
    void an()
    {
        putpixel(x,y,getbkcolor());
    }
    void hien();
};
void DIEM::nhapsl()
{
    cout << "\nNhập hoành độ (cot) và tung độ (hang) của điểm:" ;
    cin >> x >> y ;
    cout << " \nNhập mã màu của điểm: " ;
    cin >> m ;
}
void DIEM::hien()
{
    int mau_ht;
    mau_ht = getcolor() ;
    putpixel(x,y,m);
    setcolor(mau_ht);
}
void kd_do_hoa()
{
    int mh, mode ;
    mh=mode=0;
    initgraph(&mh, &mode, "");
}
void main()
{

```

```

DIEM *p;
int i, n;
cout << "So diem: ";
cin >> n;
p = new DIEM[n+1];
for (i=1; i<=n; ++i)
    p[i].nhapsl();
kd_do_hoa();
for (i=1; i<=n; ++i)
    p[i].hien();
getch();
for (i=1; i<=n; ++i)
    p[i].an();
getch();
closegraph();
}

```

54. ĐỐI CỦA PHƯƠNG THỨC, CON TRỎ this

4.1. Con trỏ this là đối thứ nhất của phương thức

Chúng ta hãy xem lại phương thức nhapsl của lớp DIEM

```

void DIEM::nhapsl()
{
    cout << "\nNhap hoành do (cot) va
           tung do (hang) cua diem:";
    cin >> x >> y;
    cout << " \nNhap ma mau cua diem: ";
    cin >> m;
}

```

Rõ ràng trong phương thức này chúng ta sử dụng tên các thuộc tính x, y và m một cách đơn độc. Điều này có vẻ như mâu thuẫn với quy tắc sử dụng thuộc tính nêu trong mục trước. Song sự thế như sau:

C++ sử dụng con trỏ đặc biệt **this** trong các phương thức. Các thuộc tính viết trong phương thức được hiểu là thuộc một đối tượng do con trỏ **this** trỏ tới. Như vậy phương thức nhapsl() có thể viết một cách tường minh như sau:

```

void DIEM::nhapsl()
{
    cout << "\nNhap hoành do (cot) va
           tung do (hang) cua diem:";

```

```

cin >> this->x >> this->y;
cout << " \nNhap ma mau cua diem: ";
cin >> this->m;
}

```

Từ góc độ hàm số có thể kết luận rằng: Phương thức bao giờ cũng có ít nhất một đối là con trỏ this và nó luôn luôn là đối đầu tiên của phương thức.

4.2. Tham số ứng với đối con trỏ this

Xét một lời gọi tới phương thức nhapsl() :

```

DIEM d1;
d1.nhapsl();

```

Trong trường hợp này tham số truyền cho con trỏ this chính là địa chỉ của d1:

```
this = &d1
```

Do đó:

```

this->x chính là d1.x
this->y chính là d1.y
this->m chính là d1.m

```

Như vậy câu lệnh

```
d1.nhapsl();
```

sẽ nhập dữ liệu cho các thuộc tính của đối tượng d1. Từ đó có thể rút ra kết luận sau:

Tham số truyền cho đối con trỏ this chính là địa chỉ của đối tượng đi kèm với phương thức trong lời gọi phương thức.

4.3. Các đối khác của phương thức

Ngoài đối đặc biệt this (đối này không xuất hiện một cách tường minh), phương thức còn có các đối khác được khai báo như trong các hàm. Đối của phương thức có thể có kiểu bất kỳ (chuẩn và ngoài chuẩn).

Ví dụ để xây dựng phương thức vẽ đường thẳng qua 2 điểm ta cần đưa vào 3 đối: Hai đối là 2 biến kiểu DIEM, đối thứ ba kiểu nguyên xác định mã màu. Vì đã có đối ngầm định this là đối thứ nhất, nên chỉ cần khai báo thêm 2 đối. Phương thức có thể viết như sau:

```

void DIEM::doan_thang(DIEM d2, int mau)
{
    int mau_ht;
    mau_ht = getcolor();
    setcolor(mau);
    line(this->x, this->y, d2.x, d2.y);
}

```

```

        setcolor(mau_ht);
    }

```

Chương trình sau minh họa các phương thức có nhiều đối. Ta vẫn dùng lớp DIEM nhưng có một số thay đổi:

- + Bỏ thuộc tính m (màu)
- + Bỏ các phương thức hien và an
- + Đưa vào 4 phương thức mới:
 - ve_doan_thang (Vẽ đoạn thẳng qua 2 điểm)
 - ve_tam_giac (Vẽ tam giác qua 3 điểm)
 - do_dai (Tính độ dài của đoạn thẳng qua 2 điểm)
 - chu_vi (Tính chu vi tam giác qua 3 điểm)

Chương trình còn minh họa:

- + Việc phương thức này sử dụng phương thức khác (phương thức ve_tam_giac sử dụng phương thức ve_doan_thang, phương thức chu_vi sử dụng phương thức do_dai)

- + Sử dụng con trỏ this trong thân các phương thức ve_tam_giac và chu_vi

Nội dung chương trình là nhập 3 điểm, vẽ tam giác có đỉnh là 3 điểm vừa nhập sau đó tính chu vi tam giác.

```

#include <conio.h>
#include <iostream.h>
#include <graphics.h>
#include <math.h>
#include <stdio.h>
class DIEM
{
    private:
        int x, y ;
    public:
        void nhapsl();
        void ve_doan_thang(DIEM d2, int mau) ;
        void ve_tam_giac(DIEM d2, DIEM d3, int mau) ;
        double do_dai(DIEM d2)
        {
            DIEM d1 = *this ;
            return sqrt( pow(d1.x - d2.x,2) +
                          pow(d1.y - d2.y,2) ) ;
        }
        double chu_vi(DIEM d2, DIEM d3);
};
void DIEM::nhapsl()
{

```

```

    cout <<" \nNhap hoành do (cot) va
           tung do (hang) cua diem:" ;
    cin >> x >> y ;
}
void kd_do_hoa()
{
    int mh, mode ;
    mh=mode=0;
    initgraph(&mh, &mode, "");
}
void DIEM::ve_doan_thang(DIEM d2, int mau)
{
    setcolor(mau);
    line(this->x,this->y,d2.x,d2.y);
}
void DIEM::ve_tam_giac(DIEM d2, DIEM d3,int mau)
{
    (*this).ve_doan_thang(d2,mau);
    d2.ve_doan_thang(d3,mau);
    d3.ve_doan_thang(*this,mau);
}
double DIEM::chu_vi(DIEM d2, DIEM d3)
{
    double s;
    s=(*this).do_dai(d2)+d2.do_dai(d3)+d3.do_dai(*this);
    return s;
}
void main()
{
    DIEM d1, d2, d3;
    char tb_cv[20] ;
    d1.nhapsl();
    d2.nhapsl();
    d3.nhapsl();
    kd_do_hoa();
    d1.ve_tam_giac(d2,d3,15);
    double s = d1.chu_vi(d2,d3);
    sprintf(tb_cv,"Chu vi = %0.2f", s);
    outtextxy(10,10,tb_cv);
    getch();
    closegraph();
}

```

Một số nhận xét về đối của phương thức và lời gọi phương thức

+ Quan sát nguyên mẫu phương thức:

```
void ve_doan_thang(DIEM d2, int mau) ;
```

sẽ thấy phương thức có 3 đối:

Đối thứ nhất là một đối tượng DIEM do this trỏ tới

Đối thứ hai là đối tượng DIEM d2

Đối thứ ba là biến nguyên màu

Nội dung phương thức là vẽ một đoạn thẳng đi qua các điểm *this và d2 theo mã màu màu. Xem thân của phương sẽ thấy được nội dung này:

```
void DIEM::ve_doan_thang(DIEM d2, int mau)
{
    setcolor(mau);
    line(this->x, this->y, d2.x, d2.y);
}
```

Tuy nhiên trong trường hợp này, vai trò của this không cao lắm, vì nó được đưa vào chỉ cốt làm rõ đối thứ nhất. Trong thân phương thức có thể bỏ từ khoá this vẫn được.

+ Vai trò của this trở nên quan trọng trong phương thức ve_tam_giac:

```
void ve_tam_giac(DIEM d2, DIEM d3, int mau) ;
```

Phương thức này có 4 đối là:

this trỏ tới một đối tượng kiểu DIEM

d2 một đối tượng kiểu DIEM

d3 một đối tượng kiểu DIEM

mau một biến nguyên

Nội dung phương thức là vẽ 3 cạnh:

cạnh 1 đi qua *this và d2

cạnh 2 đi qua d2 và d3

cạnh 3 đi qua d3 và *this

Các cạnh trên được vẽ nhờ sử dụng phương thức ve_doan_thang:

Vẽ cạnh 1 dùng lệnh: (*this).ve_doan_thang(d2,mau);

Vẽ cạnh 2 dùng lệnh: d2.ve_doan_thang(d3,mau);

Vẽ cạnh 3 dùng lệnh: d3.ve_doan_thang(*this,mau);

Trong trường hợp này rõ ràng vai trò của this rất quan trọng. Nếu không dùng nó thì công việc trở nên khó khăn, dài dòng và khó hiểu hơn. Chúng ta hãy so sánh 2 phương án:

Phương án dùng this trong phương thức ve_tam_giac:

```
void DIEM::ve_tam_giac(DIEM d2, DIEM d3, int mau)
{
    (*this).ve_doan_thang(d2,mau);
    d2.ve_doan_thang(d3,mau);
    d3.ve_doan_thang(*this,mau);
}
```

Phương án không dùng this trong phương thức ve_tam_giac:

```

void DIEM::ve_tam_giac(DIEM d2, DIEM d3, int mau)
{
    DIEM d1;
    d1.x = x;
    d1.y = y;
    d1.ve_doan_thang(d2, mau);
    d2.ve_doan_thang(d3, mau);
    d3.ve_doan_thang(d1, mau);
}

```

§5. NÓI THÊM VỀ KIỂU PHƯƠNG THỨC VÀ KIỂU ĐỐI CỦA PHƯƠNG THỨC

5.1. Kiểu phương thức

Phương thức có thể không có giá trị trả về (kiểu void) hoặc có thể trả về một giá trị có kiểu bất kỳ, kể cả giá trị kiểu đối tượng, con trỏ đối tượng, tham chiếu đối tượng.

5.2. Đối của phương thức

Đối của phương thức (cũng giống như đối của hàm) có thể có kiểu bất kỳ:

- + Kiểu dữ liệu chuẩn như int, float, char, ... Con trỏ hoặc tham chiếu đến kiểu dữ liệu chuẩn như int*, float*, char*, int&, float&, char&, ...

- + Các kiểu ngoài chuẩn đã định nghĩa trước như đối tượng, cấu trúc, hợp, enum, ... Con trỏ hoặc tham chiếu đến các kiểu ngoài chuẩn này.

- + Kiểu đối tượng của chính phương thức, con trỏ hoặc tham chiếu đến kiểu đối tượng này.

5.3. Các ví dụ

Ví dụ 1 minh họa:

- + Thuộc tính (thành phần dữ liệu) của lớp có thể là đối tượng của lớp khác đã định nghĩa bên trên.

- + Phương thức có giá trị trả về kiểu đối tượng và con trỏ đối tượng.

Nội dung chương trình là nhập một dãy hình chữ nhật, sau đó tìm hình chữ nhật có max diện tích và hình chữ nhật có max chu vi.

Chương trình được tổ chức thành 2 lớp:

- + Lớp HINH_CN gồm:

- Các thuộc tính: d và r (chiều dài và chiều rộng)

- Các phương thức

```
void nhapsl(); // Nhập chiều dài, rộng
```

```
int dien_tich(); // Tính diện tích
```

```

        int chu_vi(); // Tính chu vi
+ Lớp DAY_HINH_CN gồm
- Các thuộc tính:
    int n; // số hình chữ nhật của dãy
    HINH_CN *h; // Con trỏ tới dãy đối tượng của lớp HINH_CN
- Các phương thức
    void nhapsl(); // Nhập một dãy hình chữ nhật
    HINH_CN hình_dt_max(); // Trả về hình chữ nhật có
        // diện tích max
    HINH_CN *hình_cv_max(); // Trả về con trỏ tới HCN có
        // chu vi max

#include <conio.h>
#include <iostream.h>
class HINH_CN
{
private:
    int d, r; // chiều dài và chiều rộng
public:
    void nhapsl()
    {
        cout << " \nNhập chiều dài và chiều rộng: " ;
        cin >> d >> r ;
    }
    void in()
    {
        cout << "\nchiều dài = " << d ;
        cout << " chiều rộng: " << r;
    }
    int dien_tich()
    {
        return d*r;
    }
    int chu_vi()
    {
        return 2*(d+r);
    }
} ;
class DAY_HINH_CN
{
private:
    int n; // Số hình chữ nhật
    HINH_CN *h;
public:
    void nhapsl();
    HINH_CN hình_dt_max() ;

```

```

        HINH_CN *hinh_cv_max() ;
    } ;
void DAY_HINH_CN::nhapsl()
{
    cout << "So hinh CN = " ;
    cin >> n;
    h = new HINH_CN[n+1];
    for (int i=1; i<=n; ++i)
        h[i].nhapsl();
}
HINH_CN DAY_HINH_CN::hinh_dt_max()
{
    HINH_CN hdtmax;
    hdtmax = h[1];
    for (int i=2; i<=n; ++i)
        if (h[i].dien_tich() > hdtmax.dien_tich() )
            hdtmax = h[i];
    return hdtmax;
}
HINH_CN *DAY_HINH_CN::hinh_cv_max()
{
    int imax = 1;
    for (int i=2; i<=n; ++i)
        if (h[i].chu_vi() > h[imax].chu_vi() )
            imax = i ;
    return (h[imax]);
}
void main()
{
    DAY_HINH_CN d;
    HINH_CN hdtmax;
    d.nhapsl();
    hdtmax = d.hinh_dt_max();
    hdtmax.in() ;
    HINH_CN *hcvmax=d.hinh_cv_max();
    hcvmax->in() ;
    getch();
}

```

Ví dụ 2 minh họa:

+ Thuộc tính (thành phần dữ liệu) của lớp có thể là đối tượng của lớp khác đã định nghĩa bên trên.

+ Phương thức có giá trị trả về kiểu đối tượng

+ Vai trò của con trỏ this (xem phương thức maxdt của lớp TAM_GIAC)

+ Phương thức tĩnh (xem phương thức tao_tg của lớp TAM_GIAC)

Nội dung chương trình là nhập một dãy các điểm, sau đó tìm tam giác lớn nhất (về diện tích) có đỉnh là các điểm vừa nhập.

Chương trình được tổ chức thành 2 lớp:

+ Lớp DIEM gồm:

- Các thuộc tính: x và y (toạ độ của điểm)

- Các phương thức

```
void nhapsl(); // Nhập x, y
```

```
void in(); // In toạ độ
```

```
double do_dai(DIEM d2); // Tính độ dài đoạn thẳng qua
```

```
// 2 điểm(điểm ẩn xác định
```

```
// bởi this và điểm d2)
```

+ Lớp TAM_GIAC gồm:

- Các thuộc tính:

```
DIEM d1,d2,d3; // 3 đỉnh của tam giác
```

- Các phương thức:

```
void nhapsl(); // Nhập toạ độ 3 đỉnh
```

```
void in(); // In toạ độ 3 đỉnh
```

```
// Tạo một đối tượng TAM_GIAC từ 3 đối tượng DIEM
```

```
static TAM_GIAC tao_tg(DIEM e1, DIEM e2, DIEM e3)
```

```
double dien_tich(); // Tính diện tích
```

```
// Tìm tam giác có diện tích max trong 2 tam giác *this và t2
```

```
TAM_GIAC maxdt(TAM_GIAC t2);
```

+ Các vấn đề đáng chú ý trong chương trình là:

- Phương thức tĩnh tao_tg (sẽ giải thích bên dưới)

- Phương thức maxdt

+ Thuật toán là:

- Duyệt qua các tổ hợp 3 điểm.

- Dùng phương thức tao_tg để lập tam giác từ 3 điểm

- Dùng phương thức maxdt để chọn tam giác có diện tích lớn hơn trong 2 tam giác: tam giác vừa tạo và tam giác có diện tích max (trong số các tam giác đã tạo)

```
#include <conio.h>
```

```
#include <iostream.h>
```

```
#include <math.h>
```

```
class DIEM
```

```
{
```

```
private:
```

```
double x,y; // Toa do cua diem
```

```

public:
    void nhapsl()
    {
        cout << " Toa do x, y: " ;
        cin >> x >> y ;
    }
    void in()
    {
        cout << " x = " << x << " y = " << y;
    }
    double do_dai(DIEM d2)
    {
        return sqrt(pow(x-d2.x,2) + pow(y-d2.y,2) );
    }
};

class TAM_GIAC
{
private:
    DIEM d1,d2,d3; // 3 dinh tam giac
public:
    void nhapsl();
    void in();
    static TAM_GIAC tao_tg(DIEM e1, DIEM e2, DIEM e3)
    {
        TAM_GIAC t;
        t.d1=e1; t.d2 = e2; t.d3=e3;
        return t;
    }
    double dien_tich();
    TAM_GIAC maxdt(TAM_GIAC t2);
};

void TAM_GIAC::nhapsl()
{
    cout << "\nDinh 1 - " ;
    d1.nhapsl();
    cout << "\nDinh 2 - " ;
    d2.nhapsl();
    cout << "\nDinh 3 - " ;
    d3.nhapsl();
}

void TAM_GIAC::in()
{
    cout << "\nDinh 1: " ; d1.in();
    cout << "\nDinh 2: " ; d2.in();
    cout << "\nDinh 3: " ; d3.in();
}

double TAM_GIAC::dien_tich()
{
    double a,b,c,p,s;
    a=d1.do_dai(d2);
    b=d2.do_dai(d3);

```

```

        c=d3.do_dai{d1};
        p=(a+b+c)/2;
        return sqrt(p*(p-a)*(p-b)*(p-c));
    }
TAM_GIAC TAM_GIAC::maxdt(TAM_GIAC t2)
{
    if (this->dien_tich() > t2.dien_tich())
        return *this;
    else
        return t2;
}
void main()
{
    DIEM d[50];
    int n, i;
    clrscr();
    cout << "\n So diem= ";
    cin >> n;
    for (i=1; i<=n; ++i)
    {
        cout << "\nNhap diem " << i << " - ";
        d[i].nhapsl();
    }
    int j, k;
    TAM_GIAC tmax, t;
    tmax = TAM_GIAC::tao_tg(d[1],d[2],d[3]);
    for (i=1; i<=n-2; ++i)
        for (j=i+1; j<=n-1; ++j)
            for (k=j+1; k<=n; ++k)
            {
                t=TAM_GIAC::tao_tg(d[i],d[j],d[k]);
                tmax = tmax.maxdt(t);
            }
    cout << "\n\nTam giac co dien tich lon nhat: ";
    tmax.in();
    cout << "\nDien tich = " << tmax.dien_tich();
    getch();
}

```

Chú ý 1: Để tạo một đối tượng TAM_GIAC từ 3 đối tượng DIEM ta đã dùng phương thức tĩnh:

```

static TAM_GIAC tao_tg(DIEM e1, DIEM e2, DIEM e3)
{
    TAM_GIAC t;
    t.d1=e1; t.d2 = e2; t.d3=e3;
    return t;
}

```

Phương thức tĩnh (sẽ nói thêm trong các mục bên dưới) có các đặc điểm sau:

+ Nó giống phương thức thông thường ở chỗ: Trong thân của nó có thể truy nhập tới các thành phần của lớp (cụ thể là lớp TAM_GIAC).

+ Nó khác phương thức thông thường ở chỗ:

- Không có đối ngầm định xác định bởi con trỏ this (như phương thức thông thường). Như vậy phương thức tao_tg có đúng 3 đối.

- Nó không gắn với một đối tượng cụ thể nào của lớp, nên trong lời gọi tới phương thức ảo có thể dùng tên lớp, ví dụ (xem hàm main):

```
t--TAM_GIAC::tao_tg(d[i],d[j],d[k]);
```

Chú ý 2: Không thể thay phương thức tĩnh tao_tg bằng hàm, vì trong thân hàm không được truy xuất đến các thuộc tính của lớp TAM_GIAC. Tuy nhiên có một giải pháp khác là dùng khái niệm hàm bạn (friend). Hàm bạn của một lớp có quyền truy nhập đến các thuộc tính của lớp. Trong ví dụ 3 dưới đây ta sẽ xây dựng hàm tao_tg như một hàm bạn của lớp TAM_GIAC.

Chú ý 3: còn một giải pháp nữa là dùng hàm tạo (constructor) sẽ trình bày trong các chương sau:

Chương trình dưới đây có nội dung giống như ví dụ 2, nhưng thay phương thức tĩnh tao_tg bằng hàm bạn tao_tg.

Ví dụ 3: Minh họa cách dùng hàm bạn. Nội dung chương trình giống như trong ví dụ 2.

```
#include <conio.h>
#include <iostream.h>
#include <math.h>
class DIEM
{
private:
    double x,y; // Toa do cua diem
public:
    void nhapsl()
    {
        cout << " Toa do x, y: " ;
        cin >> x >> y ;
    }
    void in()
    {
        cout << " x = " << x << " y = " << y;
    }
    double do_dai(DIEM d2)
    {
        return sqrt(pow(x-d2.x,2) + pow(y-d2.y,2) );
    }
};
class TAM_GIAC
```

```

{
    private:
        DIEM d1,d2,d3; // 3 đỉnh tam giác
    public:
        void nhapsl();
        void in();
        friend TAM_GIAC tao_tg(DIEM e1, DIEM e2, DIEM e3)
        {
            TAM_GIAC t;
            t.d1=e1; t.d2 = e2; t.d3=e3;
            return t;
        }
        double dien_tich() ;
        TAM_GIAC maxdt(TAM_GIAC t2);
    } ;
void TAM_GIAC::nhapsl()
{
    cout << "\nĐỉnh 1 - " ;
    d1.nhapsl();
    cout << "\nĐỉnh 2 - " ;
    d2.nhapsl();
    cout << "\nĐỉnh 3 - " ;
    d3.nhapsl();
}
void TAM_GIAC::in()
{
    cout << "\nĐỉnh 1: " ; d1.in();
    cout << "\nĐỉnh 2: " ; d2.in();
    cout << "\nĐỉnh 3: " ; d3.in();
}
double TAM_GIAC::dien_tich()
{
    double a,b,c,p,s;
    a=d1.do_dai(d2);
    b=d2.do_dai(d3);
    c=d3.do_dai(d1);
    p=(a+b+c)/2;
    return sqrt(p*(p-a)*(p-b)*(p-c));
}
TAM_GIAC TAM_GIAC::maxdt(TAM_GIAC t2)
{
    if (this->dien_tich() > t2.dien_tich())
        return *this ;
    else
        return t2;
}
void main()
{
    DIEM d[50];
    int n, i ;

```

```

clrscr();
cout << "\n So diem= ";
cin >> n;
for (j=1; i<=n; ++i)
{
    cout << "\nNhap diem " << i << " - " ;
    d[i].nhapsl();
}
int j, k ;
TAM_GIAC tmax, t;
tmax = tao_tg(d[1],d[2],d[3]);
for (i=1;i<=n-2;++i)
    for (j=i+1;j<=n-1;++j)
        for (k=j+1;k<=n;++k)
        {
            t=tao_tg(d[i],d[j],d[k]);
            tmax = tmax.maxdt(t);
        }
cout << "\n\nTam giac co dien tich lon nhat: " ;
tmax.in();
cout << "\nDien tich = " << tmax.dien_tich();
getch();
}

```

Chú ý: Hàm bạn có thể xây dựng bên trong định nghĩa lớp (như chương trình trên) hoặc có thể khai báo bên trong và xây dựng bên ngoài định nghĩa lớp như sau:

```

class TAM_GIAC
{
private:
    DIEM d1,d2,d3; // 3 đỉnh tam giác
public:
    void nhapsl();
    void in();
    friend TAM_GIAC tao_tg(DIEM e1,DIEM e2,DIEM e3);
    double dien_tich();
    TAM_GIAC maxdt(TAM_GIAC t2);
};

TAM_GIAC tao_tg(DIEM e1, DIEM e2, DIEM e3)
{
    TAM_GIAC t;
    t.d1=e1; t.d2 = e2; t.d3=e3;
    return t;
}

```

Nhận xét: Không cho phép dùng từ khoá friend khi xây dựng hàm (bên ngoài lớp)

§6. HÀM, HÀM BẠN

6.1. Hàm có các tính chất sau:

+ Phạm vi của hàm là toàn bộ chương trình, vì vậy hàm có thể được gọi tới từ bất kỳ chỗ nào. Như vậy trong các phương thức có thể sử dụng hàm.

+ Đối của hàm có thể là các đối tượng, tuy nhiên có một hạn chế là trong thân hàm không cho phép truy nhập tới thuộc tính của các đối này. Ví dụ giả sử đã định nghĩa lớp:

```
class DIEM
{
    private:
        double x,y; // Toa do cua diem
    public:
        void nhapsl()
        {
            cout << " Toa do x, y: " ;
            cin >> x >> y ;
        }
        void in()
        {
            cout << " x = " << x << " y = " << y;
        }
};
```

Dùng lớp DIEM, ta xây dựng hàm tính độ dài của đoạn thẳng đi qua 2 điểm như sau:

```
double do_dai(DIEM d1, DIEM d2)
{
    return sqrt(pow(d1.x-d2.x,2) + pow(d1.y-d2.y,2));
}
```

Hàm này sẽ bị báo lỗi khi dịch, vì trong thân hàm không cho phép sử dụng các thuộc tính d1.x, d1.y, d2.x, d2.y của các đối tượng d1 và d2 thuộc lớp DIEM.

+ Phạm vi sử dụng của các phương thức (public) là toàn chương trình, vì vậy trong thân hàm có thể gọi tới các phương thức. Ví dụ giả sử đã định nghĩa lớp:

```
class DIEM
{
    private:
        double x,y; // Toa do cua diem
    public:
        void nhapsl()
```

```

    {
        cout << " Toa do x, y: " ;
        cin >> x >> y ;
    }
void in()
{
    cout << " x = " << x << " y = " << y;
}
double do_dai(DIEM d2)
{
    return sqrt(pow(x-d2.x,2) + pow(y-d2.y,2) );
}
} ;

```

Khi đó bằng cách dùng phương thức `do_dai`, ta có thể viết hàm tính diện tích tam giác có đỉnh là các đối tượng `d1`, `d2`, `d3` của lớp `DIEM` như sau:

```

double dt_tg(DIEM d1, DIEM d2, DIEM d3)
{
    double a,b,c,p,s;
    a=d1.do_dai(d2);
    b=d2.do_dai(d3);
    c=d3.do_dai(d1);
    p=(a+b+c)/2;
    return sqrt(p*(p-a)*(p-b)*(p-c));
}

```

Bằng cách dùng hàm `dt_tg`, có thể tổ chức lại chương trình tìm tam giác có diện tích lớn nhất (ở mục trên) một cách đơn giản hơn(bỏ đi lớp `TAM_GIAC`) như ví dụ sau.

Ví dụ 1:

```

#include <conio.h>
#include <iostream.h>
#include <math.h>
class DIEM
{
private:
    double x,y; // Toa do cua diem
public:
    void nhapsl()
    {
        cout << " Toa do x, y: " ;
        cin >> x >> y ;
    }
    void in()
    {
        cout << " x = " << x << " y = " << y;
    }
}

```

```

    }
    double do_dai(DIEM d2)
    {
        return sqrt(pow(x-d2.x,2) + pow(y-d2.y,2) );
    }
};

double dt_tg(DIEM d1, DIEM d2, DIEM d3)
{
    double a,b,c,p,s;
    a=d1.do_dai(d2);
    b=d2.do_dai(d3);
    c=d3.do_dai(d1);
    p=(a+b+c)/2;
    return sqrt(p*(p-a)*(p-b)*(p-c));
}

void main()
{
    DIEM d[50];
    int n, i, j, k, imax, jmax, kmax ;
    clrscr();
    cout << "\n So diem= ";
    cin >> n;
    for (i = 1; i <= n; ++i)
    {
        cout << "\nNhap diem " << i << " - ";
        d[i].nhaps1();
    }
    imax=1; jmax=2; kmax=3;
    for (i=1; i<=n-2; ++i)
        for (j=i+1; j<=n-1; ++j)
            for (k=j+1; k<=n; ++k)
                if (dt_tg(d[i],d[j],d[k]) >
                    dt_tg(d[imax],d[jmax],d[kmax]))
                {
                    imax = i ;
                    jmax = j;
                    kmax = k;
                }
    cout << "\n\nTam giac co dien tich lon nhat: " ;
    cout << "\nDinh 1 - "; d[imax].in();
    cout << "\nDinh 2 - "; d[jmax].in();
    cout << "\nDinh 3 - "; d[kmax].in();
    cout << "\nDien tich = " <<
        dt_tg(d[imax],d[jmax],d[kmax]) ;
    getch();
}

```

Nhận xét: Chương trình trên làm việc trên mảng d kiểu DIEM. Bây giờ nếu ta dùng mảng ngoài thì từ số thứ tự sẽ suy ra phần tử của mảng. Như vậy hàm

```
double dt_tg(DIEM d1, DIEM d2, DIEM d3);
```

có 3 đối kiểu DIEM có thể thay bằng hàm có 3 đối nguyên:

```
double dt_tg(int i, int j, int k);
```

để tính diện tích tam giác có đỉnh là $d[i]$, $d[j]$ và $d[k]$. Ý tưởng này được thể hiện trong ví dụ sau.

Ví dụ 2: Chương trình dùng mảng đối tượng ngoài.

Chú ý: Khai báo mảng đối tượng phải đặt sau định nghĩa kiểu đối tượng (định nghĩa lớp).

```
#include <conio.h>
#include <iostream.h>
#include <math.h>
double dt_tg(int i, int j, int k); // Khai báo hàm dt_tg
class DIEM
{
private:
    double x,y; // Toa do cua diem
public:
    void nhapsl();
    void in();
    double do_dai(DIEM d2);
};
// Chú ý: Khai báo mảng kiểu DIEM phải đặt sau
// định nghĩa lớp DIEM
DIEM d[50];
void DIEM::nhapsl()
{
    cout << " Toa do x, y: ";
    cin >> x >> y;
}
void DIEM::in()
{
    cout << " x = " << x << " y = " << y;
}
double DIEM::do_dai(DIEM d2)
{
    return sqrt(pow(x-d2.x,2) + pow(y-d2.y,2));
}
double dt_tg(int i, int j, int k)
{
    double a,b,c,p,s;
    a=d[i].do_dai(d[j]);
    b=d[j].do_dai(d[k]);
    c=d[k].do_dai(d[i]);
    p=(a+b+c)/2;
    return sqrt(p*(p-a)*(p-b)*(p-c));
}
void main()
```

```

{
    int n, i, j, k, imax, jmax, kmax ;
    clrscr();
    cout << "\n So diem= ";
    cin >> n;
    for (i=1; i<=n; ++i)
    {
        cout << "\nNhap diem " << i << " - " ;
        d[i].nhaps1();
    }
    imax=1; jmax=2; kmax=3;
    for (i=1; i<=n-2; ++i)
        for (j=i+1; j<=n-1; ++j)
            for (k=j+1; k<=n; ++k)
                if (dt_tg(i, j, k) > dt_tg(imax, jmax, kmax))
                {
                    imax = i ;
                    jmax = j;
                    kmax = k;
                }
    cout << "\n\nTam giac co dien tich lon nhat: " ;
    cout << "\nDinh 1 - "; d[imax].in();
    cout << "\nDinh 2 - "; d[jmax].in();
    cout << "\nDinh 3 - "; d[kmax].in();
    cout << "\nDien tich = " << dt_tg(imax, jmax, kmax);
    getch();
}

```

6.2. Hàm bạn (friend function)

6.2.1. Để một hàm trở thành bạn của một lớp, có 2 cách viết:

Cách 1: Dùng từ khoá friend để khai báo hàm trong lớp và xây dựng hàm bên ngoài như các hàm thông thường (không dùng từ khoá friend).
Mẫu viết như sau:

```

class A
{
    private:
        // Khai báo các thuộc tính
    public:
        ...
        // Khai báo các hàm bạn của lớp A
        friend void f1(...);
        friend double f2(...);
        friend A f3(...) ;
        ...
};
// Xây dựng các hàm f1, f2, f3
void f1(...)

```

```

{
    ...
}
double f2(...)
{
    ...
}
A f3(...)
{
    ...
}

```

Cách 2: Dùng từ khoá friend để xây dựng hàm trong định nghĩa lớp. Mẫu viết như sau:

```

class A
{
private:
    // Khai báo các thuộc tính
public:
    ...
    // Xây dựng các hàm bạn của lớp A
    friend void f1(...)
    {
        ...
    }
    friend double f2(...)
    {
        ...
    }
    friend A f3(...)
    {
        ...
    }
    ...
} ;

```

6.2.2. Tính chất của hàm bạn

Trong thân hàm bạn của một lớp có thể truy nhập tới các thuộc tính của các đối tượng thuộc lớp này. Đây là sự khác nhau duy nhất giữa hàm bạn và hàm thông thường. Chú ý rằng hàm bạn không phải là phương thức của lớp. Phương thức có một đối ẩn (ứng với con trỏ this) và lời gọi của phương thức phải gắn với một đối tượng nào đó (địa chỉ đối tượng này được truyền cho con trỏ this). Lời gọi của hàm bạn giống như lời gọi của hàm thông thường.

Ví dụ sau sẽ so sánh phương thức, hàm bạn và hàm tự do (hàm thông thường). Xét lớp SP (số phức). Hãy so sánh 3 phương án để thực hiện việc cộng 2 số phức:

Phương án 1: Dùng phương thức

```
class SP
{
    private:
        double a; // Phần thực
        double b; // Phần ảo
    public:
        SP cong(SP u2)
        {
            SP u;
            u.a = this->a + u2.a ;
            u.b = this->b + u2.b ;
            return u;
        }
};
```

Cách dùng

```
SP u, u1, u2;
u = u1.cong(u2);
```

Phương án 2: Dùng hàm bạn

```
class SP
{
    private:
        double a; // Phần thực
        double b; // Phần ảo
    public:
        friend SP cong(SP u1, SP u2)
        {
            SP u;
            u.a = u1.a + u2.a ;
            u.b = u1.b + u2.b ;
            return u;
        }
};
```

Cách dùng

```
SP u, u1, u2;
u = cong(u1, u2);
```

Phương án 3: Dùng hàm tự do

```
class SP
{
    private:
        double a; // Phần thực
        double b; // Phần ảo
    public:
        ...
};

SP cong(SP u1, SP u2)
```

```

{
    SP u:
    u.a = u1.a + u2.a ;
    u.b = u1.b + u2.b ;
    return u;
}

```

Phương án này không được chấp nhận, Trình biên dịch sẽ báo lỗi vì trong thân hàm không được quyền truy xuất đến các thuộc tính riêng (private) a, b của các đối tượng u, u1 và u2 thuộc lớp SP.

6.2.3. Một hàm có thể là bạn của nhiều lớp được không?

Câu trả lời là được. Khi một hàm là bạn của nhiều lớp, thì nó có quyền truy nhập tới tất cả các thuộc tính của các đối tượng trong các lớp này. Để làm cho hàm f trở thành bạn của các lớp A, B và C ta sử dụng mẫu viết sau:

```

class A; // Khai báo trước lớp A
class B; // Khai báo trước lớp B
class C; // Khai báo trước lớp C
// Định nghĩa lớp A
class A
{
    // Khai báo f là bạn của A
    friend void f(...) ;
};
// Định nghĩa lớp B
class B
{
    // Khai báo f là bạn của B
    friend void f(...) ;
};
// Định nghĩa lớp C
class C
{
    // Khai báo f là bạn của C
    friend void f(...) ;
};
// Xây dựng hàm f
void f(...)
{
    ...
}

```

Chương trình sau đây minh họa cách dùng hàm bạn (bạn của một lớp và bạn của nhiều lớp). Chương trình đưa vào 2 lớp VT (véc tơ), MT (ma trận) và 3 hàm bạn để thực hiện các thao tác trên 2 lớp này:

```

// Hàm bạn với lớp VT dùng để in một véc tơ
friend void in(const VT &x);
// Hàm bạn với lớp MT dùng để in một ma trận
friend void in(const MT &a);

```

// Hàm bạn với cả 2 lớp MT và VT dùng để nhân ma trận với véc tơ

```
friend VT lich(const MT &a,const VT &x);
```

Nội dung chương trình là nhập một ma trận vuông cấp n và một véc tơ cấp n, sau đó thực hiện phép nhân ma trận với véc tơ vừa nhập.

```
// Chương trình CT3_09.CPP
```

```
#include <conio.h>
```

```
#include <iostream.h>
```

```
#include <math.h>
```

```
class VT;
```

```
class MT ;
```

```
class VT
```

```
{
```

```
private:
```

```
int n;
```

```
double x[20];
```

```
public:
```

```
void nhapsl();
```

```
friend void in(const VT &x);
```

```
friend VT lich(const MT &a,const VT &x) ;
```

```
};
```

```
class MT
```

```
{
```

```
private:
```

```
int n;
```

```
double a[20][20];
```

```
public:
```

```
friend VT lich(const MT &a,const VT &x);
```

```
friend void in(const MT &a);
```

```
void nhapsl();
```

```
};
```

```
void VT::nhapsl()
```

```
{
```

```
cout << "\n Cap vec to = ";
```

```
cin >> n ;
```

```
for (int i =1; i<=n ; ++i)
```

```
{
```

```
cout << "\nPhan tu thu " << i << " = " ;
```

```
cin >> x[i];
```

```
}
```

```
}
```

```
void MT::nhapsl()
```

```
{
```

```
cout << "\n Cap ma tran = ";
```

```
cin >> n ;
```

```
for (int i=1; i<=n ; ++i)
```

```
for (int j=1; j<=n; ++j)
```

```
{
```

```
cout << "\nPhan tu thu hang " << i <<
```

```

        " cot " << j << "=" ;
        cin >> a[i][j];
    }
}
VT tich(const MT &a,const VT &x)
{
    VT y;
    int n=a.n;
    if (n!=x.n)
        return x;
    y.n = n;
    for (int i=1; i<=n; ++i)
    {
        y.x[i]=0;
        for (int j=1; j<=n; ++j)
            y.x[i] += a.a[i][j]*x.x[j];
    }
    return y;
}
void in(const VT &x)
{
    cout << "\n";
    for (int i=1; i<=x.n; ++i)
        cout << x.x[i] << " ";
}
void in(const MT &a)
{
    for (int i=1; i<=a.n; ++i)
    {
        cout << "\n" ;
        for (int j=1; j<=a.n; ++j)
            cout << a.a[i][j] << " ";
    }
}
void main()
{
    MT a; VT x,y;
    clrscr();
    a.nhapsl();
    x.nhapsl();
    y=tich(a,x);
    clrscr();
    cout << "\nMa tran A:";
    in(a);
    cout << "\n\nVec to x: " ;
    in(x);
    cout << "\n\nVec y = Ax: " ;
    in(y);
    getch();
}

```

§7. PHẠM VI TRUY XUẤT

7.1. Các từ khoá **private** và **public**

Các thành phần (thuộc tính và phương thức) của lớp có thể khai báo là **private** hoặc **public** theo mẫu:

```
private:
    // Khai báo các thành phần riêng của lớp
public:
    // Khai báo các thành phần chung (công cộng)
```

Chú ý: Các thành phần khai báo mặc định (không dùng các từ khoá **private** và **public**) được xem là các thành phần **private**.

7.2. Các thành phần riêng của lớp chỉ được sử dụng trong phạm vi của lớp (trong thân các phương thức của lớp). Chúng không thể đem ra sử dụng bên ngoài lớp.

+ Một thuộc tính **private**: Thuộc tính này (của một đối tượng nào đó) chỉ có thể được sử dụng trong thân của các phương thức cùng lớp.

+ Một phương thức **private**: Chỉ được sử dụng trong thân của các phương thức cùng lớp.

Ví dụ sau minh họa cách dùng phương thức **private**. Xét lớp **PS** (phân số) với 2 thuộc tính nguyên là **t** (tử) và **m** (mẫu). Giả sử cần xây dựng các phương thức để thực hiện các phép toán cộng trừ, nhân, chia phân số. Do các phép toán này cần dùng trong toàn bộ chương trình, nên các phương thức thực hiện các phép toán cần khai báo là **public**. Để thực hiện các phép tính trên phân số cần dùng đến phép rút gọn phân số. Ta có thể dùng một phương thức **private** để làm điều này vì việc rút gọn chỉ dùng trong nội bộ lớp.

7.3. Các thành phần công cộng của lớp có phạm vi sử dụng trong toàn chương trình. Như vậy nếu một thuộc tính được khai báo là **public**, thì nó có thể được truy nhập trong thân của bất kỳ hàm nào trong chương trình.

Ví dụ trong §6 đã chỉ ra phương án dùng một hàm (tự do) để thực hiện phép cộng 2 số phức như sau là sai:

Phương án 3: Dùng hàm tự do

```
class SP
{
    private:
        double a; // Phần thực
        double b; // Phần ảo
    public:
```

```

        ...
    } ;
    SP cong(SP u1, SP u2)
    {
        SP u;
        u.a = u1.a + u2.a ;
        u.b = u1.b + u2.b ;
        return u;
    }

```

Tuy nhiên nếu sửa chữa bằng cách khai báo các thuộc tính a và b là public thì lại được.

Nhận xét: Các thuộc tính thường khai báo là private để đảm bảo tính đầu kín, an toàn dữ liệu của lớp.

§8. CÁC PHƯƠNG THỨC TOÁN TỬ

8.1. Cách đặt tên

Các phương thức toán tử được xây dựng như các phương thức thông thường, chỉ có khác cách đặt tên. Tên các phương thức toán tử (cũng giống như hàm toán tử) được tạo bằng cách ghép từ khoá operator với một phép toán, ví dụ:

```

operator+
operator<<
operator>>

```

8.2. Con trỏ this

Cũng giống như phương thức thông thường, phương thức toán tử có đối đầu tiên (đối không tường minh) là con trỏ this.

8.3. Toán tử một toán hạng

Các phương thức toán tử một toán hạng: Dùng ngay con trỏ this để biểu thị toán hạng duy nhất này, nên trong phương thức sẽ không có đối tượng minh. Ví dụ phương thức toán tử - (đổi dấu) một đối tượng kiểu SP (số phức) có thể viết như sau:

```

class SP
{
    private:
        double a; // Phần thực
        double b; // Phần ảo
    public:
        SP operator-();
} ;
SP SP:: operator-()

```

```

{
    SP u ;
    u.a = - this->a ;
    u.b = - this->b ;
    return u;
}

```

Cách dùng:

```

SP    u, v;
u    =    -v;

```

8.4. Toán tử hai toán hạng

Các phương thức toán tử hai toán hạng: Con trỏ this ứng với toán hạng thứ nhất, nên trong phương thức chỉ cần dùng một đối tượng mình để biểu thị toán hạng thứ hai. Ví dụ phương thức toán tử + (cộng) hai đối tượng kiểu SP (số phức) có thể viết như sau:

```

class SP
{
    private:
        double a; // Phần thực
        double b; // Phần ảo
    public:
        SP operator+(SP u2);
} ;
SP SP:: operator+(SP u2)
{
    SP u ;
    u.a = this->a + u2.a ;
    u.b = this->b + u2.b ;
    return u;
}

```

Cách dùng:

```

SP    p, p, r;
r    =    p + q ;

```

8.5. Lớp DT (Đa thức)

Chương trình sau sẽ định nghĩa lớp DT và đưa vào các phương thức, hàm:

+ Các thuộc tính:

int n ; // bậc đa thức

double *a ; // trỏ tới vùng nhớ chứa các hệ số đa thức

+ Các phương thức operator+, operator- dùng để đổi dấu các hệ số đa thức

operator+ dùng để cộng 2 đa thức

operator- dùng để trừ 2 đa thức

operator* dùng để nhân 2 đa thức
operator^ dùng để tính giá trị đa thức
operator[] dùng để cho biết bậc và hệ số của đa thức

+ Các hàm bạn:

operator<< dùng để in các hệ số đa thức
operator>> dùng để nhập các hệ số đa thức

+ Hàm (tự do)

double F(DT p, double x) dùng để tính $p(x)$ -giá trị đa thức tại x

+ Nói thêm về phương thức chỉ số và hàm tự do F

- Nếu p là đối tượng của lớp DT, thì hàm chỉ số cho biết:

$p[-1] = \text{double}(n)$

$p[i] = a[i], i=0, 1, \dots, n$

- Hàm tự do F sẽ dùng phương thức chỉ số để xác định n, các hệ số đa thức và dùng chúng để tính giá trị đa thức.

+ Trong chương trình sử dụng hàm new để cấp phát vùng nhớ chứa hệ số đa thức.

+ Nội dung chương trình gồm:

- Nhập, in các đa thức p, q, r, s
- Tính đa thức: $f = -(p + q) * (r - s)$
- Nhập các số thực x1 và x2
- Tính $f(x1)$ (bằng cách dùng phương thức operator^)
- Tính $f(x2)$ (bằng cách dùng hàm F)

// Chương trình CT3_10.CPP

```
#include <conio.h>
```

```
#include <iostream.h>
```

```
#include <math.h>
```

```
class DT
```

```
{
```

```
private:
```

```
int n;    // Bậc đa thức
```

```
double *a; // Trỏ tới vùng nhớ chứa các hệ số đa thức  
          // a0, a1, ...
```

```
public:
```

```
friend ostream& operator<< (ostream& os, const DT &d);
```

```
friend istream& operator>> (istream& is, DT &d);
```

```
DT operator-();
```

```

    DT operator+(const DT &d2);
    DT operator-(DT d2);
    DT operator*(const DT &d2);
    double operator^(const double &x); // Tinh gia tri
                                         // da thuc

    double operator[](int i)
    {
        if(i<0)
            return double(n);
        else
            return a[i];
    }
};

// Ham tinh gia tri da thuc
double F(DT d,double x)
{
    double s=0.0, t=1.0;
    int n;
    n = int(d[-1]);
    for (int i=0; i<=n; ++i)
    {
        s += d[i]*t;
        t *= x;
    }
    return s;
}

ostream& operator<< (ostream& os,const DT &d)
{
    os << " - Cac he so (tu ao): " ;
    for (int i=0 ; i<= d.n ; ++i)
        os << d.a[i] <<" ";
    return os;
}

istream& operator>> (istream& is,DT &d)
{
    cout << " - Bac da thuc: " ;
    cin >> d.n;
    d.a = new double[d.n+1];
    cout << "Nhap cac he so da thuc:\n" ;
    for (int i=0 ; i<= d.n ; ++i)
    {
        cout << "He so bac " << i << " = " ;
        is >> d.a[i] ;
    }
    return is;
}

```

```

DT DT::operator-()
{
    DT p;
    p.n = n;
    p.a = new double[n+1];
    for (int i=0 ; i<=n ; ++i)
        p.a[i] = -a[i];
    return p;
}

DT DT::operator+(const DT &d2)
{
    DT d;
    int k,i;
    k = n > d2.n ? n : d2.n ;
    d.a = new double[k+1];
    for (i=0; i<=k ; ++i)
        if (i<=n && i<=d2.n)
            d.a[i] = a[i] + d2.a[i];
        else if (i<=n)
            d.a[i] = a[i];
        else
            d.a[i] = d2.a[i];
    i=k;
    while(i>0 && d.a[i]==0.0) --i;
    d.n = i;
    return d ;
}

DT DT::operator-(DT d2)
{
    return (*this + (-d2));
}

DT DT::operator*(const DT &d2)
{
    DT d;
    int k, i, j;
    k = d.n = n + d2.n ;
    d.a = new double[k+1];
    for (i=0; i<=k; ++i) d.a[i] = 0;
    for (i=0 ; i<= n ; ++i)
        for (j=0 ; j<= d2.n ; ++j)
            d.a[i+j] += a[i]*d2.a[j] ;
    return d;
}

double DT::operator^(const double &x)
{
    double s=0.0, t=1.0;

```

```

    for (int i=0 ; i<= n ; ++i)
    {
        s += a[i]*t;
        t *= x;
    }
    return s;
}

void main()
{
    DT p,q,r,s,f;
    double x1,x2,g1,g2;
    clrscr();
    cout <<"\nNhap da thuc P " ; cin >> p;
    cout << "\nDa thuc p " << p ;
    cout <<"\nNhap da thuc Q " ; cin >> q;
    cout << "\nDa thuc q " << q ;
    cout <<"\nNhap da thuc R " ; cin >> r;
    cout << "\nDa thuc r " << r ;
    cout <<"\nNhap da thuc S " ; cin >> s;
    cout << "\nDa thuc s " << s ;
    f = -(p+q)*(r-s);
    cout << "\nNhap so thuc x1: " ; cin >> x1;
    cout << "\nNhap so thuc x2: " ; cin >> x2;
    g1 = f^x1;
    g2 = f(f,x2);
    cout << "\nDa thuc f " << f ;
    cout << "\n f("<<x1<<") = " << g1;
    cout << "\n f("<<x2<<") = " << g2;
    getch();
}

```

Chương 4

HÀM TẠO, HÀM HUỖ VÀ CÁC VẤN ĐỀ LIÊN QUAN

Chương này trình bày một số vấn đề có tính chuyên sâu hơn về lớp như:

- Hàm tạo (constructor)
- Hàm huỷ (destructor)
- Toán tử gán và hàm tạo sao chép
- Mối liên quan giữa hàm tạo và đối tượng thành phần
- Các thành phần tĩnh
- Lớp bạn, hàm bạn
- Đối tượng hằng
- Phương thức inline

§1. HÀM TẠO (CONSTRUCTOR)

1.1. Công dụng

Hàm tạo cũng là một phương thức của lớp (nhưng khá đặc biệt) dùng để tạo dựng một đối tượng mới. Chương trình dịch sẽ cấp phát bộ nhớ cho đối tượng sau đó sẽ gọi đến hàm tạo. Hàm tạo sẽ khởi gán giá trị cho các thuộc tính của đối tượng và có thể thực hiện một số công việc khác nhằm chuẩn bị cho đối tượng mới.

1.2. Cách viết hàm tạo

1.2.1. Điểm khác của hàm tạo và các phương thức thông thường

Khi viết hàm tạo cần để ý 3 sự khác biệt của hàm tạo so với các phương thức khác như sau:

- + Tên của hàm tạo: Tên của hàm tạo bắt buộc phải trùng với tên của lớp.
- + Không khai báo kiểu cho hàm tạo.
- + Hàm tạo không có kết quả trả về.

1.2.2. Sự giống nhau của hàm tạo và các phương thức thông thường

Ngoài 3 điểm khác biệt trên, hàm tạo được viết như các phương thức khác:

- + Hàm tạo có thể được xây dựng bên trong hoặc bên ngoài định nghĩa lớp.

- + Hàm tạo có thể có đối hoặc không có đối.
- + Trong một lớp có thể có nhiều hàm tạo (cùng tên nhưng khác bộ đối).

Ví dụ sau định nghĩa lớp DIEM_DH (Điểm đồ họa) có 3 thuộc tính:

```
int x; // hoành độ (cột) của điểm
int y; // tung độ (hàng) của điểm
int m; // màu của điểm
```

và đưa vào 2 hàm tạo để khởi gán cho các thuộc tính của lớp:

```
// Hàm tạo không đối: Dùng các giá trị cố định để khởi gán cho
// x, y, m
DIEM_DH() ;
// Hàm tạo có đối: Dùng các đối xl, yl, ml để khởi gán cho
x, y, m
// Đối ml có giá trị mặc định 15 (màu trắng)
DIEM_DH(int xl, int yl, int ml=15) ;
class DIEM_DH
{
private:
    int x, y, m ;
public:
    //Hàm tạo không đối: khởi gán cho x=0, y=0, m=1
    // Hàm này viết bên trong định nghĩa lớp
    DIEM_DH()
    {
        x=y=0;
        m=1;
    }
    // Hàm tạo này xây dựng bên ngoài định nghĩa lớp
    DIEM_DH(int xl, int yl, int ml=15) ;
    // Các phương thức khác
} ;
// Xây dựng hàm tạo bên ngoài định nghĩa lớp
DIEM_DH::DIEM_DH(int xl, int yl, int ml)
{
    x=xl; y=yl; m=ml;
}
```

1.3. Dùng hàm tạo trong khai báo

+ Khi đã xây dựng các hàm tạo, ta có thể dùng chúng trong khai báo để tạo ra một đối tượng đồng thời khởi gán cho các thuộc tính của đối tượng được tạo. Dựa vào các tham số trong khai báo mà Trình biên dịch sẽ biết cần gọi đến hàm tạo nào.

+ Khi khai báo một biến đối tượng có thể sử dụng các tham số để khởi gán cho các thuộc tính của biến đối tượng.

+ Khi khai báo mảng đối tượng không cho phép dùng các tham số để khởi gán.

+ Câu lệnh khai báo một biến đối tượng sẽ gọi tới hàm tạo 1 lần

+ Câu lệnh khai báo một mảng n đối tượng sẽ gọi tới hàm tạo n lần.

Ví dụ:

```
DIEM_DH d; // Gọi tới hàm tạo không đối.  
                // Kết quả d.x=0, d.y=0, d.m=1  
DIEM_DH u(200,100,4); // Gọi tới hàm tạo có đối.  
                // Kết quả u.x=200, u.y=100, d.m=4  
DIEM_DH v(300,250); // Gọi tới hàm tạo có đối.  
                // Kết quả v.x=300, v.y=250, d.m=15  
DIEM_DH p[10] ; // Gọi tới hàm tạo không đối 10 lần
```

Chú ý: Với các hàm có đối kiểu lớp, thì đối chỉ xem là các tham số hình thức, vì vậy khai báo đối (trong dòng đầu của hàm) sẽ không tạo ra đối tượng mới và do đó không gọi tới các hàm tạo.

1.4. Dùng hàm tạo trong cấp phát bộ nhớ

+ Khi cấp phát bộ nhớ cho một đối tượng có thể dùng các tham số để khởi gán cho các thuộc tính của đối tượng, ví dụ:

```
DIEM_DH *q = new DIEM_DH(50,40,6); //Gọi tới hàm tạo có đối  
                // Kết quả q->x=50, q->y=40, q->m=6  
DIEM_DH *r = new DIEM_DH ; // Gọi tới hàm tạo không đối  
                // Kết quả r->x=0, r->y= 0, r->m=1
```

+ Khi cấp phát bộ nhớ cho một dãy đối tượng không cho phép dùng tham số để khởi gán, ví dụ:

```
int n=20;  
DIEM_DH *s = new DIEM_DH[n]; // Gọi tới hàm tạo không  
                // đối 20 lần.
```

1.5. Dùng hàm tạo để biểu diễn các đối tượng hằng

+ Như đã biết, sau khi định nghĩa lớp DIEM_DH thì có thể xem lớp này như một kiểu dữ liệu như int, double, char, ...

Với kiểu int chúng ta có các hằng int, như 356.

Với kiểu double chúng ta có các hằng double, như 98.75

Khái niệm hằng kiểu int, hằng kiểu double có thể mở rộng cho hằng kiểu DIEM_DH

+ Để biểu diễn một hằng đối tượng (hay còn gọi: Đối tượng hằng) chúng ta phải dùng tới hàm tạo. Mẫu viết như sau:

Tên lớp(danh sách tham số);

Ví dụ đối với lớp DIEM_DH nói trên, có thể viết như sau:

```
DIEM_DH(345,123,8) // Biểu thị một đối tượng kiểu DIEM_DH
// có các thuộc tính x=345, y=123, m=8
```

Chú ý: Có thể sử dụng một hằng đối tượng như một đối tượng. Nói cách khác, có thể dùng hằng đối tượng để thực hiện một phương thức, ví dụ nếu viết:

```
DIEM_DH(345,123,8).in();
```

thì có nghĩa là thực hiện phương thức in() đối với hằng đối tượng.

1.6. Ví dụ minh họa

Chương trình sau đây minh họa cách xây dựng hàm tạo và cách sử dụng hàm tạo trong khai báo, trong cấp phát bộ nhớ và trong việc biểu diễn các hằng đối tượng.

```
//CT4 02.CPP
#include <conio.h>
#include <iostream.h>
#include <iomanip.h>
class DIEM_DH
{
private:
    int x,y,m;
public:
    // Hàm bạn dùng để in đối tượng DIEM_DH
    friend void in(DIEM_DH d)
    {
        cout <<"\n " << d.x << " " << d.y<<" " << d.m ;
    }
    // Phương thức dùng để in đối tượng DIEM_DH
    void in()
    {
        cout <<"\n " << x << " " << y<<" " << m ;
    }
    //Hàm tạo không đối
    DIEM_DH()
    {
        x=y=0;
        m=1;
    }
    // Hàm tạo có đối, đối m1 có giá trị mặc định
    // là 15 (màu trắng)
    DIEM_DH(int x1,int y1,int m1=15);
};
// Xây dựng hàm tạo
DIEM_DH::DIEM_DH(int x1,int y1,int m1)
{
    x=x1; y=y1; m=m1;
}
void main()
```

```

{
    DIEM DH d1; // Gọi tới hàm tạo không đổi
    DIEM_DH d2(200,200,10); // Gọi tới hàm tạo có đổi
    DIEM_DH *d;
    d= new DIEM_DH(300,300); // Gọi tới hàm tạo có đổi
    clrscr();
    in(d1); //Gọi hàm bạn in()
    d2.in();//Gọi phương thức in()
    in(*d); //Gọi hàm bạn in()
    DIEM_DH(2,2,2).in();//Gọi phương thức in()
    DIEM_DH t[3]; // 3 lần gọi hàm tạo không đổi
    DIEM_DH *q; // Gọi hàm tạo không đổi
    int n;
    cout << "\nN= ";
    cin >> n;
    q=new DIEM_DH[n+1]; // (n+1) lần gọi hàm tạo không đổi
    for (int i=0;i<=n;++i)
        q[i]=DIEM_DH(300+i,200+i,8);// (n+1) lần gọi hàm tạo
                                   // có đổi
    for (i=0;i<=n;++i)
        q[i].in(); // Gọi phương thức in()
    for (i=0;i<=n;++i)
        DIEM_DH(300+i,200+i,8).in();// Gọi phương thức in()
    getch();
}

```

§2. LỚP KHÔNG CÓ HÀM TẠO VÀ HÀM TẠO MẶC ĐỊNH

Các chương trình nêu trong chương 3 đều không có hàm tạo. Vậy khi đó các đối tượng được hình thành như thế nào ?

2.1. Nếu lớp không có hàm tạo, Chương trình dịch sẽ cung cấp một hàm tạo mặc định không đổi (default). Hàm này thực chất không làm gì cả. Như vậy một đối tượng tạo ra chỉ được cấp phát bộ nhớ, còn các thuộc tính của nó chưa được xác định. Chúng ta có thể kiểm chứng điều này, bằng cách chạy chương trình sau:

```

//CT4_03.CPP - Hàm tạo mặc định
#include <conio.h>
#include <iostream.h>
class DIEM DH
{
private:
    int x,y,m;
public:
    // Phương thức
    void in()
    {
        cout << "\n " << x << " " << y << " " << m ;
    }
}

```

```

};
void main()
{
    DIEM DH d;
    d.in();
    DIEM DH *p;
    p= new DIEM DH[10];
    clrscr();
    d.in();
    for (int i=0;i<10;++i)
        (p+i)->in();
    getch();
}

```

2.2. Nếu trong lớp đã có ít nhất một hàm tạo, thì hàm tạo mặc định sẽ không được phát sinh nữa. Khi đó mọi câu lệnh xây dựng đối tượng mới đều sẽ gọi đến một hàm tạo của lớp. Nếu không tìm thấy hàm tạo cần gọi thì Chương trình dịch sẽ báo lỗi. Điều này thường xảy ra khi chúng ta không xây dựng hàm tạo không đối, nhưng lại sử dụng các khai báo không tham số như ví dụ sau:

```

#include <conio.h>
#include <iostream.h>
class DIEM DH
{
    private:
        int x,y,m;
    public: // Phương thức dùng để in đối tượng DIEM_DH
        void in()
        {
            cout <<"\n " << x << " " << y<<" " << m ;
        }
        //Hàm tạo có đối
        DIEM_DH::DIEM_DH(int x1,int y1,int m1)
        {
            x=x1; y=y1; m=m1;
        }
};
void main()
{
    DIEM_DH d1(200,200,10); // Gọi tới hàm tạo có đối
    DIEM_DH d2; // Gọi tới hàm tạo không đối
    d2= DIEM_DH(300,300,8); // Gọi tới hàm tạo có đối
    d1.in();
    d2.in();
    getch();
}

```

Trong các câu lệnh trên, chỉ có câu lệnh thứ 2 trong hàm main() là bị báo lỗi. Câu lệnh này sẽ gọi tới hàm tạo không đối, mà hàm này chưa được xây dựng.

Giải pháp: Có thể chọn một trong 2 giải pháp sau:

- Xây dựng thêm hàm tạo không đối.
- Gán giá trị mặc định cho tất cả các đối x1, y1 và m1 của hàm tạo đã xây dựng ở trên.

Theo phương án 2, chương trình có thể sửa như sau:

```
#include <conio.h>
#include <iostream.h>
class DIEM_DH
{
private:
    int x,y,m;
public:
        // Phương thức dùng để in đối tượng DIEM_DH
    void in()
    {
        cout <<"\n " << x << " " << y << " " << m ;
    }
    //hàm tạo có đối, tất cả các đối đều có giá trị mặc định
    DIEM_DH::DIEM_DH(int x1=0,int y1=0,int m1=15)
    {
        x=x1; y=y1; m=m1;
    }
};
void main()
{
    DIEM_DH d1(200,200,10); // Gọi tới hàm tạo, không dùng
                           // tham số mặc định
    DIEM_DH d2; // Gọi tới hàm tạo, dùng 3 tham số mặc định
    d2= DIEM_DH(300,300); // Gọi tới hàm tạo, dùng 1
                           // tham số mặc định

    d1.in();
    d2.in();
    getch();
}
```

§3. LỚP ĐA THỨC

Chương trình dưới đây là sự cải tiến chương trình trong mục 8.5 của chương 3 bằng cách đưa vào 2 hàm tạo:

```
//Hàm tạo không đối
DT()
{
    this->n=0; this->a=NULL;
}
//Hàm tạo có đối
DT(int n1)
{
    this->n=n1 ;
    this->a = new double[n1+1];
}
```

Hàm tạo có đối sẽ tạo một đối tượng mới (kiểu DT) gồm 2 thuộc tính là biến nguyên n và con trỏ a. Ngoài ra còn cấp phát bộ vùng nhớ (cho a) để chứa các hệ số của đa thức.

Nếu không xây dựng hàm tạo, mà sử dụng hàm tạo mặc định thì các đối tượng (kiểu DT) tạo ra bởi các lệnh khai báo sẽ chưa có bộ nhớ để chứa đa thức. Như vậy đối tượng tạo ra chưa hoàn chỉnh và chưa dùng được. Để có một đối tượng hoàn chỉnh phải qua 2 bước:

+ Dùng khai báo để tạo các đối tượng, ví dụ:

```
DT d;
```

+ Cấp phát vùng nhớ (cho đối tượng) để chứa đa thức, ví dụ:

```
d.n = m;
```

```
d.a = new double[m+1] ;
```

Quy trình này được áp dụng trong các phương thức toán tử của chương trình trong mục 8.5 chương 3. Rõ ràng quy trình này vừa dài vừa không tiện lợi, lại hay mắc lỗi, vì người lập trình hay quên không cấp phát bộ nhớ.

Việc dùng các hàm tạo để sản sinh ra các đối tượng hoàn chỉnh tỏ ra tiện lợi hơn, vì tránh được các thao tác phụ (như cấp phát bộ nhớ) nằm bên ngoài khai báo. Phương án dùng hàm tạo sẽ được sử dụng trong các phương thức toán tử của chương trình dưới đây:

+ Nội dung chương trình gồm:

- Nhập, in các đa thức p, q, r, s

- Tính đa thức: $f = -(p + q) * (r - s)$

- Nhập các số thực x1 và x2

- Tính $f(x1)$ (bằng cách dùng phương thức operator^)

- Tính $f(x2)$ (bằng cách dùng hàm F)

```
// CT4_05.CPP
```

```
#include <conio.h>
```

```

#include <iostream.h>
#include <math.h>
class DT
{
private:
    int n; // Bac da thuc
    double *a; // Tro toi vung nho chua cac he so
                // da thuc a0, a1,...
public:
    DT()
    {
        this->n=0; this->a=NULL;
    }
    DT(int n1)
    {
        this->n=n1 ;
        this->a = new double[n1+1];
    }
    friend ostream& operator<< (ostream& os,const DT &d);
    friend istream& operator>> (istream& is,DT &d);
    DT operator-();
    DT operator+(const DT &d2);
    DT operator-(DT d2);
    DT operator*(const DT &d2);
    double operator^(const double &x); // Tinh gia tri
                                        // da thuc

    double operator[](int i)
    {
        if (i<0)
            return double(n);
        else
            return a[i];
    }
};

// Ham tinh gia tri da thuc
double F(DT d,double x)
{
    double s=0.0, t=1.0;
    int n;
    n = int(d[-1]);
    for (int i=0; i<=n; ++i)
    {
        s += d[i]*t;
        t *= x;
    }
    return s;
}

ostream& operator<< (ostream& os,const DT &d)
{
    os << " - Cac he so (tu ao): " ;
    for (int i=0 ; i<= d.n ; ++i)

```

```

        os << d.a[i] << " " ;
        return os;
    }
istream& operator>> (istream& is, DT &d)
{
    if (d.a!=NULL) delete d.a;
    cout << " - Bac da thuc: " ;
    cin >> d.n;
    d.a = new double[d.n+1];
    cout << "Nhap cac he so da thuc:\n" ;
    for (int i=0 ; i<= d.n ; ++i)
    {
        cout << "He so bac " << i << " = " ;
        is >> d.a[i] ;
    }
    return is;
}
DT DT::operator-()
{
    DT p(this->n);
    for (int i=0 ; i<=n ; ++i)
        p.a[i] = -a[i];
    return p;
}
DT DT::operator+(const DT &d2)
{
    int k,i;
    k = n > d2.n ? n : d2.n ;
    DT d(k);
    for (i=0; i<=k ; ++i)
        if (i<=n && i<=d2.n)
            d.a[i] = a[i] + d2.a[i];
        else if (i<=n)
            d.a[i] = a[i];
        else
            d.a[i] = d2.a[i];
    i=k;
    while(i>0 && d.a[i]==0.0) --i;
    d.n = i;
    return d ;
}
DT DT::operator-(DT d2)
{
    return (*this + (-d2));
}
DT DT::operator*(const DT &d2)
{
    int k, i, j;
    k = n + d2.n ;
    DT d(k);
    for (i=0; i<=k; ++i) d.a[i] = 0;

```

```

        for (i=0 ; i<= n ; ++i)
            for (j=0 ; j<= d2.n ; ++j)
                d.a[i+j] += a[i]*d2.a[j] ;
        return d;
    }
double DT::operator^(const double &x)
{
    double s=0.0, t=1.0;
    for (int i=0 ; i<= n ; ++i)
    {
        s += a[i]*t ; t *= x;
    }
    return s;
}
void main()
{
    DT p,q,r,s,f;
    double x1,x2,g1,g2;
    clrscr();
    cout <<"\nNhap da thuc P " ; cin >> p;
    cout << "\nDa thuc p " << p ;
    cout <<"\nNhap da thuc Q " ; cin >> q;
    cout << "\nDa thuc q " << q ;
    cout <<"\nNhap da thuc R " ; cin >> r;
    cout << "\nDa thuc r " << r ;
    cout <<"\nNhap da thuc S " ; cin >> s;
    cout << "\nDa thuc s " << s ;
    f = -(p+q)*(r-s);
    cout << "\nNhap so thuc x1: " ; cin >> x1;
    cout << "\nNhap so thuc x2: " ; cin >> x2;
    g1 = f^x1;
    g2 = F(f,x2);
    cout << "\nDa thuc f " << f ;
    cout << "\n f("<<x1<<" ) = " << g1;
    cout << "\n f("<<x2<<" ) = " << g2;
    getch();
}

```

§4. HÀM TẠO SAO CHÉP (COPY CONSTRUCTOR)

4.1. Hàm tạo sao chép mặc định

Giả sử đã định nghĩa một lớp nào đó, ví dụ lớp PS (phân số). Khi đó:

+ Ta có thể dùng câu lệnh khai báo hoặc cấp phát bộ nhớ để tạo các đối tượng mới, ví dụ:

```
PS p1, p2 ;
```

```
PS *p = new PS ;
```

+ Ta cũng có thể dùng lệnh khai báo để tạo một đối tượng mới từ một đối tượng đã tồn tại, ví dụ:

PS u;

PS v(u) ; // Tạo v theo u

Ý nghĩa của câu lệnh này như sau:

- Nếu trong lớp PS chưa xây dựng hàm tạo sao chép, thì câu lệnh này sẽ gọi tới một hàm tạo sao chép mặc định (của C⁺⁺). Hàm này sẽ sao chép nội dung từng bit của u vào các bit tương ứng của v. Như vậy các vùng nhớ của u và v sẽ có nội dung như nhau. Rõ ràng trong đa số các trường hợp, nếu lớp không có các thuộc tính kiểu con trỏ hay tham chiếu, thì việc dùng các hàm tạo sao chép mặc định (để tạo ra một đối tượng mới có nội dung như một đối tượng cho trước) là đủ và không cần xây dựng một hàm tạo sao chép mới.

- Nếu trong lớp PS đã có hàm tạo sao chép (cách viết sẽ nói sau) thì câu lệnh:

PS v(u) ;

sẽ tạo ra đối tượng mới v, sau đó gọi tới hàm tạo sao chép để khởi gán v theo u.

Ví dụ sau minh họa cách dùng hàm tạo sao chép mặc định:

Trong chương trình đưa vào lớp PS (phân số):

+ Các thuộc tính gồm: t (tử số) và m (mẫu).

+ Trong lớp không có phương thức nào cả mà chỉ có 2 hàm bạn là các hàm toán tử nhập (>>) và xuất (<<).

+ Nội dung chương trình là: Dùng lệnh khai báo để tạo một đối tượng u (kiểu PS) có nội dung như đối tượng đã có d.

```
//CT4_06.CPP
```

```
// Ham tao sao chep mac dinh
```

```
#include <conio.h>
```

```
#include <iostream.h>
```

```
class PS
```

```
{
```

```
private:
```

```
int t,m ;
```

```
public:
```

```
friend ostream& operator<< (ostream& os,const PS &p)
```

```
{
```

```
os << " = " << p.t << "/" << p.m;
```

```
return os;
```

```
}
```

```
friend istream& operator>> (istream& is, PS &p)
```

```

        {
            cout << " - Nhập tu và mau:  " ;
            is >> p.t >> p.m ;
            return is;
        }
    };

void main()
{
    PS d;
    cout << "\n Nhập PS d"; cin >> d;
    cout << "\n PS d " << d;
    PS u(d);
    cout << "\n PS u " << u;
    getch();
}

```

4.2. Cách xây dựng hàm tạo sao chép

+ Hàm tạo sao chép sử dụng một đối **kiểu tham chiếu đối tượng** để khởi gán cho đối tượng mới. Hàm tạo sao chép được viết theo mẫu:

```

Tên_lớp (const 'Tên_lớp & dt)
{
    // Các câu lệnh dùng các thuộc tính của đối tượng dt
    // để khởi gán cho các thuộc tính của đối tượng mới
}

```

+ **Ví dụ** có thể xây dựng hàm tạo sao chép cho lớp PS như sau:

```

class PS
{
    private:
        int t,m ;
    public:
        PS (const PS &p)
        {
            this->t = p.t ;
            this->m = p.m ;
        }
        ...
} ;

```

4.3. Khi nào cần xây dựng hàm tạo sao chép

+ **Nhận xét:** Hàm tạo sao chép trong ví dụ trên không khác gì hàm tạo sao chép mặc định.

+ Khi lớp không có các thuộc tính kiểu con trỏ hoặc tham chiếu, thì dùng hàm tạo sao chép mặc định là đủ.

+ Khi lớp có các thuộc tính con trỏ hoặc tham chiếu, thì hàm tạo sao chép mặc định chưa đáp ứng được yêu cầu. Ví dụ lớp DT (đa thức) trong §3:

```

class DT
{
private:
    int n;    // Bậc đa thức
    double *a; // Trỏ tới vùng nhớ chứa các hệ số đa thức
               // a0, a1, ...
public:
    DT()
    {
        this->n=0; this->a=NULL;
    }
    DT(int n1)
    {
        this->n=n1 ;
        this->a = new double[n1+1];
    }
    friend ostream& operator<< (ostream& os,const DT &d);
    friend istream& operator>> (istream& is,DT &d);
    ....
} ;

```

Bây giờ chúng ta hãy theo dõi xem việc dùng hàm tạo mặc định trong đoạn chương trình sau sẽ dẫn đến sai lầm như thế nào:

```

DT d ;
// Tạo đối tượng d kiểu DT
cin >> d ;
/* Nhập đối tượng d, gồm: nhập một số nguyên dương và
   gán cho d.n, cấp phát vùng nhớ cho d.a, nhập các hệ
   số của đa thức và chứa vào vùng nhớ được cấp phát
*/
DT u(d) ;
/* Dùng hàm tạo mặc định để xây dựng đối tượng u theo d
   Kết quả: u.n = d.n và u.a = d.a. Như vậy 2 con trỏ
   u.a và d.a cùng trỏ đến một vùng nhớ.
*/

```

Nhận xét: Mục đích của ta là tạo ra một đối tượng u giống như d, nhưng độc lập với d. Nghĩa là khi d thay đổi thì u không bị ảnh hưởng gì. Thế nhưng mục tiêu này không đạt được, vì u và d có chung một vùng nhớ chứa hệ số của đa thức, nên khi sửa đổi các hệ số của đa thức trong d thì các hệ số của đa thức trong u cũng thay đổi theo. Còn một trường hợp nữa cũng dẫn đến lỗi là khi một trong 2 đối tượng u và d bị giải phóng (thu hồi vùng nhớ chứa đa thức) thì đối tượng còn lại cũng sẽ không còn vùng nhớ nữa.

Ví dụ sau sẽ minh họa nhận xét trên: Khi d thay đổi thì u cũng thay đổi và ngược lại khi u thay đổi thì d cũng thay đổi theo.

```
//CT4_07.CPP
```

```

#include <conio.h>
#include <iostream.h>
#include <math.h>
class DT
{
private:
    int n; // Bac da thuc
    double *a; // Tro toi vung nho chua cac he so
                // da thuc a0, a1,...
public:
    DT()
    {
        this->n=0;
        this->a=NULL;
    }
    DT(int n1)
    {
        this->n=n1 ;
        this->a = new double[n1+1];
    }
    friend ostream& operator<< (ostream& os,const DT &d);
    friend istream& operator>> (istream& is,DT &d);
} ;
ostream& operator<< (ostream& os,const DT &d)
{
    os << " - Cac he so (tu ao): " ;
    for (int i=0 ; i<= d.n ; ++i)
        os << d.a[i] <<" " ;
    return os;
}
istream& operator>> (istream& is,DT &d)
{
    if (d.a!=NULL) delete d.a;
    cout << " - Bac da thuc: " ;
    cin >> d.n;
    d.a = new double[d.n+1];
    cout << "Nhap cac he so da thuc:\n" ;
    for (int i=0 ; i<= d.n ; ++i)
    {
        cout << "He so bac " << i << " = " ;
        is >> d.a[i] ;
    }
    return is;
}
void main()
{
    DT d;
    clrscr();
    cout <<"\nNhap da thuc d " ; cin >> d;
    DT u(d);
    cout << "\nda thuc d " << d ;

```

```

cout << "\nĐã thực u " << u ;
cout << "\nNhập đa thức d " ; cin >> d;
cout << "\nĐã thực d " << d ;
cout << "\nĐã thực u " << u ;
cout << "\nNhập đa thức u " ; cin >> u;
cout << "\nĐã thực d " << d ;
cout << "\nĐã thực u " << u ;
    getch();
}

```

4.4. Ví dụ về hàm tạo sao chép

Trong chương trình trên đã chỉ rõ: Hàm tạo sao chép mặc định là chưa thoả mãn đối với lớp DT. Vì vậy cần viết hàm tạo sao chép để xây dựng đối tượng mới (ví dụ u) từ một đối tượng đang tồn tại (ví dụ d) theo các yêu cầu sau:

- + Gán d.n cho u.n
- + Cấp phát một vùng nhớ cho u.a để có thể chứa được $(d.n + 1)$ hệ số.
- + Gán các hệ số chứa trong vùng nhớ của d.a sang vùng nhớ của u.a

Như vậy chúng ta sẽ tạo được đối tượng u có nội dung ban đầu giống như d, nhưng độc lập với d.

Để đáp ứng các yêu cầu nêu trên, hàm tạo sao chép cần được xây dựng như sau:

```

DT::DT(const DT &d)
{
    this->n = d.n;
    this->a = new double[d.n+1];
    for (int i=0; i<=d.n; ++i)
        this->a[i] = d.a[i];
}

```

Chương trình sau sẽ minh họa điều này: Sự thay đổi của d không làm ảnh hưởng đến u và ngược lại sự thay đổi của u không làm ảnh hưởng đến d.

```

//CT4_08.CPP
// Viết hàm tạo sao chép cho lớp DT
#include <conio.h>
#include <iostream.h>
#include <math.h>
class DT
{
private:
    int n;          // Bậc đa thức
    double *a;      // Trỏ tới vùng nhớ chứa các hệ số
                   // đa thức a0, a1,...

```

```

public:
    DT()
    {
        this->n=0; this->a=NULL;
    }
    DT(int n1)
    {
        this->n=n1 ;
        this->a = new double[n1+1];
    }
    DT(const DT &d);
    friend ostream& operator<< (ostream& os,const DT &d);
    friend istream& operator>> (istream& is,DT &d);
} ;
DT::DT(const DT &d)
{
    this->n = d.n;
    this->a = new double[d.n+1];
    for (int i=0;i<=d.n;++i)
        this->a[i] = d.a[i];
}
ostream& operator<< (ostream& os,const DT &d)
{
    os << " - Cac he so (tu ao): " ;
    for (int i=0 ; i<= d.n ; ++i)
        os << d.a[i] <<" " ;
    return os;
}
istream& operator>> (istream& is,DT &d)
{
    if (d.a!=NULL) delete d.a;
    cout << " - Bac da thuc: " ;
    cin >> d.n;
    d.a = new double[d.n+1];
    cout << "Nhap cac he so da thuc:\n" ;
    for (int i=0 ; i<= d.n ; ++i)
    {
        cout << "He so bac " << i << " = " ;
        is >> d.a[i] ;
    }
    return is;
}
void main()
{
    DT d;
    clrscr();

```

```

cout << "\nNhap da thuc d " ; cin >> d;
DT u(d);
cout << "\nDa thuc d " << d ;
cout << "\nDa thuc u " << u ;
cout << "\nNhap da thuc d " ; cin >> d;
cout << "\nDa thuc d " << d ;
cout << "\nDa thuc u " << u ;
cout << "\nNhap da thuc u " ; cin >> u;
cout << "\nDa thuc d " << d ;
cout << "\nDa thuc u " << u ;
getch();
}

```

§5. HÀM HUỖ (DESTRUCTOR)

5.1. Công dụng của hàm huỷ

Hàm huỷ là một hàm thành viên của lớp (phương thức) có chức năng ngược với hàm tạo. Hàm huỷ được gọi trước khi giải phóng (xoá bỏ) một đối tượng để thực hiện một số công việc có tính “dọn dẹp” trước khi đối tượng được huỷ bỏ, ví dụ như giải phóng một vùng nhớ mà đối tượng đang quản lý, xoá đối tượng khỏi màn hình nếu như nó đang hiển thị, ...

Việc huỷ bỏ một đối tượng thường xảy ra trong 2 trường hợp sau:

- + Trong các toán tử và các hàm giải phóng bộ nhớ, như delete, free,...
- + Giải phóng các biến, mảng cục bộ khi thoát khỏi hàm, phương thức.

5.2. Hàm huỷ mặc định

Nếu trong lớp không định nghĩa hàm huỷ, thì một hàm huỷ mặc định không làm gì cả được phát sinh. Đối với nhiều lớp thì hàm huỷ mặc định là đủ, và không cần đưa vào một hàm huỷ mới.

5.3. Quy tắc viết hàm huỷ

Mỗi lớp chỉ có một hàm huỷ viết theo các quy tắc sau:

+ Kiểu của hàm: Hàm huỷ cũng giống như hàm tạo là hàm không có kiểu, không có giá trị trả về.

+ Tên hàm: Tên của hàm huỷ gồm một dấu ngã (đứng trước) và tên lớp:
~Tên_lớp

+ Đối: Hàm huỷ không có đối

Ví dụ có thể xây dựng hàm huỷ cho lớp DT (đa thức) ở §3 như sau:

```

class DT
{

```

```

private:
    int n;          // Bac da thuc
    double *a;      // Tro toi vung nho chua cac he so da thuc
                    // a0, a1, ...

public:
    ~DT()
    {
        this->n=0;
        delete this->a;
    }
    ...
} ;

```

5.4. Vai trò của hàm huỷ trong lớp DT

5.4.1. Khiếm khuyết của chương trình trong §3

Chương trình trong §3 định nghĩa lớp DT (đa thức) khá đầy đủ gồm:

- + Các hàm tạo
- + Các hàm toán tử nhập >>, xuất <<
- + Các hàm toán tử thực hiện các phép tính + - *

Tuy nhiên vẫn còn thiếu hàm huỷ để giải phóng vùng nhớ mà đối tượng kiểu DT (cần huỷ) đang quản lý.

Chúng ta hãy phân tích các khiếm khuyết của chương trình này:

+ Khi chương trình gọi tới một phương thức toán tử để thực hiện các phép tính cộng, trừ, nhân đa thức, thì một đối tượng trung gian được tạo ra. Một vùng nhớ được cấp phát và giao cho nó (đối tượng trung gian) quản lý.

+ Khi thực hiện xong phép tính sẽ ra khỏi phương thức. Đối tượng trung gian bị xoá, tuy nhiên chỉ vùng nhớ của các thuộc tính của đối tượng này được giải phóng. Còn vùng nhớ (chứa các hệ số của đa thức) mà đối tượng trung gian đang quản lý thì không hề bị giải phóng. Như vậy số vùng nhớ bị chiếm dụng vô ích sẽ tăng lên.

5.4.2. Cách khắc phục

Nhược điểm trên dễ dàng khắc phục bằng cách đưa vào lớp DT hàm huỷ viết trong 5.3 (mục trên).

5.5. Lớp hình tròn đồ họa

Chương trình dưới đây gồm:

Lớp HT (hình tròn) với các thuộc tính:

```

int r;          // Bán kính
int m;          // Màu hình tròn

```

```
int xhien,yhien; // Vị trí hiển thị hình tròn trên màn hình
char *pht;       // Con trỏ trỏ tới vùng nhớ chứa ảnh hình tròn
int hienmh;      // Trạng thái hiện (hienmh=1), ẩn (hienmh=0)
```

Các phương thức:

+ Hàm tạo không đối

```
HT();
```

thực hiện việc gán giá trị bằng 0 cho các thuộc tính của lớp.

+ Hàm tạo có đối

```
HT(int r1,int m1=15);
```

thực hiện các việc:

- Gán r1 cho r, m1 cho m
- Cấp phát bộ nhớ cho pht
- Vẽ hình tròn và lưu ảnh hình tròn vào vùng nhớ của pht

+ Hàm hủy

```
~HT();
```

thực hiện các việc:

- Xoá hình tròn khỏi màn hình (nếu đang hiển thị)
- Giải phóng bộ nhớ đã cấp cho pht

+ Phương thức

```
void hien(int x, int y);
```

có nhiệm vụ hiển thị hình tròn tại (x,y)

+ Phương thức

```
void an();
```

có nhiệm vụ làm ẩn hình tròn

Các hàm độc lập:

```
void ktdh();           //Khởi tạo đồ họa
```

```
void ve_bau_troi();    // Vẽ bầu trời đầy sao
```

```
void ht_di_dong_xuong(); // Vẽ một cặp 2 hình tròn di
                        // chuyển xuống
```

```
void ht_di_dong_len();  // Vẽ một cặp 2 hình tròn di
                        // chuyển lên trên
```

Nội dung chương trình là tạo ra các chuyển động xuống và lên của các hình tròn.

```
//CT4_09.CPP
```

```
// Lop do hoa
```

```
// Ham huy
```

```
// Trong ham huy co the goi PT khac
```

```

#include <conio.h>
#include <iostream.h>
#include <math.h>
#include <stdlib.h>
#include <graphics.h>
#include <dos.h>
void kt_dh();
void ve_hau_loi();
void ht_di_dong_xuong();
void ht_di_dong_len();
int xmax,ymax;
class HT
{
    private:
        int r,m ;
        int xhien,yhien;
        char *pht;
        int hienmh;
    public:
        HT();
        HT(int r1,int m1=15);
        ~HT();
        void hien(int x, int y);
        void an();
};

HT::HT()
{
    r=m=hienmh=0;
    xhien=yhien=0;
    pht=NULL;
}

HT::HT(int r1,int m1)
{
    r=r1; m=m1; hienmh=0;
    xhien=yhien=0;
    if (r<0) r=0;
    if (r==0)
    {
        pht=NULL;
    }
    else
    {
        int size; char *pmh;
        size = imagesize(0,0,r+r,r+r);
        pmh = new char[size];
        getimage(0,0,r+r,r+r,pmh);
    }
}

```

```

        setcolor(m);
        circle(r,r,r);
        setfillstyle(1,m);
        floodfill(r,r,m);
        pht = new char[size];
        getimage(0,0,r+r,r+r,pht);
        putimage(0,0,pmh,COPY_PUT);
        delete pmh;
        pmh=NULL;
    }
}

void HT::hien(int x, int y)
{
    if (pht!=NULL && !hienmh) // chua hien
    {
        hienmh=1;
        xhien=x; yhien=y;
        putimage(x,y,pht,XOR_PUT);
    }
}

void HT::an()
{
    if (hienmh) // dang hien
    {
        hienmh=0;
        putimage(xhien,yhien,pht,XOR_PUT);
    }
}

HT::~HT()
{
    an();
    if (pht!= NULL)
    {
        delete pht;
        pht=NULL;
    }
}

void ktdh()
{
    int mh=0,mode=0;
    initgraph(&mh,&mode,"");
    xmax = getmaxx();
    ymax = getmaxy();
}

void vc_bau_troi()

```

```

{
    for (int i=0;i<2000;++i)
        putpixel(random(xmax), random(ymax), 1+random(15));
}
void ht_di_dong_xuong()
{
    HT h(50,4);
    HT u(60,15);
    h.hien(0,0);
    u.hien(40,0);
    for (int x=0;x<=340;x+=10)
    {
        h.an();
        u.an();
        h.hien(x,x);
        delay(200);
        u.hien(x+40,x);
        delay(200);
    }
}
void ht_di_dong_len()
{
    HT h(50,4);
    HT u(60,15);
    h.hien(340,340);
    u.hien(380,340);
    for (int x=340;x>=0;x-=10)
    {
        h.an();
        u.an();
        h.hien(x,x);
        delay(200);
        u.hien(x+40,x);
        delay(200);
    }
}
void main()
{
    ktdh();
    ve_bau_troi();
    ht_di_dong_xuong();
    ht_di_dong_len();
    getch();
    closegraph();
}

```

Các nhận xét:

1. Trong thân hàm huỷ gọi tới phương thức an().

2. Điều gì xảy ra khi bỏ đi hàm huỷ:

+ Khi gọi hàm `ht_di_dong_xuong()` thì có 2 đối tượng kiểu HT được tạo ra. Trong thân hàm sử dụng các đối tượng này để vẽ các hình tròn di chuyển xuống. Khi thoát khỏi hàm thì 2 đối tượng (tạo ra ở trên) được giải phóng. Vùng nhớ của các thuộc tính của chúng bị thu hồi, nhưng vùng nhớ cấp phát cho thuộc tính `pht` chưa được giải phóng và ảnh của 2 hình tròn (ở phía dưới màn hình) vẫn không được cất đi.

+ Điều tương tự xảy ra sau khi ra khỏi hàm `ht_di_dong_len()` : vùng nhớ cấp phát cho thuộc tính `pht` chưa được giải phóng và ảnh của 2 hình tròn (ở phía trên màn hình) vẫn không được thu dọn.

§6. TOÁN TỬ GÁN

6.1. Toán tử gán mặc định

Toán tử gán (cho lớp) là một trường hợp đặc biệt so với các toán tử khác. Nếu trong lớp chưa định nghĩa một phương thức toán tử gán thì Trình biên dịch sẽ phát sinh một toán tử gán mặc định để thực hiện câu lệnh gán 2 đối tượng của lớp, ví dụ:

```
HT h1, h2(100,6);
```

```
h1 = h2 ; // Gán h2 cho h1
```

Toán tử gán mặc định sẽ sao chép đối tượng nguồn (h2) vào đối tượng đích (h1) theo từng bit một.

Trong đa số các trường hợp khi lớp không có các thành phần con trỏ hay tham chiếu thì toán tử gán mặc định là đủ dùng và không cần định nghĩa một phương thức toán tử gán cho lớp. Nhưng đối với các lớp có thuộc tính con trỏ như lớp DT (đa thức), lớp HT (hình tròn) thì toán tử gán mặc định không thích hợp và việc xây dựng toán tử gán là cần thiết.

6.2. Cách viết toán tử gán

Cũng giống như các phương thức khác, phương thức toán tử gán dùng đối con trỏ this để biểu thị đối tượng đích và dùng một đối tượng mình để biểu thị đối tượng nguồn. Vì trong thân của toán tử gán không nên làm việc với bản sao của đối tượng nguồn, mà phải làm việc trực tiếp với đối tượng nguồn, nên kiểu đối tượng mình nhất thiết phải là kiểu tham chiếu đối tượng.

Phương thức toán tử gán có thể có hoặc không có giá trị trả về. Nếu

không có giá trị trả về (kiểu void), thì khi viết chương trình không được phép viết câu lệnh gán liên tiếp nhiều đối tượng, như:

```
u = v = k = h ;
```

Nếu phương thức toán tử gán trả về tham chiếu của đối tượng nguồn, thì có thể dùng toán tử gán để thực hiện các phép gán liên tiếp nhiều đối tượng.

Ví dụ đối với lớp HT (trong mục trước), có thể xây dựng toán tử gán như sau:

```
void HT::operator=(const HT &h)
{
    r = h.r ; m = h.m ;
    xhien = yhien = 0;
    hienmh = 0 ;
    if (h.pht==NULL)
        pht = NULL;
    else
    {
        int size;
        size = imagesize(0,0,r+r,r+r);
        pht = new char[size];
        memcpy(pht,h.pht,size);
    }
}
```

Với toán tử gán này, chỉ cho phép gán đối tượng nguồn cho một đối tượng đích.

Như vậy câu lệnh sau là sai:

```
HT u, v, h ;
u = v = h ;
```

Bây giờ ta sửa lại toán gán để nó trả về tham chiếu đối tượng nguồn như sau:

```
const HT & HT::operator=(const HT &h)
{
    r = h.r ; m = h.m ;
    xhien = yhien = 0;
    hienmh = 0 ;
    if (h.pht==NULL)
        pht = NULL;
    else
    {
        int size;
        size = imagesize(0,0,r+r,r+r);
        pht = new char[size];
        memcpy(pht,h.pht,size);
    }
}
```

```

    }
    return h ;
}

```

Với toán tử gán mới này, ta có thể viết câu lệnh để gán đối tượng nguồn cho nhiều đối tượng đích. Như vậy các câu lệnh sau là được:

```

HT u, v, h ;
u = v = h ;

```

6.3. Toán tử gán và hàm tạo sao chép

+ Toán tử gán không tạo ra đối tượng mới, chỉ thực hiện phép gán giữa 2 đối tượng đã tồn tại.

+ Hàm tạo sao chép được dùng để tạo một đối tượng mới và gán nội dung của một đối tượng đã tồn tại cho đối tượng mới vừa tạo.

+ Nếu đã xây dựng toán tử gán mà lại dùng hàm tạo sao chép mặc định thì chưa đủ, vì việc khởi gán trong câu lệnh khai báo sẽ không gọi tới toán tử gán mà lại gọi tới hàm tạo sao chép.

+ Như vậy đối với lớp có thuộc tính con trỏ, thì ngoài hàm tạo, cần xây dựng thêm:

- Hàm hủy
- Hàm tạo sao chép
- Phương thức toán tử gán

Chú ý: Không phải mọi câu lệnh chứa có dấu = đều gọi đến toán tử gán. Cần phân biệt 3 trường hợp:

1. Câu lệnh new (chứa dấu =) sẽ gọi đến hàm tạo, ví dụ:

```
HT *h= new HT(50,6); // gọi đến hàm tạo có đối
```

2. Câu lệnh khai báo và khởi gán (dùng dấu =) sẽ gọi đến hàm tạo sao chép, ví dụ:

```
HT k=*h; // gọi đến hàm tạo sao chép
```

3. Câu lệnh gán sẽ gọi đến toán tử gán, ví dụ:

```
HT u;
u=*h; // gọi đến phương thức toán tử gán
```

6.4. Ví dụ minh họa

Chương trình dưới đây định nghĩa lớp HT (hình tròn) và minh họa:

- + Hàm tạo và hàm hủy
- + Phương thức toán tử gán có kiểu tham chiếu
- + Hàm tạo sao chép
- + Cách dùng con trỏ this trong hàm tạo sao chép

+ Cách dùng con trỏ `_new_handler` để kiểm tra việc cấp phát bộ nhớ.

```
//CT4_10.CPP
// Lớp đồ họa
// Hàm hủy
// toán tử gán - trả về tham chiếu
// Hàm tạo sao chép
// Trong hàm hủy có thể gọi PT khác
#include <conio.h>
#include <iostream.h>
#include <stdlib.h>
#include <graphics.h>
#include <new.h>
#include <mem.h>
static void kiem_tra_bo_nho() ;
void ktdh();
int xmax,ymax;
void kiem_tra_bo_nho()
{
    outtextxy(1,1,"LOI BO NHO");
    getch();
    closegraph();
    exit(1);
}
class HT
{
private:
    int r,m ;
    int xhien,yhien;
    char *pht;
    int hienmh;
public:
    HT();
    HT(int r1,int m1=15);
    HT(const HT &h);
    ~HT();
    void hien(int x, int y);
    void an();
    const HT &operator=(const HT &h);
};
const HT & HT::operator=(const HT &h)
{
    // outtextxy(1,1,"Gan"); getch();
    r = h.r ;
    m = h.m ;
    xhien = yhien = 0;
    hienmh = 0 ;
    if (h.pht!=NULL)
        pht = NULL;
    else
        {
```

```

        int size;
        size = imagesize(0,0,r+r,r+r);
        pht = new char[size];
        memcpy(pht,h.pht,size);
    }
    return h;
}
HT::HT(const HT &h)
{
    //outtextxy(300,1,"constructor sao chep");
    getch();
    *this = h;
}
HT::HT()
{
    r=m=hienmh=0;
    xhien=yhien=0;
    pht=NULL;
}
HT::HT(int r1,int m1)
{
    r=r1; m=m1;
    hienmh=0;
    xhien=yhien=0;
    if (r<0) r=0;
    if (r==0)
    {
        pht=NULL;
    }
    else
    {
        int size; char *pmh;
        size = imagesize(0,0,r+r,r+r);
        pmh = new char[size];
        getimage(0,0,r+r,r+r,pmh);
        setcolor(m);
        circle(r,r,r);
        setfillstyle(1,m);
        floodfill(r,r,m);
        pht = new char[size];
        getimage(0,0,r+r,r+r,pht);
        putimage(0,0,pmh,COPY_PUT);
        delete pmh;
        pmh=NULL;
    }
}
void HT::hien(int x, int y)
{
    if (pht!=NULL && !hienmh) // chua hien
    {
        hienmh=1;
        xhien=x;
    }
}

```

```

        yhien=y;
        putimage(x,y,pht,XOR_PUT);
    }
}
void HT::an()
{
    if (hienmh) // đang hien
    {
        hienmh=0;
        putimage(xhien,yhien,pht,XOR_PUT);
    }
}
HT::~HT()
{
    an();
    if (pht!=NULL)
    {
        delete pht;
        pht=NULL;
    }
}
void ktdh()
{
    int mh=0,mode=0;
    initgraph(&mh,&mode,"");
    xmax = getmaxx();
    ymax = getmaxy();
}
void main()
{
    _new_handler = kiem_tra_bo_nho ;
    ktdh();
    HT *h= new HT(50,6); // gọi hàm tạo có đối
    h->hien(100,200);
    HT k=*h; // gọi hàm tạo sao chép
    k.hien(200,200);
    HT t,v,u;
    t = v = u = *h; // gọi toán tử gán
    u.hien(300,200);
    v.hien(400,200);
    t.hien(500,200);
    getch();
    closegraph();
}

```

6.5. Vai trò của phương thức toán tử gán

Chương trình trên sẽ vẽ 5 hình tròn trên màn hình. Điều gì sẽ xảy ra nếu bỏ đi phương thức toán tử gán và hàm tạo sao chép?

- + Nếu bỏ cả hai, thì chỉ xuất hiện một hình tròn tại vị trí (100,200).

+ Nếu bỏ toán tử gán (giữ hàm tạo sao chép) thì chỉ xuất hiện 2 hình tròn tại các vị trí (100,200) và (200,200).

+ Nếu bỏ hàm tạo sao chép (giữ toán tử gán) thì xuất hiện 4 hình tròn.

§7. PHÂN LOẠI PHƯƠNG THỨC, PHƯƠNG THỨC INLINE

7.1. Phân loại các phương thức

Có thể chia phương thức thành các nhóm:

1. Các phương thức thông thường
2. Các phương thức dùng để xây dựng và huỷ bỏ đối tượng gồm:
 - + Hàm tạo không đối,
 - + Hàm tạo có đối
 - + Hàm tạo sao chép
 - + Hàm huỷ
3. Các phương thức toán tử

7.2. Con trỏ this

Mọi phương thức đều dùng con trỏ this như đối thứ nhất (đối ẩn). Ngoài ra trong phương thức có thể đưa vào các đối tượng mình được khai báo như đối của hàm.

+ Với các phương thức thông thường, thì đối ẩn biểu thị đối tượng chủ thể trong lời gọi phương thức.

+ Với các hàm tạo, thì đối ẩn biểu thị đối tượng mới được hình thành.

+ Với các hàm huỷ, thì đối ẩn biểu thị đối tượng sắp bị huỷ bỏ.

+ Với các phương thức toán tử, thì đối ẩn biểu thị toán hạng đối tượng thứ nhất.

7.3. Phương thức inline

Có 2 cách để biên dịch phương thức theo kiểu inline:

Cách 1: Xây dựng phương thức bên trong định nghĩa lớp.

Cách 2: Thêm từ khoá inline vào định nghĩa phương thức (bên ngoài định nghĩa lớp).

Chú ý là chỉ các phương thức ngắn không chứa các câu lệnh phức tạp (như chu trình, goto, switch, đệ quy) mới có thể trở thành inline. Nếu có ý định biên dịch theo kiểu inline các phương thức chứa các câu lệnh phức tạp nói trên, thì Trình biên dịch sẽ báo lỗi.

Trong chương trình dưới đây, tất cả các phương thức của lớp PS (phân số) đều là phương thức inline

```

//CT4_11.CPP
// Lop PS
// Inline
#include <conio.h>
#include <iostream.h>
class PS
{
    private:
        int t,m ;
    public:
        PS()
        {
            t=0;
            m=1;
        }
        PS(int t1, int m1);
        void nhap();
        void in();
        PS operator*=(PS p2)
        {
            t*=p2.t;
            m*=p2.m;
            return *this;
        }
};
inline PS::PS(int t1, int m1)
{
    t=t1;
    m=m1;
}
inline void PS::nhap()
{
    cout << "\nNhap tu va mau: " ;
    cin >> t >> m;
}
inline void PS::in()
{
    cout << "\nPS = " << t << "/" << m ;
}
void main()
{
    PS q,p,s(3,5);
    cout << "\n Nhap PS p";
    p.nhap();
    s.in();
    p.in();
    q = p*s;
    p.in();
    q.in();
    getch();
}

```

§8. HÀM TẠO VÀ ĐỐI TƯỢNG THÀNH PHẦN

8.1. Lớp bao, lớp thành phần

Một lớp có thuộc tính là đối tượng của lớp khác gọi là lớp bao, ví dụ:

```
class A
{
    private:
        int a, b;
        ...
};
class B
{
    private:
        double x, y, z;
        ...
};
class C
{
    private:
        int m, n;
        A u;
        B p, q;
        ...
};
```

Trong ví dụ trên thì:

C là lớp bao

A, B là các lớp thành phần (của C)

8.2. Hàm tạo của lớp bao

+ Chú ý là trong các phương thức của lớp bao không cho phép truy nhập trực tiếp đến các thuộc tính của các đối tượng của các lớp thành phần.

+ Vì vậy, khi xây dựng hàm tạo của lớp bao, phải sử dụng các hàm tạo của lớp thành phần để khởi gán cho các đối tượng thành phần của lớp bao.

Ví dụ khi xây dựng hàm tạo của lớp C, cần dùng các hàm tạo của lớp A để khởi gán cho đối tượng thành phần u và dùng các hàm tạo của lớp B để khởi gán cho các đối tượng thành phần p, q.

8.3. Cách dùng hàm tạo của lớp thành phần để xây dựng hàm tạo của lớp bao

+ Để dùng hàm tạo (của lớp thành phần) khởi gán cho đối tượng thành phần của lớp bao, ta sử dụng mẫu:

tên_đối_tượng(danh sách giá trị)

+ Các mẫu trên cần viết bên ngoài thân hàm tạo, ngay sau dòng đầu tiên. Nói một cách cụ thể hơn, hàm tạo sẽ có dạng:

```

tên_lớp(danh sách đối) : tên_đối_tượng(danh sách giá trị),
...
tên_đối_tượng( danh sách giá trị)
{
    // Các câu lệnh trong thân hàm tạo
}

```

Chú ý là các dấu ngoặc sau tên đối tượng luôn luôn phải có, ngay cả khi danh sách giá trị bên trong là rỗng.

+ Danh sách giá trị lấy từ danh sách đối. Dựa vào danh sách giá trị, Trình biên dịch sẽ biết cần dùng hàm tạo nào để khởi gán cho đối tượng. Nếu danh sách giá trị là rỗng thì hàm tạo không đối sẽ được sử dụng.

+ Các đối tượng muốn khởi gán bằng hàm tạo không đối có thể bỏ qua, không cần phải liệt kê trong hàm tạo. Nói cách khác: Các đối tượng không được liệt kê trên dòng đầu hàm tạo của lớp bao, đều được khởi gán bằng hàm tạo không đối của lớp thành phần.

Ví dụ:

```

class A
{
    private:
        int a, b;
    public:
        A()
        {
            a=b=0;
        }
        A(int a1, int b1)
        {
            a = a1; b = b1;
        }
        ...
} ;
class B
{
    private:
        double x, y, z;
    public:
        B()
        {
            x = y = z = 0.0 ;
        }
        B(double x1, double y1)
        {
            x = x1;
            y = y1;
        }
} ;

```

```

        z = 0.0 ;
    }
    B(double x1, double y1, double z1)
    {
        x = x1;
        y = y1;
        z = z1 ;
    }
    ...
} ;
class C
{
    private:
        int m, n;
        A   u, v;
        B   p, q, r;
    public:
        C(int m1, int n1, int a1, int b1, double x1, double y1,
double x2, double y2, double z2) : u(), v(a1,b1), q(x1,y1),
r(x2,y2,z2)
        {
            m = m1 ;
            n = n1;
        }
} ;

```

Trong hàm tạo nói trên của lớp C, thì các đối tượng thành phần được khởi gán như sau:

- u được khởi gán bằng hàm tạo không đối của lớp A
- v được khởi gán bằng hàm tạo 2 đối của lớp A
- q được khởi gán bằng hàm tạo 2 đối của lớp B
- r được khởi gán bằng hàm tạo 3 đối của lớp B
- p (không có mặt) được khởi gán bằng hàm tạo không đối của lớp B

8.4. Sử dụng các phương thức của lớp thành phần

Mặc dù lớp bao có các thành phần đối tượng, nhưng trong lớp bao lại không được phép truy nhập đến các thuộc tính của các đối tượng này. Vì vậy giải pháp thông thường là:

- + Trong các lớp thành phần, xây dựng sẵn các phương thức để có thể lấy ra các thuộc tính của lớp.
- + Trong lớp bao dùng các phương thức của lớp thành phần để nhận các thuộc tính của các đối tượng thành viên cần dùng đến.

8.5. Các ví dụ

Hai chương trình dưới đây minh họa các điều đã nói trong các mục trên.

Ví dụ 1:

Trong ví dụ này xét 2 lớp:

DIEM (Điểm) và DT (Đoạn thẳng)

Lớp DIEM là lớp thành phần của lớp DT

```
//CT4 12.CPP
// Thuộc tính đối tượng
#include <conio.h>
#include <iostream.h>
class DIEM
{
private:
    int x,y ;
public:
    DIEM()
    {
        x=y=0;
    }
    DIEM(int x1, int y1)
    {
        x= x1; y=y1;
    }
    void in()
    {
        cout << "(" << x << "," << y << ")" ;
    }
};
class DT
{
private:
    DIEM d1, d2;
    int m;
public:
    DT() : d1(), d2()
    {
        m 0;
    }
    DT(int m1,int x1, int y1, int x2,
        int y2) : d1(x1,y1), d2(x2,y2)
    {
        m=m1;
    }
    DT(int m1,DIEM t1, DIEM t2)
    {
        m=m1;
        d1 = t1;
        d2 = t2;
    }
    void in()
```

```

        {
            cout << "\n Diem dau : "; d1.in();
            cout << "\n Diem cuoi: "; d2.in();
            cout << "\n Mau : " << m;
        }
    };
void main()
{
    DT u, v(1,100,100,200,200),
        s(2,DIEM(300,300),DIEM(400,400));
    clrscr();
    u.in();
    v.in();
    s.in();
    getch();
}

```

Ví dụ 2:

Trong ví dụ này xét 3 lớp:

Diem (Điểm)

DTron (Đường tròn)

HTron (Hình tròn)

Lớp DTron có một lớp thành phần là lớp Diem.

Lớp HTron có 2 lớp thành phần là lớp DTron và lớp Diem.

Trong lớp DTron đưa vào phương thức vẽ đường tròn.

Trong lớp HTron đưa vào phương thức vẽ và tô màu hình tròn.

Khi xây dựng phương thức của lớp bao cần sử dụng các phương thức của lớp thành phần.

```

//CT4_13.CPP
// Thuoc tinh doi tuong
#include <conio.h>
#include <iostream.h>
#include <graphics.h>
class Diem
{
private:
    int x,y ;
public:
    Diem()
    {
        x=y=0;
    }
    Diem(int x1, int y1)
    {
        x= x1;
    }
}

```

```

        y=y1;
    }
    int getx()
    {
        return x;
    }
    int gety()
    {
        return y;
    }
} ;
class DTron    // Duong tron
{
    private:
        Diem t ; // tam
        int r ;
        int m;
    public:
        DTron()
        {
            r=m=0;
        }
        DTron(int x1,int y1,int r1,int m1): t(x1,y1)
        {
            m m1;
            r=r1;
        }
        int mau()
        {
            return m;
        }
        void ve()
        {
            setcolor(m);
            circle(t.getx(),t.gety(),r);
        }
};
class HTron
{
    private:
        DTron dt;
        Diem d;
        int m;
    public:
        HTron()
        {
            m=0;
        }
        HTron(int x1, int y1, int r1, int m1, int x,
            int y, int mt): dt(x1,y1,r1,m1), d(x,y)
        {

```

```

        m = mt;
    }
    void ve()
    {
        dt.ve();
        setfillstyle(1,m);
        floodfill(d.getx(),d.gety(),dt.mau());
    }
} ;
void main()
{
    int mh=0, mode=0;
    initgraph(&mh,&mode,"");
    setbkcolor(1);
    DTtron dt(100,100,80,6);
    HTtron ht(300,300,150,15,300,300,4);
    dt.ve();
    ht.ve();
    getch();
    closegraph();
}

```

§9. CÁC THÀNH PHẦN TÍNH

9.1. Thành phần dữ liệu tính

+ Thành phần dữ liệu được khai báo bằng từ khoá static gọi là tĩnh, ví dụ:

```

class A
{
    private:
        static int ts; // Thành phần tĩnh
        int x;
        ...
} ;

```

+ Thành phần tĩnh được cấp phát một vùng nhớ cố định. Nó tồn tại ngay cả khi lớp chưa có một đối tượng nào cả.

+ Thành phần tĩnh là chung cho cả lớp, nó không phải là riêng của mỗi đối tượng. Ví dụ xét 2 đối tượng:

```

A u,v ; // Khai báo 2 đối tượng

```

thì giữa các thành phần x và ts có sự khác nhau như sau:

u.x và v.x có 2 vùng nhớ khác nhau

u.ts và v.ts chỉ là một, chúng cùng biểu thị một vùng nhớ

thành phần ts tồn tại ngay khi u và v chưa khai báo

+ Để biểu thị thành phần tĩnh, ta có thể dùng tên lớp, ví dụ: Đối với ts thì 3 cách viết sau là tương đương:

A::ts u.ts v.ts

+ Khai báo và khởi gán, giá trị cho thành phần tĩnh

Thành phần tĩnh sẽ được cấp phát bộ nhớ và khởi gán giá trị ban đầu bằng một câu lệnh khai báo đặt sau định nghĩa lớp (bên ngoài các hàm, kể cả hàm main), theo các mẫu:

```
int A::ts ;    // Khởi gán cho ts giá trị 0
int A::ts = 1234; // Khởi gán cho ts giá trị 1234
```

Chú ý: Khi chưa khai báo thì thành phần tĩnh chưa tồn tại. Ví dụ xét chương trình sau:

```
#include <conio.h>
#include <iostream.h>
class HDBH    // Hoá đơn bán hàng
{
    private:
        char *tenhang ;    // Tên hàng
        double tienban ;    // Tiền bán
        static int tshd ;    // Tổng số hoá đơn
        static double tstienban ;    // Tổng số tiền bán
    public:
        static void in()
        {
            cout <<"\n" << tshd;
            cout <<"\n" << tstienban;
        }
};
void main()
{
    HDBH::in();
    getch();
}
```

Các thành phần tĩnh tshd và tstienban chưa khai báo, nên chưa tồn tại. Vì vậy các câu lệnh in giá trị các thành phần này trong phương thức in là không logic. Khi dịch chương trình, sẽ nhận được các thông báo lỗi (tại phương thức in) như sau:

```
Undefined symbol HDBH::tshd in module ...
Undefined symbol HDBH::tstienban in module ...
```

Có thể sửa chương trình trên bằng cách đưa vào các lệnh khai báo các thành phần tĩnh tshd và tstienban như sau:

```
//CT4_14.CPP
// thành phần tĩnh
// Lớp HDBH (hoa đơn bán hàng)
#include <conio.h>
#include <iostream.h>
class HDBH
```

```

{
    private:
        int shd ;
        char *tenhang ;
        double tienban ;
        static int tshd ;
        static double tstienban ;
    public:
        static void in()
        {
            cout <<"\n" << tshd;
            cout <<"\n" << tstienban;
        }
} ;
int HDBH::tshd=5;
double HDBH::tstienban=20000.0;
void main()
{
    HDBH::in();
    getch();
}

```

9.2. Khi nào cần sử dụng các thành phần dữ liệu tĩnh

Xét một ví dụ về quản lý các hoá đơn bán hàng. Mỗi hoá đơn có: Tên hàng, số tiền bán. Rõ ràng các thuộc tính nói trên là riêng của mỗi hoá đơn. Mặt khác nếu chúng ta quan tâm đến tổng số hoá đơn đã bán, tổng số tiền đã bán, thì các thông tin này là chung. Vì vậy khi thiết kế lớp HDBH (hoá đơn bán hàng), thì ta có thể đưa vào 4 thành phần dữ liệu là:

tenhang (tên hàng)

tienban (tiền bán)

tshd (tổng số hoá đơn)

tstienban (tổng số tiền bán) Các thuộc tính tenhang và tienban là riêng của mỗi hoá đơn, nên chúng được chọn là các thuộc tính thông thường. Còn các thuộc tính tshd và tstienban là chung cho cả lớp nên chúng được chọn là các thuộc tính tĩnh.

9.3. Phương thức tĩnh

+ Có 2 cách viết phương thức tĩnh:

Cách 1: Dùng từ khoá static đặt trước định nghĩa phương thức viết bên trong định nghĩa lớp (như phương thức in() ví dụ cuối của mục 9.1).

Cách 2: Nếu phương thức xây dựng bên ngoài định nghĩa lớp, thì dùng từ khoá static đặt trước khai báo phương thức bên trong định nghĩa lớp. Chú ý không cho phép dùng từ khoá static đặt trước định nghĩa phương thức viết bên ngoài định nghĩa lớp.

+ Phương thức tĩnh là chung cho cả lớp, nó không lệ thuộc vào một đối tượng cụ thể, nó tồn tại ngay khi lớp chưa có đối tượng nào (xem ví dụ trong mục 9.1).

+ Lời gọi phương thức tĩnh:

Có thể xuất phát từ một đối tượng nào đó (như vẫn dùng khi gọi các phương thức khác)

Có thể dùng tên lớp

Ví dụ xét lớp HDBH trong mục 9.1 và xét các câu lệnh:

```
HDBH u, v;
```

Khi đó để gọi phương thức tĩnh in() có thể dùng một trong các lệnh sau:

```
u.in();
```

```
v.in();
```

```
HDBH::in();
```

+ Vì phương thức tĩnh là độc lập với các đối tượng, nên không thể dùng phương thức tĩnh để xử lý dữ liệu của các đối tượng chủ thể trong lời gọi phương thức tĩnh. Nói cách khác không cho phép truy nhập tới các thuộc tính (trừ thuộc tính tĩnh) trong thân phương thức tĩnh. Điều đó cũng đồng nghĩa với việc không cho phép dùng con trỏ this trong phương thức tĩnh.

Ví dụ nếu lập phương thức tĩnh in() để in các thuộc tính của lớp HDBH như sau:

```
class HDBH
{
    private:
        int shd ;
        char *tenhang ;
        double tienban ;
        static int tshd ;
        static double tstienban ;
    public:
        static void in()
        {
            cout <<"\n" << tshd;
            cout <<"\n" << tstienban;
            cout <<"\n" << tenhang;
            cout <<"\n" << tienban;
        }
};
```

thì sẽ bị lỗi, vì trong thân phương thức tĩnh không cho phép truy nhập đến các thuộc tính tenhang và tienban.

9.4. Ví dụ minh họa việc dùng phương thức tĩnh

Xét bài toán quản lý hoá đơn bán hàng. Mỗi hoá đơn có 2 dữ liệu là tên hàng và tiền bán. Sử dụng hàm tạo để tạo ra các hoá đơn, dùng hàm huỷ để bỏ đi (loại đi) các hoá đơn không cần lưu trữ, dùng một phương thức để sửa chữa nội dung hoá đơn (nhập lại tiền bán). Vấn đề đặt ra là sau một số thao tác: Tạo, sửa và huỷ hoá đơn thì tổng số hoá đơn còn lại là bao nhiêu và tổng số tiền trên các hoá đơn còn lại là bao nhiêu?

Chương trình dưới đây nhằm đáp ứng yêu cầu đặt ra.

```
//CT4_14.CPP
// thanh phan tinh
// Lop HDBH (hoa don ban hang)
#include <conio.h>
#include <iostream.h>
class HDBH
{
    private:
        char *tenhang ;
        double tienban ;
        static int tshd ;
        static double tstienban ;
    public:
        HDBH(char *tenhang1=NULL, double tienban1=0.0 )
        {
            tienban=tienban1;
            tenhang=tenhang1;
            tshd++;
            tstienban += tienban;
        }
        ~HDBH()
        {
            --tshd;
            tstienban -= tienban;
        }
        void sua();
        static void in();
    } ;
int HDBH::tshd=0;
double HDBH::tstienban=0;
void HDBH::in()
{
    cout << "\n\nTong so hoa don: " << tshd;
    cout << "\nTong so tien: " << tstienban;
}
void HDBH::sua()
{
    cout << "\n\nTen hang: " << tenhang;
```

```

cout << "\nTien ban : " << tienban;
tstienban -= tienban;
cout << "\nSua tien ban thanh : " ;
cin >> tienban;
tstienban += tienban;
}
void main()
{
    HDBH *h1 = new HDBH("Xi mang",2000);
    HDBH *h2 = new HDBH("Sat thep",3000);
    HDBH *h3 = new HDBH("Ti vi",4000);
    clrscr();
    HDBH::in();
    getch();
    delete h1;
    HDBH::in();
    getch();
    h2->sua();
    HDBH::in();
    getch();
    delete h3;
    HDBH::in();
    getch();
}

```

§10. MẢNG ĐỐI TƯỢNG

10.1. Khai báo

Có thể dùng tên lớp để khai báo mảng đối tượng (giống như khai báo mảng int, float, char, ...) theo mẫu:

```
Tên_lớp  tên_mảng[kích_cỡ];
```

Ví dụ giả sử đã định nghĩa lớp DIEM (Điểm), khi đó có thể khai báo các mảng đối tượng DIEM như sau:

```
DIEM  a[10], b[20];
```

Ý nghĩa: a là mảng kiểu DIEM gồm 10 phần tử

b là mảng kiểu DIEM gồm 20 phần tử

Câu lệnh khai báo mảng sẽ gọi tới hàm tạo không đối để tạo các phần tử mảng. Trong ví dụ trên, hàm tạo được gọi 30 lần để tạo 30 phần tử mảng đối tượng.

10.2. Khai báo và khởi gán

Để khai báo mảng và khởi gán giá trị cho các phần tử mảng đối tượng, cần dùng các hàm tạo có đối theo mẫu sau:

```
Tên_lớp  tên_mảng[kích_cỡ] = {Tên_lớp(các tham số), ...,
                               Tên_lớp(các tham số)};
```

Ví dụ giả sử lớp DIEM đã định nghĩa:

```
class DIEM
{
    private:
        int x, y ;
    public:
        DIEM()
        {
            x = y = 0;
        }
        DIEM(int x1, int y1)
        {
            x = x1;
            y = y1;
        }
        void nhaps1();
        void ve_doan_thang(DIEM d2, int mau) ;
};
```

Khi đó các câu lệnh khai báo dưới đây đều đúng:

```
DIEM d[5]    {DIEM(1,1),DIEM(200,200)};
DIEM u[] = {DIEM(1,1),DIEM(200,200)};
```

Ý nghĩa của các lệnh này như sau:

Câu lệnh đầu gọi tới hàm tạo 2 lần để khởi gán cho d[1], d[2] và gọi tới hàm tạo không đối 3 lần để tạo các phần tử d[3], d[4] và d[5].

Câu lệnh sau gọi tới hàm tạo 2 lần để khởi gán cho u[1], u[2]. Mảng u sẽ gồm 2 phần tử.

10.3. Biểu thị thành phần của phần tử mảng

Để biểu thị thuộc tính của phần tử mảng đối tượng, ta viết như sau:

Tên_mảng[chỉ_số].Tên_thuộc_tính

Để thực hiện phương thức đối với phần tử mảng ta viết như sau:

Tên_mảng[chỉ_số].Tên_phương_thức(danh_sách_tham_số);

Ví dụ để vẽ đoạn thẳng nối điểm d[1] với d[2] theo màu đỏ, ta có thể dùng phương thức ve_doan_thang như sau:

```
d[1].ve_doan_thang(d[2], 4); // Thực hiện phương thức
                             // đối với d[1]
```

10.4. Ví dụ

Chương trình dưới đây đưa vào lớp TS (thí sinh) và xét bài toán: Nhập một danh sách thí sinh, sắp xếp danh sách theo thứ tự giảm của tổng điểm. Chương trình minh họa:

- + Cách dùng mảng đối tượng.
- + Vai trò con trỏ this (trong phương thức hv(hoán vị)) .
- + Các hàm tạo, hàm hủy.
- + Vai trò của toán tử gán (nếu sử dụng phép gán mặc định chương trình sẽ cho kết quả sai).

```
//CT4_15.CPP
// mảng doi tuong
// Lop TS (thi sinh)
// Chu y vai tro cua toan tu gan
#include <conio.h>
#include <iostream.h>
#include <string.h>
class TS
{
private:
    char *ht;
    double td;
public:
    TS()
    {
        ht = new char[20];
        td = 0;
    }
    ~TS()
    {
        delete ht;
    }
    const TS &operator=(const TS &ts2)
    {
        this->td = ts2.td;
        strcpy(this->ht,ts2.ht);
        return ts2;
    }
    void nhap(int i);
    void in();
    double gettd()
    {
        return td;
    }
    void hv(TS &ts2)
    {
        TS tg;
        tg = *this ;
        *this = ts2 ;
    }
}
```

```

        ts2 = tg;
    }
} ;
void TS::in()
{
    cout << "\nHo ten: " << ht << " Tong diem: " << td;
}
void TS::nhap(int i)
{
    cout << "\nNhap thi sinh " << i ;
    cout << "\nHo ten: " ; cin >> ht;
    cout << "Tong diem: " ; cin >> td;
}
void main()
{
    TS ts[100];
    int n, i, j;
    clrscr();
    cout << "\n So thi sinh: " ;
    cin >> n;
    for (i=1; i<= n; ++i)
        ts[i].nhap(i);
    cout << "\n Danh sach nhap vao:";
    for (i=1; i<= n; ++i)
        ts[i].in();
    for (i=1; i<n ; ++i)
        for (j=i+1 ; j<=n; ++j)
            if (ts[i].gettd() < ts[j].gettd())
                ts[i].hv(ts[j]);
    cout << "\n\n Danh sach sau khi sap xep:";
    for (i=1; i<= n; ++i)
        ts[i].in();
    getch();
}

```

§11. CẤP PHÁT BỘ NHỚ CHO ĐỐI TƯỢNG

11.1. Cách cấp phát bộ nhớ cho đối tượng

Có thể dùng new và tên lớp để cấp phát một vùng nhớ cho một hoặc một dãy các đối tượng. Bộ nhớ cấp phát được quản lý bởi một con trỏ kiểu đối tượng. Ví dụ sau khi đã định nghĩa lớp DIEM như trong mục trên, ta có thể thực hiện các lệnh cấp phát bộ nhớ như sau:

```

int n = 10;
DIEM *p, *q, *r;

```

```

p   new DIEM; // Cấp phát bộ nhớ cho một đối tượng
q = new DIEM[n]; //Cấp phát bộ nhớ cho n đối tượng
r = new DIEM(200,100); // Cấp phát bộ nhớ và khởi gán
                        // cho một đối tượng

```

11.2. Làm việc với đối tượng thông qua con trỏ

+ Giả sử con trỏ p trỏ tới vùng nhớ của một đối tượng nào đó. Khi đó:

- Để biểu thị một thành phần (thuộc tính hoặc phương thức) của đối tượng, ta dùng mẫu viết sau:

```
p->tên_thành_phần
```

- Để biểu thị đối tượng, ta dùng mẫu viết sau:

```
*p
```

+ Giả sử con trỏ q trỏ tới địa chỉ đầu vùng nhớ của một dãy đối tượng. Khi đó:

- Để biểu thị một thành phần (thuộc tính hoặc phương thức) của đối tượng thứ i, ta dùng một trong các mẫu viết sau:

```
q[i].tên_thành_phần
```

```
(q+i)-> tên_thành_phần
```

- Để biểu thị đối tượng thứ i, ta dùng một trong các mẫu viết sau:

```
q[i]
```

```
*(q+i)
```

11.3. Bài toán sắp xếp thí sinh

Trong mục 10.4. đã sử dụng mảng đối tượng để giải quyết bài toán: Nhập một danh sách thí sinh, sắp xếp danh sách theo thứ tự giảm của tổng điểm. Dưới đây sẽ đưa ra phương án mới bằng cách dùng con trỏ và cấp phát bộ nhớ cho các đối tượng. Chương trình chỉ thay đổi hàm main() như sau:

```

void main()
{
    TS *ts;
    int n, i, j;
    clrscr();
    cout << "\n So thi sinh: " ;
    cin >> n;
    ts = new TS[n+1];
    for (i=1; i<= n; ++i)
        ts[i].nhap(i);
    cout << "\n Danh sach nhap vao:";
    for (i=1; i<= n; ++i)
        ts[i].in();
    for (i=1; i<n ; ++i)
        for (j=i+1 ; j<=n; ++j)

```

```

        if (ts[i].gettd() < ts[j].gettd())
            ts[i].hv(ts[j]);
        cout << "\n\n Danh sach sau khi sap xep:";
        for (i=1; i<= n; ++i)
            ts[i].in();
        getch();
    }

```

Nhận xét: Sự khác biệt giữa hàm main mới và hàm main trong 10.4 là rất ít.

11.4. Danh sách móc nối

Chương trình dưới đây định nghĩa lớp tự trò TS (lớp có thuộc tính kiểu *TS). Lớp này được dùng để tổ chức danh sách móc nối. Chương trình nhập một danh sách thí sinh và chứa trong một danh sách móc nối. Sau đó duyệt trên danh sách này để in các thí sinh trúng tuyển. So với lớp TS nêu trong mục 10.4, lớp TS ở đây có một số điểm khác như sau:

+ Thêm thuộc tính:

```

    TS *dc; // Dùng để chứa địa chỉ của một
            // đối tượng kiểu TS

```

+ Thêm các phương thức:

```

    void setdc(TS *dc1); // Gán dc1 cho thuộc tính dc
    TS *getdc(); // Nhận giá trị của dc

```

+ Phương thức nhập trong chương trình trước có kiểu void nay sửa là:

```

    int nhap(int i);

```

Phương thức trả về 1 nếu họ tên nhập vào khác trống, trả về 0 nếu trái lại.

+ Bỏ đi các phương thức không dùng đến như: Toán tử gán, hoán vị.

```

//CT4_16.CPP
// Danh sách móc nối
// Lớp TS (thí sinh)
#include <conio.h>
#include <iostream.h>
#include <string.h>
#include <ctype.h>
#include <stdio.h>
class TS
{
    private:
        char *ht;
        double td;
        TS *dc;
    public:
        TS()

```

```

    {
        ht = new char[20];
        td = 0;
        dc=NULL;
    }
~TS()
{
    delete ht; dc=NULL;
}
int nhap(int i);
void in();
double gettd()
{
    return td;
}
void setdc(TS *dcl)
{
    dc=dcl;
}
TS *getdc()
{
    return dc;
}
} ;
void TS::in()
{
    cout << "\nlo ten: " << ht << " Tong diem: " << td;
}
int TS::nhap(int i)
{
    cout << "\nNhap thi sinh " << i;
    cout << "\nlo ten (Bấm Enter để kết thúc nhập): ";
    fflush(stdin);
    gets(ht);
    if (ht[0]==0) return 0;
    cout << "Tong diem: " ; cin >> td;
    dc=NULL;
    return 1;
}
void main()
{
    int i=0;
    TS *pdau, *p, *q;
    pdau=NULL;
    clrscr();
    while(1)
    {
        q=new TS;
        i++;
        if (q->nhap(i)==0)
        {

```

```

        delete q; break;
    }
    if (pdau==NULL)
        pdau = p = q;
    else
    {
        p->setdc(q) ;
        p = q;
    }
}
/* In */
double diemchuan;
cout << "\nDiem chuan: " ;
cin >> diemchuan;
cout << "\nDanh sach trung tuyen:" ;
p=pdau;
while (p!=NULL)
{
    if (p->gettd()>=diemchuan)
        p->in();
    p = p->getdc();
}
getch();
}

```

§12. ĐỐI TƯỢNG HẲNG, PHƯƠNG THỨC HẲNG

+ Cũng giống như các phần tử dữ liệu khác, một đối tượng có thể được khai báo là hằng bằng cách dùng từ khoá const. Ví dụ:

```

class DIEM
{
    private:
        int x, y;
        int m;
    public:
        DIEM()
        {
            x = y = m = 0;
        }
        DIEM(int x1, int y1, int m1=15)
        {
            x= x1;
            y= y1; m= m1;
        }
        ...
};

const DIEM d = DIEM(200,100); // Khai báo đối tượng hằng

```

+ Khi khai báo cần sử dụng các hàm tạo để khởi gán giá trị cho đối tượng hằng. Giá trị khởi tạo có thể là các hằng, các biến, các biểu thức và các hàm, ví dụ:

```
int x0=100, y0=50; m0 =4;
const DIEM d5 = DIEM(x0 + getmaxx() / 2, y0 +
                    getmaxy()/2, m0);
```

+ Các phương thức có thể sử dụng cho các đối tượng hằng là hàm tạo và hàm huỷ. Về lý thuyết các đối tượng hằng không thể bị thay đổi, mà chỉ được tạo ra hoặc huỷ bỏ đi.

Khi dùng một phương thức cho đối tượng hằng, thì CTBD (Chương trình biên dịch) sẽ cảnh báo (warning):

Non-const function called for const object

Tuy nhiên, chương trình EXE vẫn được tạo và khi thực hiện chương trình, thì nội dung các đối tượng hằng vẫn bị thay đổi. Chương trình dưới đây sẽ minh họa điều này. Chương trình đưa vào lớp PS (phân số). Phương thức toán tử ++ vẫn có thể làm thay đổi đối tượng hằng (mặc dù khi biên dịch có 3 cảnh báo).

```
//CT4_19.CPP
// doi tuong const
// Lop PS (phan so)
#include <conio.h>
#include <iostream.h>
#include <string.h>
#include <math.h>
class PS
{
private:
    int t,m;
public:
    PS()
    {
        t = m = 0;
    }
    PS(int t1, int m1)
    {
        t = t1; m = m1;
    }
    PS operator++()
    {
        t += m ;
        return *this ;
    }
    void in()
    {
        cout << "\nPS= " << t << "/" << m;
```

```

    }
    void nhap()
    {
        cout << "\n Nhap tu va mau: " ;
        cin >> t >> m;
    }
} ;

void main()
{
    int t1=-3, m1=5;
    const PS p = PS(abs(t1)+2,m1+2); // Khai báo
                                     // đối tượng hằng
    clrscr();
    p.in();
    ++p;
    p.in();
    getch();
}

```

+ Phương thức const

Để biến một phương thức thành const ta chỉ việc viết thêm từ khoá const vào sau dòng đầu của phương thức.

Chú ý: Nếu phương thức được khai báo bên trong và định nghĩa bên ngoài lớp, thì từ khoá const cần được bổ sung cả trong khai báo và định nghĩa phương thức.

Trong thân phương thức const không cho phép làm thay đổi các thuộc tính của lớp. Vì vậy việc dùng phương thức const cho các đối tượng hằng sẽ đảm bảo giữ nguyên nội dung của các đối tượng hằng.

Đương nhiên các phương thức const vẫn dùng được cho các đối tượng khác.

Ví dụ sau về lớp PS (phân số) minh họa việc dùng phương thức const.

```

// Đối tượng const
// Phương thức const
// Lớp PS (phân số)
#include <conio.h>
#include <iostream.h>
#include <string.h>
#include <math.h>
class PS
{
private:
    int t,m;
public:
    PS()
    {

```

```

        t = m = 0;
    }
    PS(int t1, int m1)
    {
        t = t1; m = m1;
    }
    PS operator++()
    {
        t += m ;
        return *this ;
    }
    void in() const ;
    void nhap()
    {
        cout << "\n Nhap tu va mau: " ;
        cin >> t >> m;
    }
} ;

void PS::in() const
{
    cout << "\nPS= " << t << "/" << m;
}

void main()
{
    int t1=-3, m1=5;
    const PS p = PS(abs(t1)+2,m1+2);
    PS q;
    clrscr();
    q.nhap();
    p.in();
    q.in();
    getch();
}

```

§13. HÀM BẠN, LỚP BẠN

13.1. Hàm bạn (xem mục §6, chương 3) của một lớp, tuy không phải là phương thức của lớp, nhưng có thể truy nhập đến các thành phần riêng (private) của lớp. Một hàm có thể là bạn của nhiều lớp.

13.2. Nếu lớp A được khai báo là bạn của lớp B thì tất cả các phương thức của A đều có thể truy nhập đến các thành phần riêng của lớp B. Một lớp có thể là bạn của nhiều lớp khác. Cũng có thể khai báo A là bạn của B và B là bạn của A.

13.3. Cách khai báo lớp bạn

Giả sử có 3 lớp A, B và C. Để khai báo lớp này là bạn của lớp kia, ta viết theo mẫu sau:

```
// Khai báo trước các lớp
class A;
class B ;
class C;
// Định nghĩa các lớp
class A
{
    ...
    friend class B ; // Lớp B là bạn của A
    friend class C ; // Lớp C là bạn của A
    ...
};
class B
{
    ...
    friend class A ; // Lớp A là bạn của B
    friend class C ; // Lớp C là bạn của B
    ...
};
class C
{
    ...
    friend class B ; // Lớp B là bạn của C
    ...
};
```

13.4. Ví dụ

Chương trình dưới đây có 2 lớp:

MT (ma trận vuông)

VT (véc tơ)

Lớp MT là bạn của VT và lớp VT là bạn của MT. Trong chương trình sử dụng các phương thức trùng tên:

2 phương thức nhập():

nhập ma trận

nhập véc tơ

2 phương thức in():

in ma trận

in véc tơ

4 phương thức tích():

tích ma trận với ma trận, kết quả là ma trận

tích ma trận với véc tơ, kết quả là véc tơ

tích véc tơ với ma trận, kết quả là véc tơ

tích véc tơ với véc tơ, kết quả là số thực

Nội dung chương trình là:

+ Nhập các ma trận A, B, C

+ Nhập các véc tơ

+ Tính tích $D = AB$

+ Tính tích $u = Dy$

+ Tính tích $v = xC$

+ Tính tích $s = vu$

```
//CT4_17.CPP
// Lop ban
// Lop MT, lop VT
#include <conio.h>
#include <iostream.h>
class MT;
class VT;
class MT
{
    private:
        double a[10][10];
        int n;
    public:
        friend class VT;
        MT()
        {
            n=0;
        }
        void nhap();
        void in();
        VT tinh(const VT &y);
        MT tinh(const MT &b) ;
} ;
class VT
{
    private:
        double x[10];
        int n;
    public:
        friend class MT;
        VT()
```

```

        {
            n=0;
        }
        void nhap();
        void in();
        VT tich(const MT &b);
        double tich(const VT &y) ;
    } ;
void MT::nhap()
{
    cout << "\n Cap ma tran: " ;
    cin >> n;
    for (int i=1; i<=n; ++i)
        for (int j=1; j<=n; ++j)
        {
            cout << "\nPhan tu hang " << i << "   cot " << j << "
" ;
            cin >> a[i][j];
        }
    }
void MT::in()
{
    for (int i=1; i<=n; ++i)
    {
        cout << "\n" ;
        for (int j=1; j<=n; ++j)
            cout << a[i][j] << " " ;
    }
}
void VT::nhap()
{
    cout << "\n Cap vec to: " ;
    cin >> n;
    for (int i=1; i<=n; ++i)
    {
        cout << "\nPhan tu thu " << i << " = " ;
        cin >> x[i];
    }
}
void VT::in()
{
    for (int i=1; i<=n; ++i)
        cout << x[i] << " " ;
}
VT MT::tich(const VT &y)
{

```

```

VT z;
int i, j;
for (i=1; i<=n; ++i)
{
    z.x[i] = 0.0 ;
    for (j=1; j<=n; ++j)
        z.x[i] += a[i][j]*y.x[j];
}
z.n = n;
return z;
}
MT MT::tich(const MT &b)
{
    MT c;
    int i, j, k;
    for (i=1; i<=n; ++i)
        for (j=1; j<=n; ++j)
        {
            c.a[i][j] = 0.0 ;
            for (k=1; k<=n; ++k)
                c.a[i][j] += a[i][k]*b.a[k][j];
        }
    c.n = n;
    return c;
}
VT VT::tich(const MT &b)
{
    VT z;
    int i, j;
    for (j=1; j<=n; ++j)
    {
        z.x[j] = 0.0 ;
        for (i=1; i<=n; ++i)
            z.x[j] += b.a[i][j]*x[i];
    }
    z.n = n;
    return z;
}
double VT::tich(const VT &y)
{
    double tg=0.0;
    for (int i=1; i<=n; ++i)
        tg += x[i]*y.x[i];
    return tg;
}
void main()

```

```

{
    MT a,b,c;
    VT x,y;
    clrscr();
    cout << "\nMa tran A";
    a.nhap();
    cout << "\nMa tran B";
    b.nhap();
    cout << "\nMa tran C";
    c.nhap();
    cout << "\nvec to X";
    x.nhap();
    cout << "\nvec to Y";
    y.nhap();
    MT d= a.tich(b);
    VT u = d.tich(y);
    VT v = x.tich(c);
    double s = v.tich(u);
    cout << "\n\nVec to v\n";
    v.in();
    cout << "\n\nMa tran D";
    d.in();
    cout << "\n\nVec to y\n";
    y.in();
    cout << "\n\nS= vDy = " << s;
    getch();
}

```

DẪN XUẤT VÀ THỪA KẾ

Có 2 khái niệm rất quan trọng đã làm nên toàn bộ thể mạnh của phương pháp lập trình hướng đối tượng đó là tính kế thừa (inheritance) và tính tương ứng bội (polymorphism). Tính kế thừa cho phép các lớp được xây dựng trên các lớp đã có. Trong chương này sẽ nói về sự thừa kế của các lớp.

§1. SỰ DẪN XUẤT VÀ TÍNH THỪA KẾ

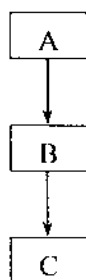
1.1. Lớp cơ sở và lớp dẫn xuất

Một lớp được xây dựng thừa kế một lớp khác gọi là lớp dẫn xuất. Lớp dùng để xây dựng lớp dẫn xuất gọi là lớp cơ sở.

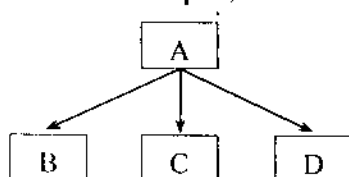
Lớp nào cũng có thể là một lớp cơ sở. Hơn thế nữa, một lớp có thể là cơ sở cho nhiều lớp dẫn xuất khác nhau. Đến lượt mình, lớp dẫn xuất lại có thể dùng làm cơ sở để xây dựng các lớp dẫn xuất khác. Ngoài ra một lớp có thể dẫn xuất từ nhiều lớp cơ sở.

Dưới đây là một số sơ đồ về quan hệ dẫn xuất của các lớp:

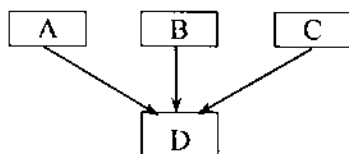
Sơ đồ 1: Lớp B dẫn xuất từ lớp A, lớp C dẫn xuất từ lớp B



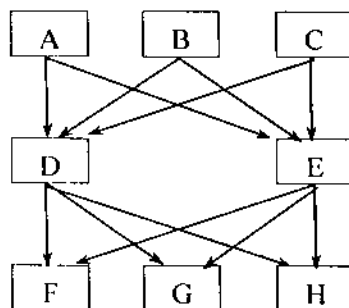
Sơ đồ 2: Lớp A là cơ sở của các lớp B, C và D



Sơ đồ 3: Lớp D dẫn xuất từ 3 lớp A, B, C



Sơ đồ 4: Lược đồ dẫn xuất tổng quát



Tính thừa kế: Một lớp dẫn xuất ngoài các thành phần của riêng nó, nó còn được thừa kế tất cả các thành phần của các lớp cơ sở có liên quan. Ví dụ trong sơ đồ 1 thì lớp C được thừa kế các thành phần của các lớp B và A. Trong sơ đồ 3 thì lớp D được thừa kế các thành phần của các lớp A, B và C. Trong sơ đồ 4 thì lớp G được thừa kế các thành phần của các lớp D, E, A, B và C.

1.2. Cách xây dựng lớp dẫn xuất

Giả sử đã định nghĩa các lớp A và B. Để xây dựng lớp C dẫn xuất từ A và B, ta viết như sau:

```

class C : public A, public B
{
    private:
        // Khai báo các thuộc tính
    public:
        // Các phương thức
} ;
  
```

1.3. Thừa kế private và public

Trong ví dụ trên, lớp C thừa kế public các lớp A và B. Nếu thay từ khoá public bằng private, thì sự thừa kế là private.

Chú ý: Nếu bỏ qua không dùng từ khoá thì hiểu là private, ví dụ nếu định nghĩa:

```

class C : public A, B
  
```

```

{
    private:
        // Khai báo các thuộc tính
    public:
        // Các phương thức
} ;

```

thì A là lớp cơ sở public của C, còn B là lớp cơ sở private của C.

Theo kiểu thừa kế public thì tất cả các thành phần public của lớp cơ sở cũng là các thành phần public của lớp dẫn xuất.

Theo kiểu thừa kế private thì tất cả các thành phần public của lớp cơ sở sẽ trở thành các thành phần private của lớp dẫn xuất.

1.4. Thừa kế các thành phần dữ liệu (thuộc tính)

Các thuộc tính của lớp cơ sở được thừa kế trong lớp dẫn xuất. Như vậy tập thuộc tính của lớp dẫn xuất sẽ gồm: các thuộc tính mới khai báo trong định nghĩa lớp dẫn xuất và các thuộc tính của lớp cơ sở.

Tuy vậy trong lớp dẫn xuất không cho phép truy nhập đến các thuộc tính private của lớp cơ sở.

Chú ý: Cho phép đặt trùng tên thuộc tính trong các lớp cơ sở và lớp dẫn xuất.

Ví dụ:

```

class A
{
    private:
        int a, b, c;
    public:
        ...
};
class B
{
    private:
        double a, b, x;
    public:
        ...
};
class C : public A, B
{
    private:
        char *a, *x ;
        int b ;
    public:
        ...
};

```

Khi đó lớp C sẽ có các thuộc tính:

A::a, A::b, A::c (kiểu int) - thừa kế từ A

B::a, B::b, B::x (kiểu double) - thừa kế từ B

a, x (kiểu char*) và b (kiểu int) - khai báo trong C .

Trong các phương thức của C chỉ cho phép truy nhập trực tiếp tới các thuộc tính khai báo trong C.

1.5. Thừa kế phương thức

Trừ:

+ Hàm tạo

+ Hàm huỷ

+ Toán tử gán

các phương thức (public) khác của lớp cơ sở được thừa kế trong lớp dẫn xuất.

Ví dụ: Trong chương trình dưới đây:

+ Đầu tiên định nghĩa lớp DIEM có:

Các thuộc tính x, y

Hai hàm tạo

Phương thức in()

+ Sau đó xây dựng lớp HINH_TRON dẫn xuất từ lớp DIEM, đưa thêm:

Thuộc tính r

Hai hàm tạo

Phương thức getR

Chú ý cách dùng hàm tạo của lớp cơ sở (lớp DIEM) để xây dựng hàm tạo của lớp dẫn xuất.

+ Trong hàm main:

Khai báo đối tượng h kiểu HINH_TRON

Sử dụng phương thức in() đối với h (sự thừa kế)

Sử dụng phương thức getR đối với h

```
//CT5-01
```

```
// Lop co so
```

```
#include <conio.h>
```

```
#include <iostream.h>
```

```
class DIEM
```

```
{
```

```
private:
```

```
double x, y;
```

```

public:
    DIEM()
    {
        x = y = 0.0;
    }
    DIEM(double x1, double y1)
    {
        x = x1; y = y1;
    }
    void in()
    {
        cout << "\nx= " << x << "   y= " << y;
    }
};

class HINH_TRON : public DIEM
{
private:
    double r;
public:
    HINH_TRON()
    {
        r = 0.0;
    }
    HINH_TRON(double x1, double y1,
               double r1): DIEM(x1,y1)
    {
        r = r1;
    }
    double getR()
    {
        return r;
    }
};

void main()
{
    HINH_TRON h(2.5,3.5,8);
    clrscr();
    cout << "\nHinh tron co tam: ";
    h.in();
    cout << "\nCo ban kinh= " << h.getR();
    getch();
}

```

1.6. Lớp cơ sở và đối tượng thành phần

Lớp cơ sở thường được xử lý giống như một thành phần kiểu đối tượng của lớp dẫn xuất. Ví dụ chương trình trong 1.5 có thể thay bằng một chương trình khác trong đó thay việc dùng lớp cơ sở DIEM bằng một thành phần kiểu DIEM trong lớp HINH_TRON. Chương trình mới có thể viết như sau:

```

//CT5-02
// lop co doi tuong thanh phan
#include <conio.h>
#include <iostream.h>
class DIEM
{
    private:
        double x, y;
    public:
        DIEM()
        {
            x = y = 0.0;
        }
        DIEM (double x1, double y1)
        {
            x = x1; y = y1;
        }
        void in()
        {
            cout << "\nx= " << x << " y= " << y;
        }
};

class HINH_TRON
{
    private:
        DIEM d;
        double r;
    public:
        HINH_TRON() : d()
        {
            r = 0.0;
        }
        HINH_TRON(double x1, double y1, double r1): d(x1,y1)
        {
            r = r1;
        }
        void in()
        {
            d.in();
        }
        double getR()
        {
            return r;
        }
};

void main()
{
    HINH_TRON h(2.5,3.5,8);
    clrscr();
    cout << "\nhinh tron co tam: ";
    h.in();
}

```

```
cout << "\nCo ban kinh  " << h.getR();
getch();
}
```

§2. HÀM TẠO, HÀM HUỖ ĐỐI VỚI TÍNH THỪA KẾ

2.1. Lớp dẫn xuất không thừa kế các hàm tạo, hàm huỷ, toán tử gán của các lớp cơ sở

2.2. Cách xây dựng hàm tạo của lớp dẫn xuất

+ Hàm tạo cần có các đối để khởi gán cho các thuộc tính (thành phần dữ liệu) của lớp.

+ Có thể phân thuộc tính làm 3 loại ứng với 3 cách khởi gán khác nhau:

1. Các thuộc tính mới khai báo trong lớp dẫn xuất. Trong các phương thức của lớp dẫn xuất có thể truy xuất đến các thuộc tính này. Vì vậy chúng thường được khởi gán bằng các câu lệnh gán viết trong thân hàm tạo.

2. Các thành phần kiểu đối tượng. Trong lớp dẫn xuất không cho phép truy nhập đến các thuộc tính của các đối tượng này. Vì vậy để khởi gán cho các đối tượng thành phần cần dùng hàm tạo của lớp tương ứng. Điều này đã trình bày trong mục §8 chương 4.

3. Các thuộc tính thừa kế từ các lớp cơ sở. Trong lớp dẫn xuất không được phép truy nhập đến các thuộc tính này. Vì vậy để khởi gán cho các thuộc tính nói trên, cần sử dụng hàm tạo của lớp cơ sở. Cách thức cũng giống như khởi gán cho các đối tượng thành phần, chỉ khác nhau ở chỗ: Để khởi gán cho các đối tượng thành phần ta dùng tên đối tượng thành phần, còn để khởi gán cho các thuộc tính thừa kế từ các lớp cơ sở ta dùng tên lớp cơ sở:

Tên_đối_tượng_thành_phần(danh sách giá trị)

Tên_lớp_cơ_sở(danh sách giá trị)

Danh sách giá trị lấy từ các đối của hàm tạo của lớp dẫn xuất đang xây dựng

(xem ví dụ mục 2.4 và §6, ví dụ 1)

2.3. Hàm huỷ

Khi một đối tượng của lớp dẫn xuất được giải phóng (bị huỷ), thì các đối tượng thành phần và các đối tượng thừa kế từ các lớp cơ sở cũng bị giải phóng theo. Do đó các hàm huỷ tương ứng sẽ được gọi đến.

Như vậy khi xây dựng hàm hủy của lớp dẫn xuất, chúng ta chỉ cần quan tâm đến các thuộc tính (không phải là đối tượng) khai báo thêm trong lớp dẫn xuất mà thôi. Ta không cần để ý đến các đối tượng thành phần và các thuộc tính thừa kế từ các lớp cơ sở. (xem ví dụ mục 2.4 và §6, ví dụ 2)

2.4. Ví dụ xét các lớp

+ Lớp NGUOI gồm:

- Các thuộc tính

```
char *ht ; // Họ tên
```

```
int ns ;
```

- Hai hàm tạo, phương thức in() và hàm hủy

+ Lớp MON_HOC gồm:

- Các thuộc tính

```
char *monhoc ; // Tên môn học
```

```
int st ; // Số tiết
```

- Hai hàm tạo, phương thức in() và hàm hủy

+ Lớp GIAO_VIEN :

- Kế thừa từ lớp NGUOI

- Đưa thêm các thuộc tính

```
char *bomon ; // Bộ môn công tác
```

```
MON_HOC mh ; // Môn học đang dạy
```

- Hai hàm tạo, phương thức in() và hàm hủy

Hãy để ý cách xây dựng các hàm tạo, hàm hủy của lớp dẫn xuất GIAO_VIEN. Trong lớp GIAO_VIEN có thể gọi tới 2 phương thức in():

```
GIAO_VIEN::in() // Được xây dựng trong lớp GIAO_VIEN
```

```
NGUOI::in() // Thừa kế từ lớp NGUOI
```

Hãy chú ý cách gọi tới 2 phương thức in() trong chương trình dưới đây.

```
//CT5-03
```

```
// Hàm tạo của lớp dẫn xuất
```

```
#include <conio.h>
```

```
#include <iostream.h>
```

```
#include <string.h>
```

```
class MON_HOC
```

```
{
```

```
private:
```

```
char *monhoc;
```

```
int st;
```

```
public:
```

```

MON_HOC()
{
    monhoc=NULL;
    st=0;
}
MON_HOC(char *monhoc1, int st1)
{
    int n = strlen(monhoc1);
    monhoc = new char[n+1];
    strcpy(monhoc,monhoc1);
    st=st1;
}
~ MON_HOC()
{
    if (monhoc!=NULL)
    {
        delete monhoc;
        st=0;
    }
}
void in()
{
    cout << "\nTen mon: " << monhoc << " so tiet: " << st;
}
};

class NGUOI
{
private:
    char *ht;
    int ns;
public:
    NGUOI()
    {
        ht=NULL;
        ns=0;
    }
    NGUOI(char *ht1, int ns1)
    {
        int n = strlen(ht1);
        ht = new char[n+1];
        strcpy(ht,ht1);
        ns=ns1;
    }
    ~NGUOI()
    {
        if (ht!=NULL)
        {
            delete ht;
            ns=0;
        }
    }
}

```

```

        void in()
        {
            cout << "\nHo ten : " << ht << " nam sinh: " << ns;
        }
    };
class GIAO_VIEN : public NGUOI
{
    private:
        char *bomon;
        MON_HOC mh;
    public:
        GIAO_VIEN():mh(),NGUOI() //Su dung ham tao khong doi
        {
            bomon=NULL;
        }
        GIAO_VIEN(char *ht1, int ns1, char *monhoc1,int st1,
                    char *bomon1 ):
        NGUOI(ht1,ns1),mh(monhoc1, st1)
        {
            int n = strlen(bomon1);
            bomon = new char[n+1];
            strcpy(bomon,bomon1);
        }
        ~GIAO_VIEN()
        {
            if (bomon!=NULL)
                delete bomon;
        }
        void in()
        {
            // Su dung phuong thuc in
            NGUOI::in();
            cout << "\n Cong tac tai bo mon: " << bomon;
            mh.in();
        }
};

void main()
{
    clrscr();
    GIAO_VIEN g1; // Goi toi cac ham tao khong doi
    GIAO_VIEN *g2;
    //Goi toi cac ham tao co doi
    g2 = new GIAO_VIEN("PHAM VAN AT", 1945, "CNPM",
                        60, "TIN HOC");

    g2->in();
    /*
    co the viet
    g2->GIAO_VIEN::in();
    */
    g2->NGUOI::in();
    getch();
}

```

```

delete g2; // Goi toi cac ham huy
getch();
}

```

53. PHẠM VI TRUY NHẬP ĐẾN CÁC THÀNH PHẦN CỦA LỚP CƠ SỞ

3.1. Các từ khoá quy định phạm vi truy nhập của lớp cơ sở

+ Mặc dù lớp dẫn xuất được thừa kế tất cả các thành phần của lớp cơ sở, nhưng trong lớp dẫn xuất không thể truy nhập tới tất cả các thành phần này. Giải pháp thường dùng là sử dụng các phương thức của lớp cơ sở để truy nhập đến các thuộc tính của chính lớp cơ sở đó. Cũng có thể sử dụng các giải pháp khác dưới đây.

+ Các thành phần private của lớp cơ sở không cho phép truy nhập trong lớp dẫn xuất.

+ Các thành phần public của lớp cơ sở có thể truy nhập bất kỳ chỗ nào trong chương trình. Như vậy trong các lớp dẫn xuất có thể truy nhập được tới các thành phần này.

+ Các thành phần khai báo là protected có phạm vi truy nhập rộng hơn so với các thành phần private, nhưng hẹp hơn so với các thành phần public. Các thành phần protected của một lớp chỉ được mở rộng phạm vi truy nhập cho các lớp dẫn xuất trực tiếp từ lớp này.

3.2. Hai kiểu dẫn xuất

Có 2 kiểu dẫn xuất là private và public, chúng cho các phạm vi truy nhập khác nhau tới lớp cơ sở. Cụ thể như sau:

+ Các thành phần public và protected của lớp cơ sở sẽ trở thành các thành phần public và protected của lớp dẫn xuất theo kiểu public.

+ Các thành phần public và protected của lớp cơ sở sẽ trở thành các thành phần private của lớp dẫn xuất theo kiểu private.

Ví dụ :

Giả sử lớp A có:

thuộc tính public a1

thuộc tính protected a2

và lớp B dẫn xuất public từ A, thì A::a1 trở thành public trong B, A::a2 trở thành protected trong B.

Do đó nếu dùng B làm lớp cơ sở để xây dựng lớp C. Thì trong C có thể truy nhập tới A::a1 và A::a2.

Thế nhưng nếu sửa đổi để B dẫn xuất private từ A, thì cả A::a1 và A::a2

trở thành private trong B, và khi đó trong C không được phép truy nhập tới các thuộc tính A::a1 và A::a2.

Để biết tường tận hơn, chúng ta hãy biên dịch chương trình:

```
//CT5-04
// Phạm vi truy nhập
#include <conio.h>
#include <iostream.h>
#include <string.h>
class A
{
protected:
    int a1;
public:
    int a2;
    A()
    {
        a1=a2=0;
    }
    A(int t1, int t2)
    {
        a1=t1; a2= t2;
    }
    void in()
    {
        cout << a1 <<" " << a2 ;
    }
};
class B: private A
{
protected:
    int b1;
public:
    int b2;
    B()
    {
        b1=b2=0;
    }
    B(int t1, int t2, int u1, int u2)
    {
        a1=t1; a2=t2; b1=u1;b2=u2;
    }
    void in()
    {
        cout << a1 <<" " << a2 << " " << b1 <<
            " " << b2;
    }
};
class C : public B
{
}
```

```

public:
    C()
    {
        b1=b2=0;
    }
    C(int t1, int t2, int u1,int u2)
    {
        a1=t1; a2=t2; b1=u1;b2=u2;
    }
    void in()
    {
        cout << a1;
        cout <<" " << a2 << " " << b1 << " " << b2;
    }
};

void main()
{
    C c(1,2,3,4);
    c.in();
    getch();
}

```

Chúng ta sẽ nhận được 4 thông báo lỗi sau trong lớp C (tại hàm tạo có đổi và phương thức in):

A::a1 is not accessible

A::a2 is not accessible

A::a1 is not accessible

A::a2 is not accessible

Bây giờ nếu sửa đổi để lớp B dẫn xuất public từ A thì chương trình sẽ không có lỗi và làm việc tốt.

§4. THỪA KẾ NHIỀU MỨC VÀ SỰ TRÙNG TÊN

4.1. Sơ đồ xây dựng các lớp dẫn xuất theo nhiều mức

Như đã biết:

+ Khi đã định nghĩa một lớp (ví dụ lớp A), ta có thể dùng nó làm cơ sở để xây dựng lớp dẫn xuất (ví dụ B).

+ Đến lượt mình, B có thể dùng làm cơ sở để xây dựng lớp dẫn xuất mới (ví dụ C).

+ Tiếp đó lại có thể dùng C làm cơ sở để xây dựng lớp dẫn xuất mới.

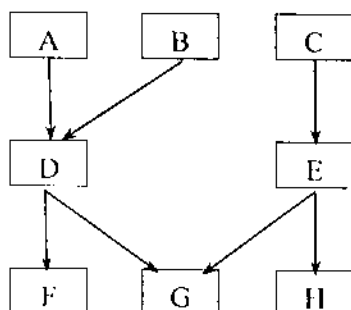
+ Sự tiếp tục theo cách trên là không hạn chế.

Sơ đồ trên chính là sự thừa kế nhiều mức. Ngoài ra chúng ta cũng đã biết:

+ Một lớp có thể được dẫn xuất từ nhiều lớp cơ sở.

+ Một lớp có thể dùng làm cơ sở cho nhiều lớp.

Hình vẽ dưới đây là một ví dụ về sơ đồ thừa kế khá tổng quát, thể hiện được các điều nói trên:



Diễn giải:

Lớp D dẫn xuất từ A và B

Lớp E dẫn xuất từ C

Lớp F dẫn xuất từ D

Lớp G dẫn xuất từ D và E

Lớp H dẫn xuất từ E

4.2. Sự thừa kế nhiều mức.

+ Như đã biết: Lớp dẫn xuất thừa kế tất cả các thành phần (thuộc tính và phương thức) của lớp cơ sở, kể cả các thành phần mà lớp cơ sở được thừa kế.

+ Hãy áp dụng nguyên lý trên để xét lớp G:

- Lớp G thừa kế các thành phần của các lớp D và E
- Lớp D thừa kế các thành phần của lớp A và B
- Lớp E thừa kế các thành phần của lớp C

Như vậy các thành phần có thể sử dụng trong lớp G gồm:

- Các thành phần khai báo trong G (của riêng G)
- Các thành phần khai báo trong các lớp D, E, A, B, C (được thừa kế).

4.3. Sự trùng tên

Như đã nói trong 4.2: Trong lớp G có thể sử dụng (trực tiếp hay gián tiếp) các thành phần của riêng G và các thành phần mà nó được thừa kế từ các lớp D, E, A, B, C. Yêu cầu về cách đặt tên ở đây là:

- + Tên các lớp không được trùng lặp
- + Tên các thành phần trong một lớp cũng không được trùng lặp
- + Tên các thành phần trong các lớp khác nhau có quyền được trùng lặp.

Để phân biệt các thành phần trùng tên trong lớp dẫn xuất, cần sử dụng thêm tên lớp (xem ví dụ trong 4.4).

4.4. Sử dụng các thành phần trong lớp dẫn xuất

Như đã nói ở trên: Thành phần của lớp dẫn xuất gồm:

- + Các thành phần khai báo trong lớp dẫn xuất
- + Các thành phần mà lớp dẫn xuất thừa kế từ các lớp cơ sở

Quy tắc sử dụng các thành phần trong lớp dẫn xuất:

Cách 1: Dùng tên lớp và tên thành phần. Khi đó Chương trình dịch C++ dễ dàng phân biệt thành phần thuộc lớp nào. Ví dụ:

D h; // h là đối tượng của lớp D dẫn xuất từ A và B

h.D::n là thuộc tính n khai báo trong D

h.A::n là thuộc tính n thừa kế từ A (khai báo trong A)

h.D::nhap() là phương thức nhập() định nghĩa trong D

h.A::nhap() là phương thức nhập() định nghĩa trong A

Cách 2: Không dùng tên lớp, chỉ dùng tên thành phần. Khi đó Chương trình dịch C++ phải tự phán đoán để biết thành phần đó thuộc lớp nào. Cách phán đoán như sau: Trước tiên xem thành phần đang xét có trùng tên với một thành phần nào của lớp dẫn xuất không? Nếu trùng thì đó là thành phần của lớp dẫn xuất. Nếu không trùng thì tiếp tục xét các lớp cơ sở theo thứ tự: Các lớp có quan hệ gần với lớp dẫn xuất xét trước, các lớp quan hệ xa xét sau. Hãy chú ý trường hợp sau: Thành phần đang xét có mặt đồng thời trong 2 lớp cơ sở có cùng một đẳng cấp quan hệ với lớp dẫn xuất. Gặp trường hợp này Chương trình dịch C++ không thể quyết định được thành phần này thừa kế từ lớp nào và buộc phải đưa ra một thông báo lỗi (xem ví dụ dưới đây). Cách khắc phục: Trường hợp này phải sử dụng thêm tên lớp như trình bày trong cách 1.

Ví dụ xét lớp dẫn xuất D. Lớp D có 2 cơ sở là các lớp A và B. Giả sử các lớp A, B và D được định nghĩa:

```
class A
{
    private:
        int n;
        float a[20];
    public:
        void nhap();
        void xuat();
};

class B
{
    private:
        int m,n;
```

```

        float    a[20][20];
    public:
        void nhap();
        void xuat();
    } ;
class D : public A, public B
{
    private:
        int k;
    public:
        void nhap();
        void xuat();
    } ;

```

Hãy chú ý các điểm sau:

1. Dùng các phương thức của các lớp A, B để xây dựng các phương thức của D

```

// Xây dựng phương thức nhap()
void D::nhap()
{
    cout << "\n Nhập k : " ;
    cin >> k ;    // k là thuộc tính của D
    A::nhap();    // Nhập các thuộc tính mà D thừa kế từ A
    B::nhap();    // Nhập các thuộc tính mà D thừa kế từ B
}
// Xây dựng phương thức xuat()
void D::xuat()
{
    cout << "\n k = " << k ;
    A::xuat();    // Xuất các thuộc tính mà D thừa kế từ A
    B::xuat();    // Xuất các thuộc tính mà D thừa kế từ B
}

```

2. Làm việc với các đối tượng của lớp dẫn xuất

```

D h ; // Khai báo h là đối tượng của lớp D
h.nhap() ; // tương đương với h.D::nhap();
h.A::xuat(); // In giá trị các thuộc tính h.A::n và h.A::a
h.B::xuat(); // In giá trị các thuộc tính h.B::m, h.B::n và h.B::a
h.D::xuat() ; // In giá trị tất cả các thuộc tính của h
h.xuat() ;    // tương đương với h.D::xuat() ;

```

§5. CÁC LỚP CƠ SỞ ẢO

5.1. Một lớp cơ sở xuất hiện nhiều lần trong lớp dẫn xuất

Một điều hiển nhiên là không thể khai báo 2 lần cùng một lớp trong danh sách của các lớp cơ sở cho một lớp dẫn xuất. Chẳng hạn ví dụ sau là không cho phép:

```
class B : public A, public A
{
} ;
```

Tuy nhiên vẫn có thể có trường hợp cùng một lớp cơ sở được đề cập nhiều hơn một lần trong các lớp cơ sở trung gian của một lớp dẫn xuất. Ví dụ:

```
#include <iostream.h>
class A
{
public:
    int a;
} ;
class B : public A
{
public:
    int b;
} ;
class C : public A
{
public:
    int c;
} ;
class D : public B, public C
{
public:
    int d;
} ;
void main()
{
    D h ;
    h.d = 4 ; // tốt
    h.c = 3 ; // tốt
    h.b = 2 ; // tốt
    h.a = 1 ; // lỗi
}
```

Trong ví dụ này A là cơ sở cho cả 2 lớp cơ sở trực tiếp của D là B và C. Nói cách khác có 2 lớp cơ sở A cho lớp D. Vì vậy trong câu lệnh:

```
h.a = 1 ;
```

thì Chương trình dịch C++ không thể nhận biết được thuộc tính a thừa kế thông qua B hay thông qua C và nó sẽ đưa ra thông báo lỗi sau:

```
Member is ambiguous: 'A::a' and 'A::a'
```

5.2. Các lớp cơ sở ảo

Giải pháp cho vấn đề nói trên là khai báo A như một lớp cơ sở kiểu virtual cho cả B và C. Khi đó B và C được định nghĩa như sau:

```

class B : virtual public A
{
    public:
        int b;
} ;
class C : virtual public A
{
    public:
        int c;
} ;

```

Các lớp cơ sở ảo (virtual) sẽ được kết hợp để tạo một lớp cơ sở duy nhất cho bất kỳ lớp nào dẫn xuất từ chúng. Trong ví dụ trên, hai lớp cơ sở A (A là cơ sở của B và A là cơ sở của C) sẽ kết hợp lại để trở thành một lớp cơ sở A duy nhất cho bất kỳ lớp dẫn xuất nào từ B và C. Như vậy bây giờ D sẽ chỉ có một lớp cơ sở A duy nhất, do đó phép gán:

```
h.a = 1 ;
```

sẽ gán 1 cho thuộc tính a của lớp cơ sở A duy nhất mà D kế thừa.

§6. MỘT SỐ VÍ DỤ VỀ HÀM TẠO, HÀM HUỖ TRONG THỪA KẾ NHIỀU MỨC

Ví dụ 1. Ví dụ này minh họa cách xây dựng hàm tạo trong các lớp dẫn xuất. Ngoài ra còn minh họa cách dùng các phương thức của các lớp cơ sở trong lớp dẫn xuất và cách xử lý các đối tượng thành phần.

Xét 4 lớp A, B, C và D. Lớp C dẫn xuất từ B, lớp D dẫn xuất từ C và có thành phần là đối tượng kiểu A.

```

//CT5-06
// Thừa kế nhiều mức
// Hàm tạo
#include <conio.h>
#include <iostream.h>
#include <string.h>
class A
{
    private:
        int a;
        char *str ;
    public:
        A()
        {
            a=0; str=NULL;
        }
        A(int a1,char *str1)
        {

```

```

        a=a1; str=strdup(str1);
    }
    void xuat()
    {
        cout << "\n" << "So nguyen lop A= " << a
            << " Chuoi lop A: " << str ;
    }
} ;
class B
{
    private:
        int b;
        char *str ;
    public:
        B()
        {
            b=0; str=NULL;
        }
        B(int b1,char *str1)
        {
            b=b1; str=strdup(str1);
        }
        void xuat()
        {
            cout << "\n" << "So nguyen lop B = " << b
                << " Chuoi lop B: " << str ;
        }
} ;
class C : public B
{
    private:
        int c;
        char *str ;
    public:
        C():B()
        {
            c=0; str=NULL;
        }
        C(int b1,char *strb,int c1, char *strc) : B(b1,strb)
        {
            c=c1; str=strdup(strc);
        }
        void xuat()
        {
            B::xuat();
            cout << "\n" << "So nguyen lop C = " << c
                << " Chuoi lop C: " << str ;
        }
} ;
class D : public C
{

```

```

private:
    int d;
    char *str ;
    A    u;
public:
    D():C(),u()
    {
        d=0;
        str=NULL;
    }
    D(int a1, char *stra,int b1,char *strb,int c1,
        char *strc, int d1,
        char *strd) : u(a1,stra), C(b1,strb,c1,strc)
    {
        d=d1;
        str=strdup(strd);
    }
    void xuat()
    {
        u.xuat();
        C::xuat();
        cout << "\n" << "So nguyen lop D = " <<
            d << " Chuoi lop D: " << str ;
    }
} ;
void main()
{
    D h(1,"AA",2,"BB",3,"CC",4,"DD");
    clrscr();
    cout << "\n\n Cac thuoc tinh cua h thua ke B: ";
    h.B::xuat();
    cout << "\n\n Cac thuoc tinh cua h thua ke B va C: ";
    h.C::xuat();
    cout << "\n\n Cac thuoc tinh cua h thua ke B,C
        va khai bao trong D: " ;
    h.xuat();
    getch();
}

```

Ví dụ 2. Ví dụ này minh họa cách xây dựng hàm hủy trong lớp dẫn xuất. Chương trình trong ví dụ này lấy từ chương trình của ví dụ 1, sau đó đưa thêm vào các hàm hủy.

```

//CT5-07
// Thua ke nhieu muc
// Ham tao
// Ham huy
#include <conio.h>
#include <iostream.h>
#include <string.h>

```

```

class A
{
private:
    int a;
    char *str ;
public:
    A()
    {
        a=0; str=NULL;
    }
    A(int a1,char *str1)
    {
        a=a1; str=strdup(str1);
    }
    ~A()
    {
        cout <<"\n Huy A"; getch();
        a=0;
        if (str!=NULL) delete str;
    }
    void xuat()
    {
        cout << "\n" << "So nguyen lop A= " << a
             << " Chuoi lop A: " << str ;
    }
} ;

class B
{
private:
    int b;
    char *str ;
public:
    B()
    {
        b=0; str=NULL;
    }
    B(int b1,char *str1)
    {
        b=b1; str=strdup(str1);
    }
    ~B()
    {
        cout <<"\n Huy B";
        getch();
        b=0;
        if (str!=NULL) delete str;
    }
    void xuat()
    {
        cout << "\n" << "So nguyen lop B = " << b
             << " Chuoi lop B: " << str ;
    }
} ;

```

```

    }
} ;
class C : public B
{
private:
    int c;
    char *str ;
public:
    C():B()
    {
        c=0;
        str=NULL;
    }
    C(int b1,char *strb,int c1, char *strc) : B(b1,strb)
    {
        c=c1;
        str=strdup(strc);
    }
    ~C()
    {
        cout <<"\n Huy C"; getch();
        c=0;
        if (str!=NULL) delete str;
    }
    void xuat()
    {
        B::xuat();
        cout << "\n" << "So nguyen lop C = " << c
            << " Chuoi lop C: " << str ;
    }
} ;
class D : public C
{
private:
    int d;
    char *str ;
    A u;
public:
    D():C(),u()
    {
        d=0; str=NULL;
    }
    D(int a1, char *stra,int b1,char *strb,int c1,
        char *strc, int d1,
        char *strd) : u(a1,stra), C(b1,strb,c1,strc)
    {
        d=d1; str=strdup(strd);
    }
    ~D()
    {
        cout <<"\n Huy D"; getch();
    }
} ;

```

```

        d=0;
        if (str!=NULL) delete str;
    }
    void xuat()
    {
        u.xuat();
        C::xuat();
        cout << "\n" << "So nguyen lop D = " << d
              << " Chuoi lop D: " << str ;
    }
} ;
void main()
{
    D *h;
    h = new D(1, "AA", 2, "BB", 3, "CC", 4, "DD");
    clrscr();
    cout << "\n\n Cac thuoc tinh cua h thua ke B: " ;
    h->B::xuat();
    cout << "\n\n Cac thuoc tinh cua h thua ke B va C: ," ;
    h->C::xuat();
    cout << "\n\n Cac thuoc tinh cua h thua ke B,C
           va khai bao trong D:" ;
    h->xuat();
    delete h; // Lan luot gọi tới các hàm hủy
              // của các lớp D, A, C, B
    getch();
}

```

§7. TOÁN TỬ GÁN CỦA LỚP DẪN XUẤT

7.1. Khi nào cần xây dựng toán tử gán

Khi lớp dẫn xuất có các thuộc tính (kể cả thuộc tính thừa kế từ các lớp cơ sở) là con trỏ, thì nhất thiết không được dùng toán tử gán mặc định, mà phải xây dựng cho lớp dẫn xuất một toán tử gán.

7.2. Cách xây dựng toán tử gán cho lớp dẫn xuất

+ Trước hết cần xây dựng toán tử gán cho các lớp cơ sở và các lớp thành phần.

+ Khi xây dựng toán tử gán cho lớp dẫn xuất cần gọi toán tử gán của các lớp cơ sở và toán tử gán của các lớp thành phần.

+ Để sử dụng toán tử gán của lớp cơ sở, cần ép kiểu theo mẫu sau (giả sử A là lớp cơ sở):

```

A(*this) = A::operator=(h); // Gọi tới phép gán
                             // của lớp cơ sở A

```

Các điều nói trên được minh họa trong ví dụ dưới đây.

7.3. Ví dụ

Chương trình dưới đây minh họa cách xây dựng toán tử gán cho lớp D có 2 lớp cơ sở là C và B (C là lớp cơ sở trực tiếp, còn B là cơ sở của C). Ngoài ra D còn có một thuộc tính là đối tượng của lớp A.

```
//CT5-08
// Thua ke nhieu muc
// Toan tu gan
#include <conio.h>
#include <stdio.h>
#include <iostream.h>
#include <string.h>
class A
{
private:
    int a;
    char *str ;
public:
    A()
    {
a=0; str=NULL;
    }
    A& operator=(A& h)
    {
this->a = h.a;
if(this->str!=NULL) delete this->str;
this->str = strdup(h.str);
return h;
    }
    void nhap()
    {
cout << "\nNhap so nguyen lop A: " ; cin >> a ;
if(str!=NULL) delete str;
cout << "\nNhap chuoi lop A: " ;
char tg[30];
fflush(stdin);
gets(tg);
str = strdup(tg);
    }
    void xuat()
    {
cout << "\n" << "So nguyen lop A = " << a
        << " Chuoi lop A: " << str ;
    }
} ;
class B
{
private:
    int b;
    char *str ;
```

```

public:
    B()
    {
        b=0;
        str=NULL;
    }
    B& operator=(B& h)
    {
        this->b = h.b;
        if(this->str!=NULL) delete this->str;
        this->str = strdup(h.str);
        return h;
    }
    void nhap()
    {
        cout << "\nNhap so nguyen lop B: " ; cin >> b ;
        if(str!=NULL) delete str;
        cout << "\nNhap chuoi lop B: " ;
        char tg[30];
        fflush(stdin); gets(tg);
        str = strdup(tg);
    }
    void xuat()
    {
        cout << "\n" << "So nguyen lop B = " << b
              << " Chuoi lop B: " << str ;
    }
} ;

class C : public B
{
private:
    int c;
    char *str ;
public:
    C():B()
    {
        c=0;
        str=NULL;
    }
    C& operator=(C& h)
    {
        // *((B*)this) = B::operator=(h);
        B(*this) = B::operator=(h);
        this->c = h.c;
        if(this->str!=NULL) delete this->str;
        this->str = strdup(h.str);
        return h;
    }
    void nhap()
    {
        B::nhap();
    }

```

```

cout << "\nNhap so nguyen lop C: " ; cin >> c ;
if(str!=NULL) delete str;
cout << "\nNhap chuoi lop C: " ;
char tg[30];
fflush(stdin); gets(tg);
str = strdup(tg);
}
void xuat()
{
B::xuat();
cout << "\n" << "So nguyen lop C = " << c
    << " Chuoi lop C: " << str ;
}
} ;
class D : public C
{
private:
    int d;
    char *str ;
    A    u;
public:
    D():C(),u()
    {
d=0; str=NULL;
    }
    D& operator=(D& h)
    {
// *((C*)this) = C::operator=(h);
C(*this) = C::operator=(h);
this->u = h.u;
this->d = h.d;
if(this->str!=NULL) delete this->str;
this->str = strdup(h.str);
return h;
    }
    void nhap()
    {
u.nhap();
C::nhap();
cout << "\nNhap so nguyen lop D: " ; cin >> d ;
if(str!=NULL) delete str;
cout << "\nNhap chuoi lop D: " ;
char tg[30];
fflush(stdin);
gets(tg);
str = strdup(tg);
    }
    void xuat()
    {
u.xuat();
C::xuat();
}
}

```

```

        cout << "\n" << "So nguyen lop D = " << d
            << " Chuoi lop D: " << str ;
    }
} ;
void main()
{
    D h1,h2,h3;
    clrscr();
    h1.nhap();
    h3=h2=h1;
    cout<<"\n\nH2:";
    h2.xuat();
    cout<<"\n\nH3:";
    h3.xuat();
    h1.nhap();
    cout<<"\n\nH2:";
    h2.xuat();
    cout<<"\n\nH3:";
    h3.xuat();
    cout<<"\n\nH1:";
    h1.xuat();
    getch();
}

```

58. HÀM TẠO SAO CHÉP CỦA LỚP DẪN XUẤT

8.1. Khi nào cần xây dựng hàm tạo sao chép

Khi lớp dẫn xuất có các thuộc tính (kể cả thuộc tính thừa kế từ các lớp cơ sở) là con trỏ, thì nhất thiết không được dùng hàm tạo sao chép mặc định, mà phải xây dựng cho lớp dẫn xuất một hàm tạo sao chép.

8.2. Cách xây dựng hàm tạo sao chép cho lớp dẫn xuất

+ Trước hết cần xây dựng hàm tạo sao chép cho các lớp cơ sở và các lớp thành phần.

+ Khi xây dựng hàm tạo sao chép cho lớp dẫn xuất cần gọi tới hàm tạo sao chép của các lớp cơ sở và hàm tạo sao chép của các lớp thành phần.

Điều này được minh họa trong ví dụ dưới đây.

8.3. Ví dụ

Chương trình dưới đây minh họa cách xây dựng hàm tạo sao chép cho lớp D có 2 lớp cơ sở là C và B (C là lớp cơ sở trực tiếp, còn B là cơ sở của C). Ngoài ra D còn có một thuộc tính là đối tượng của lớp A. Chương trình này dựa trên chương trình trong mục 7.3 với 2 thay đổi:

+ Xây dựng thêm hàm tạo sao chép cho lớp D.

+ Thay đổi một số câu lệnh trong hàm main để sử dụng hàm tạo sao chép.

Để thấy rõ vai trò của hàm tạo sao chép chúng ta hãy so sánh kết quả nhận được trong 2 trường hợp: Có hàm tạo sao chép và bỏ đi hàm này.

```
//CT5-09
// Thua ke nhieu muc
// Ham tao sao chep
#include <conio.h>
#include <stdio.h>
#include <iostream.h>
#include <string.h>
class A
{
private:
    int a;
    char *str ;
public:
    A()
    {
a=0; str=NULL;
    }
    A(A& h)
    {
this->a = h.a;
this->str = strdup(h.str);
    }
    void nhap()
    {
cout << "\nNhap so nguyen lop A: " ; cin >> a ;
if(str!=NULL) delete str;
cout << "\nNhap chuoi lop A: " ;
char tg[30];
fflush(stdin); gets(tg);
str = strdup(tg);
    }
    void xuat()
    {
cout << "\n" << "So nguyen lop A = " << a
    << " Chuoi lop A: " << str ;
    }
} ;
class B
{
private:
    int b;
    char *str ;
public:
    B()
    {
b=0; str=NULL;
    }
}
```

```

    }
    B(B& h)
    {
    this->b = h.b;
    this->str = strdup(h.str);
    }
    void nhap()
    {
    cout << "\nNhap so nguyen lop B: " ; cin >> b ;
    if(str!=NULL) delete str;
    cout << "\nNhap chuoi lop B: " ;
    char tg[30];
    fflush(stdin); gets(tg);
    str = strdup(tg);
    }
    void xuat()
    {
    cout << "\n" << "So nguyen lop B = " << b
        << " Chuoi lop B: " << str ;
    }
} ;
class C : public B
{
private:
    int c;
    char *str ;
public:
    C():B()
    {
    c=0;
    str=NULL;
    }
    C(C& h):B(h)
    {
    this->c = h.c;
    this->str = strdup(h.str);
    }
    void nhap()
    {
    B::nhap();
    cout << "\nNhap so nguyen lop C: " ; cin >> c ;
    if(str!=NULL) delete str;
    cout << "\nNhap chuoi lop C: " ;
    char tg[30];
    fflush(stdin); gets(tg);
    str = strdup(tg);
    }
    void xuat()
    {
    B::xuat();
    cout << "\n" << "So nguyen lop C = " << c
        << " Chuoi lop C: " << str ;
    }
} ;

```

```

    }
} ;
class D : public C
{
private:
    int d;
    char *str ;
    A    u;
public:
    D():C(),u()
    {
        d=0;
        str=NULL;
    }
    D(D& h):C(h),u(h.u)
    {
        this->d = h.d;
        this->str = strdup(h.str);
    }
    void nhap()
    {
        u.nhap();
        C::nhap();
        cout << "\nNhap so nguyen lop D: " ; cin >> d ;
        if(str!=NULL) delete str;
        cout << "\nNhap chuoi lop D: " ;
        char tg[30];
        fflush(stdin); gets(tg);
        str = strdup(tg);
    }
    void xuat()
    {
        u.xuat();
        C::xuat();
        cout << "\n" << "So nguyen lop D : " << d
              << " Chuoi lop D: " << str ;
    }
} ;
void main()
{
    D h1;
    clrscr();
    h1.nhap();
    D h2(h1) ;
    cout<<"\n\nh2:";
    h2.xuat();
    h1.nhap();
    cout<<"\n\nh2:";
    h2.xuat();
    cout<<"\n\nh1:";
    h1.xuat();
    getch();
}

```

§9. PHÁT TRIỂN, HOÀN THIỆN CHƯƠNG TRÌNH

Có thể dùng tính thừa kế để phát triển khả năng của chương trình.

9.1. Ý tưởng của việc phát triển chương trình như sau

Sau khi đã xây dựng được một lớp, ta sẽ phát triển lớp này bằng cách xây một lớp dân xuất trong đó đưa thêm các thuộc tính và phương thức mới. Quá trình trên lại tiếp tục với lớp vừa nhận được. Ta cũng có thể xây dựng các lớp mới có thuộc tính là đối tượng của các lớp cũ. Bằng cách này, sẽ nhận được một dãy các lớp càng ngày càng hoàn thiện và có nhiều khả năng hơn.

9.2. Ví dụ về việc phát triển chương trình

Giả sử cần xây dựng chương trình vẽ một số hình phẳng. Chúng ta có thể phát triển chương trình này như sau:

Đầu tiên định nghĩa lớp DIEM (Điểm) gồm 2 thuộc tính x, y . Từ lớp DIEM xây dựng lớp DUONG_TRON (Đường tròn) bằng cách bổ sung 2 biến nguyên là r (bán kính) và md (màu đường). Từ lớp DUONG_TRON xây dựng lớp HINH_TRON bằng cách thêm vào biến nguyên mt (màu tô). Đi theo một nhánh khác: Xây dựng lớp DOAN_THANG (Đoạn thẳng) gồm 2 đối tượng kiểu DIEM, và lớp TAM_GIAC gồm 3 đối tượng DIEM.

Chương trình dưới đây cho phép vẽ các đường tròn, hình tròn, đoạn thẳng và hình tam giác.

Chương trình còn minh họa cách dùng con trỏ **this** trong lớp dẫn xuất để thực hiện các phương thức của lớp cơ sở. Ngoài ra còn minh họa cách dùng toán tử chỉ số `[]` để nhận các tọa độ x, y từ các đối tượng của lớp DIEM.

```
//CT5-10
// Phát triển chương trình
#include <conio.h>
#include <stdio.h>
#include <iostream.h>
#include <string.h>
#include <graphics.h>
class DIEM
{
private:
    int x,y;
public:
    DIEM()
    {
        x=y=0;
    }
};
```

```

    }
    DIEM(int x1, int y1)
    {
        x=x1; y=y1;
    }
    DIEM(DIEM &d)
    {
        this->x= d.x;
        this->y= d.y;
    }
    int operator[](int i)
    {
        if (i==1) return x;
        else return y;
    }
};

class DUONG_TRON : public DIEM
{
private:
    int r,md;
public:
    DUONG_TRON() : DIEM()
    {
        r=md=0;
    }
    DUONG_TRON(DIEM d, int r1, int md1) : DIEM(d)
    {
        r=r1;
        md=md1;
    }
    void ve()
    {
        setcolor(md);
        circle ( (*this)[1],(*this)[2],r);
    }
    int getmd()
    {
        return md;
    }
};

class HINH_TRON : public DUONG_TRON
{
private:
    int mt;
public:
    HINH_TRON() : DUONG_TRON()
    {
        mt=0;
    }
    HINH_TRON(DIEM d, int r1, int md1, int mt1) :
        DUONG_TRON(d,r1,md1)

```

```

        {
            mt=mt1;
        }
        void ve()
        {
            DUONG_TRON::ve();
            setfillstyle(1,mt);
            floodfill((*this)[1],(*this)[2],this->getmd());
        }
    } ;
class DOAN_THANG
{
    private:
        DIEM d1, d2;
        int md;
    public:
        DOAN_THANG() : d1(), d2()
        {
            md=0;
        }
        DOAN_THANG(DIEM t1, DIEM t2, int mdl)
        {
            d1=t1;
            d2 = t2;
            md=mdl;
        }
        void ve()
        {
            setcolor(md);
            line(d1[1],d1[2],d2[1],d2[2]);
        }
    } ;
class TAM_GIAC
{
    private:
        DIEM d1,d2,d3;
        int md, mt;
    public:
        TAM_GIAC(): d1(), d2(), d3()
        {
            md=mt=0;
        }
        TAM_GIAC(DIEM t1,DIEM t2,DIEM t3,int mdl,int mt1)
        {
            d1=t1; d2=t2;
            d3 = t3;
            md=mdl;
            mt=mt1;
        }
        void ve()
        {

```

```

        DOAN_THANG(d1,d2,md).ve();
        DOAN_THANG(d1,d3,md).ve();
        DOAN_THANG(d2,d3,md).ve();
        setfillstyle(1,mt);
        floodfill((d1[1]+d2[1]+d3[1])/3,(d1[2]+d2[2]+
            d3[2])/3,md);
    }
} ;

void ktdh()
{
    int mh=0,mode=0;
    initgraph(&mh,&mode,"");
}

void main()
{
    ktdh();
    DUONG_TRON dt(DIEM(100,100),80,MAGENTA);
    HINH_TRON ht(DIEM(400,100),80,RED,YELLOW);
    DOAN_THANG t(DIEM(100,100),DIEM(400,100),BLUE);
    TAM_GIAC tg(DIEM(250,150), DIEM(100,400),
        DIEM(400,400), CYAN, CYAN);

    dt.ve();
    ht.ve();
    t.ve();
    tg.ve();
    getch();
    closegraph();
}

```

§10. BỔ SUNG, NÂNG CẤP CHƯƠNG TRÌNH

Có thể dùng tính thừa kế để sửa đổi, bổ sung, nâng cấp chương trình.

10.1. Ý tưởng của việc nâng cấp chương trình như sau: Giả sử đã có một chương trình hoạt động tốt. Bây giờ cần có một số bổ sung thay đổi không nhiều lắm. Có 2 giải pháp: Hoặc sửa chữa các lớp đang hoạt động ổn định, hoặc xây dựng một lớp dẫn xuất để thực hiện các bổ sung, sửa chữa trên lớp này. Rõ ràng giải pháp thứ 2 tỏ ra hợp lý hơn.

10.2. Ví dụ

Giả sử đã có chương trình quản lý giáo viên gồm 3 lớp MON_HOC (Môn học), GV (Giáo viên) và BM (Bộ môn):

```

class MON_HOC
{
private:
    char tenmh[20]; // Tên môn học
    int    soliet; // Số Liệt

```

```

public:
    MON_HOC(); // Hàm tạo
    const MON_HOC& operator=(const MON_HOC& m);
                                   // Gán
    void nhap(); // Nhập
    void xuat(); // Xuất
};
class GV
{
private:
    char ht[25]; // Ho ten
    int ns;      // Nam sinh
    int sm;      // So mon hoc co the day
    MON_HOC *mh; //Danh sach cac mon hoc
public:
    GV(); // Hàm tạo
    ~GV(); //Hàm huỷ
    int getsn(); // Cho biết số môn
                                   // (dùng trong BM::sapxep)
    const GV& operator=(const GV& g); // Gán (dùng trong
                                   // BM::sapxep)
    void nhap(); // Nhập
    void xuat(); // Xuất
};
class BM // Bo mon
{
private:
    char tenbm[20]; // Tên bộ môn
    int n; // So giao vien
    GV *gv; // Danh sach giao vien
public:
    BM() // Hàm tạo
    void nhap(); // Nhập
    void xuat(); // Xuất
    void sapxep(); // Sắp xếp
};

```

Chương trình cho phép:

1. Nhập danh sách giáo viên của bộ môn.
2. Sắp xếp danh sách giáo viên theo thứ tự giảm của số môn mà mỗi giáo viên có thể giảng dạy.
3. In danh sách giáo viên.

Nội dung chương trình như sau:

//CT5-11

// Nang cap chuong trinh, CT ban dau

#include <conio.h>

#include <stdio.h>

#include <iostream.h>

```

#include <string.h>
class MON_HOC
{
    private:
        char tenmh[20];
        int sotiet;
    public:
        MON_HOC()
        {
            tenmh[0]=sotiet=0;
        }
        const MON_HOC& operator=(const MON_HOC& m)
        {
            strcpy(this->tenmh,m.tenmh);
            this->sotiet = m.sotiet;
            return m;
        }
        void nhap()
        {
            cout << "\nTen mon hoc:";
            fflush(stdin);
            gets(tenmh);
            cout << "So tiet: " ;
            cin >> sotiet;
        }
        void xuat()
        {
            cout << "\nTen mon hoc:" << tenmh
                << " so tiet: " << sotiet;
        }
};

class GV
{
    private:
        char ht[25]; // Ho ten
        int ns; // Nam sinh
        int sm; // So mon hoc co the day
        MON_HOC *mh ; //Danh sach cac mon hoc
    public:
        GV()
        {
            ht[0]= ns= sm= 0 ;
            mh = NULL;
        }
        ~GV()
        {
            ht[0]= ns= sm= 0 ;
            if (mh) delete mh;
        }
        int getsm()
        {

```

```

        return sm;
    }
    const GV& operator=(const GV& g);
    void nhap();
    void xuat();
} ;
const GV& GV::operator=(const GV& g)
{
    strcpy(this->ht,g.ht);
    this->ns=g.ns;
    int n = g.sm;
    this->sm = n;
    if (this->mh) delete this->mh;
    if (n)
    {
        this->mh = new MON_HOC[n+1];
        for (int i=1; i<=n; ++i)
            this->mh[i] = g.mh[i];
    }
    return g;
}
void GV::nhap()
{
    cout << "\nHo ten: " ;
    fflush(stdin); gets(ht);
    cout << "Nam sinh: " ;
    cin >> ns;
    cout << "So mon co the giang day: " ;
    cin >> sm;
    if (this->mh) delete this->mh;
    if(sm)
    {
        this->mh = new MON_HOC[sm+1];
        for (int i=1; i<=sm; ++i)
            this->mh[i].nhap();
    }
}
void GV::xuat()
{
    cout << "\nHo ten: " << ht ;
    cout << "\nNam sinh: " << ns ;
    cout << "\nSo mon co the giang day: " << sm;
    if (sm)
    {
        cout << "\n Do la: ";
        for (int i=1; i<=sm; ++i)
            this->mh[i].xuat();
    }
}
class BM // Bo mon
{

```

```

private:
    char tenbm[20];
    int n; // So giao vien
    GV *gv; // Danh sach giao vien
public:
    BM()
    {
        tenbm[0] = n = 0;
        gv = NULL;
    }
    void nhap();
    void xuat();
    void sapxep();
} ;
void BM::nhap()
{
    cout << "\n\nTen bo mon: " ;
    fflush(stdin); gets(tenbm);
    cout << "So giao vien: ";
    cin >> n;
    if (gv) delete gv;
    if (n)
    {
        gv = new GV[n+1];
        for (int i=1; i<=n; ++i)
            gv[i].nhap();
    }
}
void BM::xuat()
{
    cout << "\nBo mon: " << tenbm;
    cout << "\nSo giao vien: " << n;
    if (n)
    {
        cout << "\n Danh sach giao vien cua bo mon:";
        for (int i=1; i<=n; ++i)
            gv[i].xuat();
    }
}
void BM::sapxep()
{
    GV tg;
    int i, j;
    if (n)
        for (i=1; i<n; ++i)
            for (j=i+1; j<=n; ++j)
                if (gv[i].getsm()<gv[j].getsm())
                {
                    tg=gv[i]; gv[i]=gv[j]; gv[j]=tg;
                }
}

```

```

void main()
{
    BM b;
    b.nhap();
    b.sapxep();
    b.xuat();
    getch();
}

```

Vấn đề đặt ra là: Hiện nay các giáo viên đã bắt đầu hướng dẫn luận văn tốt nghiệp cho sinh viên. Vì vậy cần bổ sung thông tin này vào phần dữ liệu giáo viên. Để nâng cấp chương trình chúng ta sẽ định nghĩa lớp GV2 dẫn xuất từ lớp GV, sau đó trong lớp BM sẽ thay GV bằng GV2. Có 2 chỗ cần bổ sung và một chỗ cần sửa đổi như sau:

1. Bổ sung trong lớp GV phương thức:

```

GV*  getGV()
{
    return this;
}

```

Phương thức này dùng để xây dựng toán tử gán cho lớp GV2.

2. Trong lớp BM thay GV bằng GV2. Điều này có thể đạt được bằng sửa chữa trực tiếp hoặc bằng một câu lệnh #define :

```
#define  GV  GV2
```

3. Định nghĩa thêm 2 lớp: LV (Luận văn) và GV2 (Lớp GV2 dẫn xuất từ lớp GV) như sau:

```

class LV  // Luan van
{
    private:
        char tenlv[30]; // Ten luan van
        char tensv[25]; // Ten sinh vien
        int nambv; // Nam bao ve luan van
    public:
        LV(); // Hàm tạo
        const LV& operator=(const LV& l) ; // Gán
        void nhap(); // Nhập
        void xuat() ;
} ;
class GV2 : public GV
{
    private:
        int solv; // Số luận văn đã hướng dẫn
        LV *lv; // Danh sách luận văn
    public:
        GV2(); // Hàm tạo
        ~GV2(); // Hàm hủy
}

```

```

        GV2& operator=(GV2& q); // Gán
        void nhap(); // Nhập
        void xuat(); // Xuất
    } ;
Chương trình nâng cấp như sau:
//CT5-12B
// Nâng cấp chương trình
// CT nâng cấp
#include <conio.h>
#include <stdio.h>
#include <iostream.h>
#include <string.h>
class MON_HOC
{
private:
    char tenmh[20];
    int sotiet;
public:
    MON_HOC()
    {
        tenmh[0]=sotiet=0;
    }
    const MON_HOC& operator=(const MON_HOC& m)
    {
        strcpy(this->tenmh,m.tenmh);
        this->sotiet = m.sotiet;
        return m;
    }
    void nhap()
    {
        cout << "\nTen mon hoc:";
        fflush(stdin);
        gets(tenmh);
        cout << "So tiet: " ;
        cin >> sotiet;
    }
    void xuat()
    {
        cout << "\nTen mon hoc:" << tenmh
              << " so tiet: " << sotiet;
    }
} ;
// Bỏ sung phương thức getGV cho lớp GV
// dùng để xây dựng toán tử gán cho lớp GV2
class GV
{
private:
    char ht[25]; // Họ tên
    int ns;      // Năm sinh
    int sm;      // Số môn học có thể dạy

```

```

    MON_HOC *mh ; //Danh sach cac mon hoc
public:
    GV()
    {
        ht[0]= ns= sm= 0 ;
        mh = NULL;
    }
    ~GV()
    {
        ht[0]= ns= sm= 0 ;
        if (mh) delete mh;
    }
    // Bo sung phuong thuc getGV
    GV* getGV()
    {
        return this;
    }
    int getsm()
    {
        return sm;
    }
    const GV& operator=(const GV& g);
    void nhap();
    void xuat();
} ;
const GV& GV::operator=(const GV& g)
{
    strcpy(this->ht,g.ht);
    this->ns=g.ns;
    int n = g.sm;
    this->sm = n;
    if (this->mh) delete this->mh;
    if (n)
    {
        this->mh = new MON_HOC[n+1];
        for (int i=1; i<=n; ++i)
            this->mh[i] = g.mh[i];
    }
    return g;
}
void GV::nhap()
{
    cout << "\nHo ten: " ;
    fflush(stdin); gets(ht);
    cout << "Nam sinh: " ;
    cin >> ns;
    cout << "So mon co the giang day: " ;
    cin >> sm;
    if (this->mh) delete this->mh;
    if (sm)
    {

```

```

        this->mh = new MON_HOC[sm+1];
        for (int i=1; i<=sm; ++i)
            this->mh[i].nhap();
    }
}

void GV::xuat()
{
    cout << "\nHo ten: " << ht ;
    cout << "\nNam sinh: " << ns ;
    cout << "\nSo mon co the giang day: " << sm;
    if (sm)
    {
        cout << "\n Do la: ";
        for (int i=1; i<=sm; ++i)
            this->mh[i].xuat();
    }
}

// Bo sung cac lop LV va GV2
class LV // Luan van
{
private:
    char tenlv[30]; // Ten luan van
    char tensv[25]; // Ten sinh vien
    int nambv;      // Nam bao ve luan van
public:
    LV()
    {
        tenlv[0]=tensv[0] = nambv = 0;
    }
    const LV& operator=(const LV& l)
    {
        strcpy(this->tenlv,l.tenlv);
        strcpy(this->tensv,l.tensv);
        this->nambv = l.nambv ;
        return l;
    }
    void nhap()
    {
        cout << "\nTen luan van:";
        fflush(stdin); gets(tenlv);
        cout << "Ten sinh vien:";
        fflush(stdin); gets(tensv);
        cout << "Nam bao ve: " ;
        cin >> nambv ;
    }
    void xuat()
    {
        cout << "\nTen lan van:" << tenlv
        << "      Sinh vien: " << tensv
        << "      Nam bao ve: " << nambv;
    }
}

```

```

    } ;
class GV2 : public GV
{
    private:
        int solv;
        LV *lv;
    public:
        GV2():GV()
        {
            solv = 0 ;
            lv = NULL;
        }
        ~GV2()
        {
            if (solv) delete lv;
        }
        GV2& operator=(GV2& g);
        void nhap();
        void xuat();
    } ;
GV2& GV2::operator=(GV2& g)
{
    GV *g1, *g2;
    g1 = this->getGV();
    g2 = g.getGV();
    *g1 = *g2;
    int n = g.solv;
    this->solv = n;
    if (this->lv) delete this->lv;
    if (n)
    {
        this->lv = new LV[n+1];
        for (int i=1; i<=n; ++i)
            this->lv[i] = g.lv[i];
    }
    return g;
}
void GV2::nhap()
{
    GV::nhap();
    cout << "So luan van da huong dan: " ;
    cin >> solv;
    if (this->lv) delete this->lv;
    if (solv)
    {
        this->lv = new LV[solv+1];
        for (int i=1; i<=solv; ++i)
            this->lv[i].nhap();
    }
}
void GV2::xuat()

```

```

{
    GV::xuat();
    cout << "\nSo luan van da huong dan: " << solv;
    if (solv)
    {
        cout << "\n Do la: ";
        for (int i=1; i<=solv; ++i)
            this->lv[i].xuat();
    }
}
// Sua lop BM: thay GV bang GV2
#define GV GV2
class BM    // Bo mon
{
    private:
        char tenbm[20];
        int  n;           // So giao vien
        GV  *gv;         // Danh sach giao vien
    public:
        BM()
        {
            tenbm[0] = n = 0;
            gv = NULL;
        }
        void nhap();
        void xuat();
        void sapxep();
    } ;
void BM::nhap()
{
    cout << "\n\nTen bo mon: " ;
    fflush(stdin); gets(tenbm);
    cout << "So giao vien: ";
    cin >> n;
    if (gv) delete gv;
    if (n)
    {
        gv = new GV[n+1];
        for (int i=1; i<=n; ++i)
            gv[i].nhap();
    }
}
void BM::xuat()
{
    cout << "\nBo mon: " << tenbm;
    cout << "\nSo giao vien: " << n;
    if (n)
    {
        cout << "\n Danh sach giao vien cua bo mon:";
        for (int i=1; i<=n; ++i)
            gv[i].xuat();
    }
}

```

```

    }
}
void BM::sapxep()
{
    GV tg;
    int i, j;
    if (n)
        for (i=1; i<n; ++i)
            for (j=i+1; j<=n; ++j)
                if (gv[i].getsm()<gv[j].getsm())
                {
                    tg=gv[i]; gv[i]=gv[j]; gv[j]=tg;
                }
}
#undef GV
void main()
{
    BM b;
    b.nhap();
    b.sapxep();
    b.xuat();
    getch();
}

```

§11. TỪ KHÁI QUÁT ĐẾN CỤ THỂ

Tính thừa kế cũng thường dùng để thiết kế các bài toán theo hướng từ khái quát đến cụ thể, từ chung đến riêng. Đầu tiên đưa ra các lớp để mô tả những đối tượng chung, sau đó dẫn xuất tới các đối tượng ngày một cụ thể hơn.

Một trường hợp khác cũng thường gặp là: Quản lý nhiều thực thể có những phần dữ liệu chung. Khi đó ta có thể xây dựng một lớp cơ sở gồm các phần dữ liệu chung. Mỗi thực thể sẽ được mô tả bằng một lớp dẫn xuất từ lớp cơ sở này.

Sau đây là một số ví dụ minh họa:

Ví dụ 1 (minh họa tư tưởng đi từ khái quát đến cụ thể) : Giả sử cần quản lý sinh viên của một trường đại học. Khi đó ta có thể bắt đầu từ lớp SINH_VIEN (Sinh viên). Sau đó dùng nó làm cơ sở để dẫn xuất tới các lớp mô tả các đối tượng sinh viên cụ thể hơn, ví dụ: SV Tin, SV Toán, SV Luật, SV Du lịch, ...

Các bài toán kiểu như vậy rất thường gặp trong thực tế.

Ví dụ 2 (minh họa phần chung của nhiều thực thể). Giả sử cần xây dựng phần mềm để thực hiện các phép tính về ma trận vuông và véc tơ cấp n. Ta có

nhận xét là n chung cho cả véc tơ và ma trận. Hơn nữa nó còn chung cho tất cả các ma trận và véc tơ cùng xét trong bài toán. Vì vậy có thể định nghĩa một lớp cơ sở chỉ có một thuộc tính tĩnh (static) n. Các lớp ma trận, véc tơ dẫn xuất từ lớp này và sử dụng chung cùng một giá trị n.

Dưới đây là chương trình thực hiện các phép toán ma trận, véc tơ. Chương trình được tổ chức thành 3 lớp:

Lớp CAP (Cấp ma trận, véc tơ) gồm một thành phần tĩnh n và phương thức nhập n.

Lớp VT (Véc tơ) có một thuộc tính là mảng một chiều (chứa các phần tử của véc tơ) và các phương thức nhập, xuất.

Lớp MT (Ma trận) có một thuộc tính là mảng 2 chiều (chứa các phần tử của ma trận), các phương thức nhập, xuất và nhân. Lớp MT là bạn của lớp VT.

Chương trình sẽ nhập một ma trận, nhập một véc tơ và tính tích của chúng.

```
//CT5-13
// ma tran vec to
// Dùng thuộc tính static
#include <conio.h>
#include <stdio.h>
#include <iostream.h>
#include <ctype.h>
class CAP;
class MT;
class VT;
class CAP
{
private:
    static int n;
public:
    void nhap()
    {
        int ch;
        if (n==0)
        {
            cout << "\nN= "; cin >> n;
        }
        else
        {
            cout << "\n Hien n = " << n;
            cout << "\n Co thay doi n? - C/K";
            ch=toupper(getch());
            if (ch=='C')
            {
                cout << "\nN= "; cin >> n;
            }
        }
    }
};
```

```

        }
    }
    int getN()
    {
        return n;
    }
};
int CAP::n=0;
class MT : public CAP
{
    private:
        double a[20][20];
    public:
        void nhap();
        void xuat();
        VT operator*(VT x);
};
class VT : public CAP
{
    private:
        double x[20];
    public:
        friend class MT;
        void nhap();
        void xuat();
};
void MT::nhap()
{
    int n,i,j;
    n = this->getN();
    if (n==0)
    {
        this->CAP::nhap();
        n = this->getN();
    }
    for (i=1; i<=n; ++i)
        for (j=1; j<=n; ++j)
        {
            cout << " PT hang " << i << " cot " << j << " = ";
            cin >> a[i][j];
        }
}
void MT::xuat()
{
    int n,i,j;
    n = this->getN();
    if (n)
        for (int i=1; i<=n; ++i)
        {
            cout << "\n" ;
            for (int j=1; j<=n; ++j)

```

```

        cout << a[i][j] << " ";
    }
}
VT MT::operator*(VT x)
{
    VT y;
    int n,i,j;
    n = this->getN();
    for (i=1; i<=n; ++i)
    {
        y.x[i]=0;
        for (j=1; j<=n; ++j)
            y.x[i] += a[i][j]*x.x[j];
    }
    return y;
}
void VT::nhap()
{
    int n,i;
    n = this->getN();
    if (n==0)
    {
        this->CAP::nhap();
        n = this->getN();
    }
    for (i=1; i<=n; ++i)
    {
        cout << " PT thu " << i << " = ";
        cin >> x[i];
    }
}
void VT::xuat()
{
    int n,i;
    n = this->getN();
    if (n)
    {
        cout << "\n";
        for (int i=1; i<=n; ++i)
        {
            cout << x[i] << " ";
        }
    }
}
void main()
{
    MT a;
    VT x,y;
    clrscr();
    cout<<"\nNhập ma tran A:";
    a.nhap();

```

```

cout<<"\n\nNhap Vec lo x:\n";
x.nhap();
y = a*x;
cout<<"\n\nMa tran A";
a.xuat();
cout<<"\n\nVec lo X";
x.xuat();
cout<<"\n\nVec to Y=AX";
y.xuat();
getch();
}

```

§12. TOÀN THỂ VÀ BỘ PHẬN

Thông thường khi xem xét, giải quyết một bài toán, ta thường chia nó thành các bài toán nhỏ hơn. Nói cách khác: Một bài toán lớn bao gồm nhiều bài toán bộ phận. Khi đó ta có thể định nghĩa các lớp cho các bài toán bộ phận. Lớp cho bài toán chung được dẫn xuất từ các lớp nói trên.

Xét một thí dụ đơn giản là bài toán quản lý thư viện. Nó gồm 2 bài toán bộ phận là quản lý sách và quản lý đọc giả. Chúng ta sẽ xây dựng lớp SACH và lớp DOC GIA. Sau đó dùng các lớp này làm cơ sở để xây dựng lớp THU VIEN.

TƯƠNG ỨNG BỘI VÀ PHƯƠNG THỨC ẢO

Tương ứng bội và phương thức ảo là công cụ mạnh của C++ cho phép tổ chức quản lý các đối tượng khác nhau theo cùng một lược đồ. Một khái niệm khác liên quan là: lớp cơ sở trừu tượng. Chương này sẽ trình bày cách sử dụng các công cụ trên để xây dựng chương trình quản lý nhiều đối tượng khác nhau theo một lược đồ thống nhất.

§ 1. PHƯƠNG THỨC TĨNH

1.1. Lời gọi tới phương thức tĩnh

Như đã biết một lớp dẫn xuất được thừa kế các phương thức của các lớp cơ sở tiền bối của nó. Ví dụ lớp A là cơ sở của B, lớp B lại là cơ sở của C, thì C có 2 lớp cơ sở tiền bối là B và A. Lớp C được thừa kế các phương thức của A và B. Các phương thức mà chúng ta vẫn nói là các phương thức tĩnh. Để tìm hiểu thêm về cách gọi tới các phương thức tĩnh, ta xét ví dụ về các lớp A, B và C như sau:

```
class A
{
    public:
        void xuat()
        {
            cout << "\n Lop A " ;
        }
};
class B:public A
{
    public:
        void xuat()
        {
            cout << "\n Lop B " ;
        }
};
class C: public B
{
    public:
        void xuat()
        {
```

```

        cout << "\n Lop C " ;
    }
};

```

Lớp C có 2 lớp cơ sở tiền bối là A, B và C kế thừa các phương thức của A và B. Do đó một đối tượng của C sẽ có tới 3 phương thức xuất. Hãy theo dõi các câu lệnh sau:

```

C h;          // h là đối tượng kiểu C
h.xuat();     // Gọi tới phương thức h.D::xuat()
h.B::xuat();  // Gọi tới phương thức h.B::xuat()
h.A::xuat();  // Gọi tới phương thức h.A::xuat()

```

Các lời gọi phương thức trong ví dụ trên đều xuất phát từ đối tượng h và mọi lời gọi đều xác định rõ phương thức cần gọi.

Bây giờ chúng ta hãy xét các lời gọi không phải từ một biến đối tượng mà từ một con trỏ. Xét các câu lệnh:

```

A *p, *q, *r; // p, q, r là con trỏ kiểu A
A a;          // a là đối tượng kiểu A
B b;          // b là đối tượng kiểu B
C c;          // c là đối tượng kiểu C

```

Chúng ta hãy ghi nhớ mệnh đề sau về con trỏ của các lớp dẫn xuất và cơ sở:

Phép gán con trỏ: Con trỏ của lớp cơ sở có thể dùng để chứa địa chỉ các đối tượng của lớp dẫn xuất.

Như vậy cả 3 phép gán sau đều hợp lệ:

```

p = &a ;
q = &b ;
r = &c ;

```

Chúng ta tiếp tục xét các lời gọi phương thức từ các con trỏ p, q, r:

```

p->xuat();
q->xuat();
r->xuat();

```

và hãy lý giải xem phương thức nào (trong các phương thức A::xuat, B::xuat và C::xuat) được gọi. Câu trả lời như sau:

Cả 3 câu lệnh trên đều gọi tới phương thức A::xuat(), vì các con trỏ p, q và r đều có kiểu A.

Như vậy có thể tóm lược cách thức gọi các phương thức tĩnh như sau:

Quy tắc gọi phương thức tĩnh: Lời gọi tới phương thức tĩnh bao giờ cũng xác định rõ phương thức nào (trong số các phương thức trùng tên của các lớp có quan hệ thừa kế) được gọi:

1. Nếu lời gọi xuất phát từ một đối tượng của lớp nào, thì phương thức của lớp đó sẽ được gọi.

2. Nếu lời gọi xuất phát từ một con trỏ kiểu lớp nào, thì phương thức của lớp đó sẽ được gọi bất kể con trỏ chứa địa chỉ của đối tượng nào.

1.2. Ví dụ

Xét 4 lớp A, B, C và D. Lớp B và C có chung lớp cơ sở A. Lớp D dẫn xuất từ C. Cả 4 lớp đều có phương thức xuất(). Xét hàm:

```
void hien(A *p)
{
    p->xuat();
}
```

Không cần biết tới địa chỉ của đối tượng nào sẽ truyền cho đối con trỏ p, lời gọi trong hàm luôn luôn gọi tới phương thức A::xuat() vì con trỏ p kiểu A. Như vậy bốn câu lệnh:

```
hien(&a);
hien(&b);
hien(&c);
hien(&d);
```

trong hàm main (của chương trình dưới đây) đều gọi tới A::xuat().

```
//CT6-01
// Phương thức tính
#include <conio.h>
#include <stdio.h>
#include <iostream.h>
#include <ctype.h>
class A
{
private:
    int n;
public:
    A()
    {
        n=0;
    }
    A(int nl)
    {
        n=nl;
    }
    void xuat()
    {
        cout << "\nLop A: " << n;
    }
    int getN()
    {
        return n;
    }
};
```

```

};
class B:public A
{
    public:
        B():A()
        {
        }
        B(int n1):A(n1)
        {
        }
        void xuat()
        {
            cout << "\nLop B: "<<getN();
        }
};
class C: public A
{
    public:
        C():A()
        {
        }
        C(int n1):A(n1)
        {
        }
        void xuat()
        {
            cout << "\nLop C: "<<getN();
        }
};
class D:public C
{
    public:
        D():C()
        {
        }
        D(int n1):C(n1)
        {
        }
        void xuat()
        {
            cout << "\nLop D: "<<getN();
        }
};
void hien(A *p)
{
    p->xuat();
}
void main()
{
    A a(1);
    B b(2);

```

```

C c(3);
D d(4);
clrscr();
hien(&a);
hien(&b);
hien(&c);
hien(&d);
getch();
}

```

§ 2. SỰ HẠN CHẾ CỦA PHƯƠNG THỨC TÍNH

Ví dụ sau cho thấy sự hạn chế của phương thức tính trong việc sử dụng tính thừa kế để phát triển chương trình.

Giả sử cần xây dựng chương trình quản lý thí sinh. Mỗi thí sinh đưa vào ba thuộc tính: Họ tên, số báo danh và tổng điểm. Chương trình gồm ba chức năng: Nhập dữ liệu thí sinh, in dữ liệu thí sinh ra máy in và xem - in (in họ tên ra màn hình, sau đó lựa chọn hoặc in hoặc không). Chương trình dưới đây sử dụng lớp TS (Thí sinh) đáp ứng được yêu cầu đặt ra.

```

//CT6-02
// Hạn chế phương thức tính
// Lớp TS
#include <conio.h>
#include <stdio.h>
#include <iostream.h>
#include <ctype.h>
class TS
{
private:
    char ht[25];
    int sobd;
    float td;
public:
    void nhap()
    {
        cout << "\nHọ tên: ";
        fflush(stdin); gets(ht);
        cout << "Số báo danh: ";
        cin >> sobd;
        cout << "Tổng điểm: ";
        cin >> td;
    }
    void in()
    {
        fprintf(stdout, "\nHọ tên: %s", ht);
        fprintf(stdout, "\nSố báo danh: %d", sobd);
        fprintf(stdout, "\nTổng điểm: %0.1f", td);
    }
};

```

```

    }
    void xem_in()
    {
        int ch;
        cout << "\nIlo ten: " << ht ;
        cout << "\nCo in khong? - C/K" ;
        ch = toupper(getch());
        if (ch=='C')
            this->in();
    }
} ;

void main()
{
    TS t[100];
    int i, n;
    cout << "\nSo thi sinh: ";
    cin >> n;
    for (i=1; i<=n; ++i)
        t[i].nhap();
    for (i=1; i<=n; ++i)
        t[i].xem_in();
    getch();
}

```

Giả sử Nhà trường muốn quản lý thêm địa chỉ của thí sinh. Vì sự thay đổi ở đây là không nhiều, nên chúng ta không dă động đến lớp TS mà xây dựng lớp mới TS2 dẫn xuất từ lớp TS. Trong lớp TS2 đưa thêm thuộc tính dc (địa chỉ) và các phương thức nhập, in. Cụ thể lớp TS2 được định nghĩa như sau:

```

class TS2:public TS
{
    private:
        char dc[30] ; // Địa chỉ
    public:
        void nhap()
        {
            TS::nhap();
            cout << "Địa chỉ: " ;
            fflush(stdin); gets(dc);
        }
        void in()
        {
            TS::in();
            fprintf(stdout, "\nĐịa chỉ: %s", dc);
        }
};

```

Trong lớp TS2 không xây dựng lại phương thức xem_in, mà sẽ dùng phương thức xem_in của lớp TS. Chương trình mới như sau:

```

//CT6-03
// Han che phuong thuc tinh
// Lop TS TS2
#include <conio.h>
#include <stdio.h>
#include <iostream.h>
#include <ctype.h>
class TS
{
private:
    char ht[25];
    int sobd;
    float td;
public:
    void nhap()
    {
        cout << "\nHo ten: " ;
        fflush(stdin); gets(ht);
        cout << "So bao danh: " ;
        cin >> sobd;
        cout << "Tong diem: " ;
        cin >> td;
    }
    void in()
    {
        fprintf(stdout, "\n\nHo ten: %s", ht);
        fprintf(stdout, "\nSo bao danh: %d", sobd);
        fprintf(stdout, "\nTong diem: %0.1f", td);
    }
    void xem_in()
    {
        int ch;
        cout << "\nHo ten: " << ht ;
        cout << "\nCo in khong? - C/K" ;
        ch = toupper(getch());
        if (ch-->'C')
            this->in(); //Goi den TS::in() (Vi this
                        // la con tro kieu TS)
    }
};
class TS2: public TS
{
private:
    char dc[30] ; // Dia chi
public:
    void nhap()
    {
        TS::nhap();
        cout << "Dia chi: " ;
        fflush(stdin); gets(dc);
    }
};

```

```

    }
    void in()
    {
        TS::in();
        fprintf(stderr, "\nDia chi: %s", dc);
    }
};

void main()
{
    TS2 t[100];
    int i, n;
    cout << "\nSo thi sinh: ";
    cin >> n;
    for (i=1; i<=n; ++i)
        t[i].nhap();
    for (i=1; i<=n; ++i)
        t[i].xem_in();
    getch();
}

```

Khi thực hiện chương trình này, chúng ta nhận thấy: Dữ liệu in ra vẫn không có địa chỉ. Điều này có thể giải thích như sau:

Xét câu lệnh (thứ 2 từ dưới lên trong hàm main):

```
t[i].xem_in() ;
```

Câu lệnh này gọi tới phương thức xem_in của lớp TS2 (vì t[i] là đối tượng của lớp TS2). Nhưng lớp TS2 không định nghĩa phương thức xem_in, nên phương thức TS::xem_in() sẽ được gọi tới. Hãy theo dõi phương thức này:

```

void xem_in()
{
    int ch;
    cout << "\nHọ tên: " << ht ;
    cout << "\nCó in không? - C/K" ;
    ch = toupper(getch());
    if(ch=='C')
        this->in(); // Gọi đến TS::in() (Vi this là
                    // con trỏ kiểu TS)
}

```

Các lệnh đầu của phương thức sẽ in họ tên thí sinh. Nếu chọn có (bấm phím C), thì câu lệnh:

```
this->in();
```

sẽ được thực hiện. Mặc dù địa chỉ của t[i] (là đối tượng của lớp TS2) được truyền cho con trỏ this, thế nhưng câu lệnh này luôn luôn gọi tới phương thức TS::in(), vì con trỏ this ở đây có kiểu TS và vì in() là phương thức tĩnh. Kết quả là không in được địa chỉ của thí sinh.

Như vậy việc sử dụng các phương thức tĩnh in() (trong các lớp TS và

TS2) đã không đáp ứng được yêu cầu phát triển chương trình. Có một giải pháp rất đơn giản là: Định nghĩa các phương thức in() trong các lớp TS và TS2 như các phương thức ảo (virtual).

§ 3. PHƯƠNG THỨC ẢO VÀ TƯƠNG ỨNG BỘI

3.1. Cách định nghĩa phương thức ảo

Giả sử A là lớp cơ sở, các lớp B, C, D dẫn xuất (trực tiếp hoặc gián tiếp) từ A. Giả sử trong 4 lớp trên đều có các phương thức trùng dòng tiêu đề (trùng kiểu, trùng tên, trùng các đối). Để định nghĩa các phương thức này là các phương thức ảo, ta chỉ cần:

- + Hoặc thêm từ khoá virtual vào dòng tiêu đề của phương thức bên trong định nghĩa lớp cơ sở A.

- + Hoặc thêm từ khoá virtual vào dòng tiêu đề bên trong định nghĩa của tất cả các lớp A, B, C và D.

Ví dụ:

Cách 1:

```
class A
{
    ...
    virtual void hien_thi()
    {
        cout << "\n Đây là lớp A" ;
    };
};
class B : public A
{
    ...
    void hien_thi()
    {
        cout << "\n Đây là lớp B" ;
    };
};
class C : public B
{
    ...
    void hien_thi()
    {
        cout << "\n Đây là lớp C" ;
    };
};
class D : public A
{
    ...
```

```

        void hien_thi()
        {
            cout << "\n Đây là lớp D" ;
        };
    } ;
Cách 2:
class A
{
    ...
    virtual void hien_thi()
    {
        cout << "\n Đây là lớp A" ;
    };
} ;
class B : public A
{
    ...
    virtual void hien_thi()
    {
        cout << "\n Đây là lớp B" ;
    };
} ;
class C : public B
{
    ...
    virtual void hien_thi()
    {
        cout << "\n Đây là lớp C" ;
    };
} ;
class D : public A
{
    ...
    virtual void hien_thi()
    {
        cout << "\n Đây là lớp D" ;
    };
} ;

```

Chú ý: Từ khoá virtual không được đặt bên ngoài định nghĩa lớp. Ví dụ nếu viết như sau là sai (CTBD sẽ báo lỗi).

```

class A
{
    ...
    virtual void hien_thi() ;
} ;
virtual void hien_thi() // Sai
{
    cout << "\n Đây là lớp A" ;
};

```

Cần sửa lại như sau:

```
class A
{
    ...
    virtual void hien_thi() ;
};
void hien_thi() // Đúng
{
    cout << "\n Đây là lớp A" ;
};
```

3.2. Quy tắc gọi phương thức ảo

Để có sự so sánh với phương thức tĩnh, ta nhắc lại quy tắc gọi phương thức tĩnh nêu trong §1.

3.2.1. Quy tắc gọi phương thức tĩnh

Lời gọi tới phương thức tĩnh bao giờ cũng xác định rõ phương thức nào (trong số các phương thức trùng tên của các lớp có quan hệ thừa kế) được gọi:

1. Nếu lời gọi xuất phát từ một đối tượng của lớp nào, thì phương thức của lớp đó sẽ được gọi.
2. Nếu lời gọi xuất phát từ một con trỏ kiểu lớp nào, thì phương thức của lớp đó sẽ được gọi bất kể con trỏ chứa địa chỉ của đối tượng nào.

3.2.2. Quy tắc gọi phương thức ảo

Phương thức ảo chỉ khác phương thức tĩnh khi được gọi từ một con trỏ (trường hợp 2 nêu trong mục 3.2.1). Lời gọi tới phương thức ảo từ một con trỏ chưa cho biết rõ phương thức nào (trong số các phương thức ảo trùng tên của các lớp có quan hệ thừa kế) sẽ được gọi. Điều này phụ thuộc vào đối tượng cụ thể mà con trỏ đang trỏ tới: Con trỏ đang trỏ tới đối tượng của lớp nào thì phương thức của lớp đó sẽ được gọi.

Ví dụ A, B, C và D là các lớp đã định nghĩa trong 3.1. Ta khai báo một con trỏ kiểu A và 4 đối tượng:

```
A *p ; // p là con trỏ kiểu A
A a ; // a là biến đối tượng kiểu A
B b ; // b là biến đối tượng kiểu B
C c ; // c là biến đối tượng kiểu C
D d ; // d là biến đối tượng kiểu D
```

Xét lời gọi tới các phương thức ảo hien_thi sau:

```
p : &a ; // p trỏ tới đối tượng a của lớp A
p->hien_thi() ; // Gọi tới A::hien_thi()
```

```

p = &b;           // p trỏ tới đối tượng b của lớp B
p->hien_thi();    // Gọi tới B::hien_thi()
p = &c;           // p trỏ tới đối tượng c của lớp C
p->hien_thi();    // Gọi tới C::hien_thi()
p = &d;           // p trỏ tới đối tượng d của lớp D
p->hien_thi();    // Gọi tới D::hien_thi()

```

3.3. Tương ứng bội

Chúng ta nhận thấy cùng một câu lệnh

```
p->hien_thi();
```

tương ứng với nhiều phương thức khác nhau. Đây chính là tương ứng bội. Khả năng này rõ ràng cho phép xử lý nhiều đối tượng khác nhau, nhiều công việc, thậm chí nhiều thuật toán khác nhau theo cùng một cách thức, cùng một lược đồ. Điều này sẽ được minh họa trong các mục tiếp theo.

3.4. Liên kết động

Có thể so sánh sự khác nhau giữa phương thức tĩnh và phương thức ảo trên khía cạnh liên kết một lời gọi với một phương thức. Trở lại ví dụ trong 3.2:

```

A *p; // p là con trỏ kiểu A
A a; // a là biến đối tượng kiểu A
B b; // b là biến đối tượng kiểu B
C c; // c là biến đối tượng kiểu C
D d; // d là biến đối tượng kiểu D

```

Nếu `hien_thi()` là các phương thức tĩnh, thì dù `p` chứa địa chỉ của các đối tượng `a`, `b`, `c` hay `d`, thì lời gọi:

```
p->hien_thi();
```

luôn luôn gọi tới phương thức `A::hien_thi()`

Như vậy một lời gọi (xuất phát từ con trỏ) tới phương thức tĩnh luôn luôn liên kết với một phương thức cố định và sự liên kết này xác định trong quá trình biên dịch chương trình.

Cũng với lời gọi:

```
p->hien_thi();
```

như trên, nhưng nếu `hien_thi()` là các phương thức ảo, thì lời gọi này không liên kết cứng với một phương thức cụ thể nào. Phương thức mà nó liên kết (gọi tới) còn chưa xác định trong giai đoạn dịch chương trình. Lời gọi này sẽ:

- + liên kết với `A::hien_thi()`, nếu `p` chứa địa chỉ đối tượng lớp `A`
- + liên kết với `B::hien_thi()`, nếu `p` chứa địa chỉ đối tượng lớp `B`

- + liên kết với C::hien_thi(), nếu p chứa địa chỉ đối tượng lớp C
- + liên kết với D::hien_thi(), nếu p chứa địa chỉ đối tượng lớp D

Như vậy một lời gọi (xuất phát từ con trỏ) tới phương thức ảo không liên kết với một phương thức cố định, mà tùy thuộc vào nội dung con trỏ. Đó là sự liên kết động và phương thức được liên kết (được gọi) thay đổi mỗi khi có sự thay đổi nội dung con trỏ trong quá trình chạy chương trình.

3.5. Quy tắc gán địa chỉ đối tượng cho con trỏ lớp cơ sở

+ Như đã nói trong §1, C++ cho phép gán địa chỉ đối tượng của một lớp dẫn xuất cho con trỏ của lớp cơ sở. Như vậy các phép gán sau (xem 3.2) là đúng:

```
A *p ;    // p là con trỏ kiểu A
A a ;     // a là biến đối tượng kiểu A
B b ;     // b là biến đối tượng kiểu B
C c ;     // c là biến đối tượng kiểu C
D d ;     // d là biến đối tượng kiểu D
p = &a;   // p và a cùng lớp A
p = &b;   // p là con trỏ lớp cơ sở, b là
           // đối tượng lớp dẫn xuất
p = &c;   // p là con trỏ lớp cơ sở, c là
           // đối tượng lớp dẫn xuất
p = &d;   // p là con trỏ lớp cơ sở, d là
           // đối tượng lớp dẫn xuất
```

+ *Tuy nhiên cần chú ý là:* Không cho phép gán địa chỉ đối tượng của lớp cơ sở cho con trỏ của lớp dẫn xuất. Như vậy ví dụ sau là sai:

```
B *q ;
A a ;
q = &a;
```

Sai vì: Gán địa chỉ đối tượng của lớp cơ sở A cho con trỏ của lớp dẫn xuất B

3.6. Ví dụ

Ta sửa chương trình trong §1 bằng cách định nghĩa các phương thức xuất() là ảo. Khi đó bốn câu lệnh:

```
hien(&a);
hien(&b);
hien(&c);
hien(&d);
```

trong hàm main (của chương trình dưới đây) sẽ lần lượt gọi tới 4 phương thức khác nhau:

```
A::xuat()
B::xuat()
```

```

        C::xuat()
        D::xuat()
//CT6-01B
// Phương thức ảo và tương ứng bội
#include <conio.h>
#include <stdio.h>
#include <iostream.h>
#include <ctype.h>
class A
{
    private:
        int n;
    public:
        A()
        {
            n=0;
        }
        A(int n1)
        {
            n=n1;
        }
        virtual void xuat()
        {
            cout << "\nLop A: "<< n;
        }
        int getN()
        {
            return n;
        }
};
class B:public A
{
    public:
        B():A()
        {
        }
        B(int n1):A(n1)
        {
        }
        void xuat()
        {
            cout << "\nLop B: "<<getN();
        }
};
class C:public A
{
    public:
        C():A()
        {
        }
};

```

```

        C(int n1):A(n1)
        {
        }
        void xuat()
        {
            cout << "\nLop C: " << getN();
        }
    };
class D: public C
{
    public:
        D():C()
        {
        }
        D(int n1):C(n1)
        {
        }
        void xuat()
        {
            cout << "\nLop D: " << getN();
        }
};
void hien(A *p)
{
    p->xuat();
}
void main()
{
    A a(1);
    B b(2);
    C c(3);
    D d(4);
    clrscr();
    hien(&a);
    hien(&b);
    hien(&c);
    hien(&d);
    getch();
}

```

3.5. Sự thừa kế của các phương thức ảo

Cũng giống như các phương thức thông thường khác, phương thức ảo cũng có tính thừa kế. Chẳng hạn trong chương trình trên (mục 3.4) ta bỏ đi phương thức xuat() của lớp D, thì câu lệnh:

```
hien(&d) ;
```

(câu lệnh gần cuối trong hàm main) sẽ gọi tới C::xuat(), phương thức này được kế thừa trong lớp D (vì D dẫn xuất từ C).

§4. SỰ LINH HOẠT CỦA PHƯƠNG THỨC ẢO TRONG PHÁT TRIỂN NÂNG CẤP CHƯƠNG TRÌNH

Ví dụ về các lớp TS và TS2 trong §2 đã chỉ ra sự hạn chế của phương thức tĩnh trong việc sử dụng tính thừa kế để nâng cấp, phát triển chương trình. Trong §2 cũng đã chỉ ra lớp TS2 chưa đáp ứng được yêu cầu nêu ra là in địa chỉ của thí sinh. Giải pháp cho vấn đề này rất đơn giản: Thay các phương thức tĩnh in() bằng cách dùng chúng như các phương thức ảo. Chương trình khi đó sẽ như sau:

```
//CT6-03B
// Sự linh hoạt của phương thức ảo
// Lớp TS TS2
#include <conio.h>
#include <stdio.h>
#include <iostream.h>
#include <ctype.h>
class TS
{
private:
    char ht[25];
    int sobd;
    float td;
public:
    void nhap()
    {
        cout << "\nHo ten: " ;
        fflush(stdin);
        gets(ht);
        cout << "So bao danh: " ;
        cin >> sobd;
        cout << "Tong diem: " ;
        cin >> td;
    }
    virtual void in()
    {
        fprintf(stdprn, "\n\nHo ten: %s", ht);
        fprintf(stdprn, "\nSo bao danh: %d", sobd);
        fprintf(stdprn, "\nTong diem: %0.1f", td);
    }
    void xem_in()
    {
        int ch;
        cout << "\nHo ten: " << ht ;
        cout << "\nCo in khong? - C/K" ;
        ch = toupper(getch());
        if (ch=='C')
            this->in(); // Vì in() là phương thức ảo nên
                       //có thể gọi đến TS::in() hoặc TS2::in()
```

```

    }
} ;
class TS2:public TS
{
    private:
        char dc[30] ; // Địa chỉ
    public:
        void nhap()
        {
            TS::nhap();
            cout << "Địa chỉ: " ;
            fflush(stdin); gets(dc);
        }
        void in()
        {
            TS::in();
            fprintf(stdout, "\nĐịa chỉ: %s", dc);
        }
};
void main()
{
    TS2 t{100};
    int i, n;
    cout << "\nSố thí sinh: ";
    cin >> n;
    for (i=1; i<=n; ++i)
        t[i].nhap();
    for (i=1; i<=n; ++i)
        t[i].xem_in();
    getch();
}

```

Khi thực hiện chương trình này, chúng ta nhận thấy: Dữ liệu thí sinh in ra đã có địa chỉ. Điều này có thể giải thích như sau:

Xét câu lệnh (thứ 2 từ dưới lên trong hàm main):

```
t[i].xem_in() ;
```

Câu lệnh này gọi tới phương thức xem_in của lớp TS2 (vì t[i] là đối tượng của lớp TS2). Nhưng lớp TS2 không định nghĩa phương thức xem_in, nên phương thức TS::xem_in() sẽ được gọi tới. Hãy theo dõi phương thức này:

```

void xem_in()
{
    int ch;
    cout << "\nHọ tên: " << ht ;
    cout << "\nCó in không? - C/K" ;
    ch = toupper(getch());
    this->in(); // Vì in() là phương thức ảo nên
               //có thể gọi đến TS::in() hoặc TS2::in()
}

```

Các lệnh đầu của phương thức sẽ in họ tên thí sinh. Nếu chọn Có (bấm phím C), thì câu lệnh:

```
    this->in() ;
```

sẽ được thực hiện. Địa chỉ của t[i] (là đối tượng của lớp TS2) được truyền cho con trỏ this (của lớp cơ sở TS). Vì in() là phương thức ảo và vì this đang trỏ tới đối tượng t[i] của lớp TS2, nên câu lệnh này gọi tới phương thức TS2::in(). Trong phương thức TS2::in() có in địa chỉ của thí sinh.

Như vậy việc sử dụng các phương thức tĩnh in() (trong các lớp TS và TS2) đã không đáp ứng được yêu cầu phát triển chương trình. Có một giải pháp rất đơn giản là: Định nghĩa các phương thức in() trong các lớp TS và TS2 như các phương thức ảo (virtual).

§ 5. LỚP CƠ SỞ TRỪU TƯỢNG

5.1. Lớp cơ sở trừu tượng

Một lớp cơ sở trừu tượng là một lớp chỉ được dùng làm cơ sở cho các lớp khác. Không hề có đối tượng nào của một lớp trừu tượng được tạo ra cả, bởi vì nó chỉ được dùng để định nghĩa một số khái niệm tổng quát, chung cho các lớp khác. Một ví dụ về lớp trừu tượng là lớp CON_VAT (con vật), nó sẽ dùng làm cơ sở để xây dựng các lớp con vật cụ thể như lớp CON_CHO (con chó), CON_MEO (con mèo),... (xem ví dụ bên dưới)

Trong C++, thuật ngữ “Lớp trừu tượng” đặc biệt áp dụng cho các lớp có chứa các phương thức ảo thuần túy. Phương thức ảo thuần túy là một phương thức ảo mà nội dung của nó không có gì. Cách thức định nghĩa một phương thức ảo thuần túy như sau:

```
virtual void tên_phương_thức() = 0 ;
```

Ví dụ:

```
class A
{
    public:
        virtual void nhap() = 0 ;
        virtual void xuat() = 0 ;
        void chuong();
} ;
```

Trong ví dụ trên, thì A là lớp cơ sở trừu tượng. Các phương thức nhap và xuat được khai báo là các lớp ảo thuần túy (bằng cách gán số 0 cho chúng thay cho việc cài đặt các phương thức này). Phương thức chuong() là một

phương thức bình thường và sẽ phải có một định nghĩa ở đâu đó cho phương thức này.

Không có đối tượng nào của một lớp trừu tượng lại có thể được phát sinh. Tuy nhiên các con trỏ và các biến tham chiếu đến các đối tượng của lớp trừu tượng thì vẫn hợp lệ. Bất kỳ lớp nào dẫn xuất từ một lớp cơ sở trừu tượng phải định nghĩa lại tất cả các phương thức thuần ảo mà nó thừa hưởng, hoặc bằng các phương thức ảo thuần túy, hoặc bằng những định nghĩa thực sự. Ví dụ:

```
class B : public A
{
    public:
        virtual void  nhap() = 0 ;
        virtual void  xuat()
        {
            // Các câu lệnh
        }
};
```

Theo ý nghĩa về hướng đối tượng, ta vẫn có thể có một lớp trừu tượng mà không nhất thiết phải chứa đựng những phương thức thuần túy ảo.

Một cách tổng quát mà nói thì bất kỳ lớp nào mà nó chỉ được dùng làm cơ sở cho những lớp khác đều có thể được gọi là “lớp trừu tượng”. Một cách dễ dàng để nhận biết một lớp trừu tượng là xem có dùng lớp đó để khai báo các đối tượng hay không?. Nếu không thì đó là lớp cơ sở trừu tượng.

5.2. Ví dụ

Giả sử có 20 ô, mỗi ô có thể nuôi một con chó hoặc một con mèo. Yêu cầu xây dựng chương trình gồm các chức năng:

- + Nhập một con vật mới mua (hoặc chó, hoặc mèo) vào ô rỗng đầu tiên.
- + Xuất (đem bán) một con vật (hoặc chó, hoặc mèo).
- + Thống kê các con vật đang nuôi trong 20 ô.

Chương trình được tổ chức như sau:

+ Trước tiên định nghĩa lớp CON_VAT là lớp cơ sở ảo. Lớp này có một thuộc tính là tên con vật và một phương thức ảo dùng để xưng tên.

+ Hai lớp là CON_MEO và CON_CHO được dẫn xuất từ lớp CON_VAT

+ Cuối cùng là lớp DS_CON_VAT (Danh sách con vật) dùng để quản lý chung cả mèo và chó. Lớp này có 3 thuộc tính là: số con vật cực đại (chính bằng số ô), số con vật đang nuôi và một mảng con trỏ kiểu CON_VAT. Mỗi

phần tử mảng sẽ chứa địa chỉ của một đối tượng kiểu CON_MEO hoặc CON_CHO.

Lớp sẽ có 3 phương thức để thực hiện 3 chức năng nêu trên của chương trình. Nội dung chương trình như sau:

```
//CT6-04
// Lop co so truu tuong
// Lop CON_VAT
#include <conio.h>
#include <stdio.h>
#include <iostream.h>
#include <ctype.h>
#include <string.h>
class CON_VAT
{
protected:
    char *ten;
public:
    CON_VAT()
    {
        ten = NULL;
    }
    CON_VAT(char *ten1)
    {
        ten = strdup(ten1);
    }
    virtual void xung_ten()
    {
    }
};
class CON_MEO:public CON_VAT
{
public:
    CON_MEO() : CON_VAT()
    {
    }
    CON_MEO(char *ten1) : CON_VAT(ten1)
    {
    }
    virtual void xung_ten()
    {
        cout << "\nToi la chu meo: " << ten ;
    }
};
class CON_CHO:public CON_VAT
{
public:
    CON_CHO() : CON_VAT()
    {
    }
}
```

```

        CON_CHO(char *ten1) : CON_VAT(ten1)
        {
        }
        virtual void xung_ten()
        {
            cout << "\nToi la chu cho: " << ten ;
        }
    };

class DS_CON_VAT // Danh sach con vat
{
    private:
        int max_so_con_vat;
        int so_con_vat;
        CON_VAT **h ;
    public:
        DS_CON_VAT(int max);
        ~DS_CON_VAT();
        int nhap(CON_VAT *c);
        CON_VAT* xuat(int n);
        void thong_ke();
    } ;

DS_CON_VAT::DS_CON_VAT(int max)
{
    max_so_con_vat = max;
    so_con_vat = 0;
    h = new CON_VAT*[max];
    for (int i=0; i<max; ++i)
        h[i] = NULL;
}

DS_CON_VAT::~~DS_CON_VAT()
{
    max_so_con_vat = 0;
    so_con_vat = 0;
    delete h;
}

int DS_CON_VAT::nhap(CON_VAT *c)
{
    if (so_con_vat==max_so_con_vat)
        return 0;
    int i=0;
    while (h[i]!=NULL) ++i;
    h[i]=c;
    so_con_vat++ ;
    return (i+1);
}

CON_VAT* DS_CON_VAT::xuat(int n)
{
    if (n<1 || n > max_so_con_vat)
        return NULL ;
    --n ;
    if (h[n])

```

```

        {
            CON_VAT *c = h[n];
            h[n]=NULL;
            so_con_vat--;
            return c;
        }
        else
            return NULL;
    }
void DS_CON_VAT::thong_ke()
{
    if (so_con_vat)
    {
        cout << "\n" ;
        for (int i=0; i<max_so_con_vat; ++i)
            if (h[i])
                h[i]->xung_ten();
    }
}
CON_CHO c1("MUC");
CON_CHO c2("VEN");
CON_CHO c3("LAI");
CON_CHO c4("NHAT");
CON_CHO c5("BONG");
CON_MEO m1("MUOP");
CON_MEO m2("DEN");
CON_MEO m3("TRANG");
CON_MEO m4("TAM THE");
CON_MEO m5("VANG");
void main()
{
    DS_CON_VAT d(20);
    clrscr();
    d.nhap(&c1);
    int im2 = d.nhap(&m2);
    d.nhap(&c3);
    d.nhap(&m1);
    int ic4 = d.nhap(&c4);
    d.nhap(&c5);
    d.nhap(&m5);
    d.nhap(&c2);
    d.nhap(&m3);
    d.thong_ke();
    d.xuat(im2);
    d.xuat(ic4);
    d.thong_ke();
    getch();
}

```

Chú ý: Theo quan điểm chung về cách thức sử dụng, thì lớp CON_VAT là lớp cơ sở trừu tượng. Tuy nhiên theo quan điểm của C++ thì lớp này chưa

phải là lớp cơ sở trừu tượng, vì trong lớp không có các phương thức thuần túy ảo. Phương thức `xung_ten`:

```
virtual void xung_ten()
{
}
```

là phương thức ảo, được định nghĩa đầy đủ, mặc dù thân của nó là rỗng.

Do vậy khai báo:

```
CON_VAT cv("Con vat chung");
```

vẫn được C++ chấp nhận.

Bây giờ nếu định nghĩa lại phương thức `xung_ten` như sau:

```
virtual void xung_ten() = 0 ;
```

thì nó trở thành phương thức thuần ảo và C++ sẽ quan niệm lớp `CON_VAT` là lớp trừu tượng. Khi đó câu lệnh khai báo:

```
CON_VAT cv("Con vat chung");
```

sẽ bị C++ bắt lỗi với thông báo:

```
Cannot create instance of abstract class 'CON_VAT'
```

§6. SỬ DỤNG TƯƠNG ỨNG BỘI VÀ PHƯƠNG THỨC ẢO

6.1. Chiến lược sử dụng tương ứng bội

Tương ứng bội cho phép xét các vấn đề khác nhau, các đối tượng khác nhau, các phương pháp khác nhau, các cách giải quyết khác nhau theo cùng một lược đồ chung.

Các bước áp dụng tương ứng bội có thể tổng kết lại như sau:

1. Xây dựng lớp cơ sở trừu tượng bao gồm những thuộc tính chung nhất của các thực thể cần quản lý. Đưa vào các phương thức ảo hay thuần ảo dùng để xây dựng các nhóm phương thức ảo cho các lớp dẫn xuất sau này. Mỗi nhóm phương thức ảo sẽ thực hiện một chức năng nào đó trên các lớp.

2. Xây dựng các lớp dẫn xuất bắt đầu từ lớp cơ sở trừu tượng. Số mức dẫn xuất là không hạn chế. Các lớp dẫn xuất sẽ mô tả các đối tượng cụ thể cần quản lý.

3. Xây dựng các phương thức ảo trong các dẫn xuất. Các phương thức này tạo thành các nhóm phương thức ảo trong sơ đồ các lớp có quan hệ thừa kế.

4. Xây dựng lớp quản lý các đối tượng. Dữ liệu của lớp này là một dãy con trỏ của lớp cơ sở trừu tượng ban đầu. Các con trỏ này có thể chứa địa chỉ đối tượng của các lớp dẫn xuất. Do vậy có thể dùng các con trỏ này để thực hiện các thao tác trên các đối tượng của bất kỳ lớp dẫn xuất nào.

6.2. Ví dụ

Chương trình quản lý các con vật trong §5 là một ví dụ về cách sử dụng tương ứng bội. Dưới đây là một ví dụ khác. Giả sử có 4 hình vẽ: Đoạn thẳng, hình tròn, hình chữ nhật và hình vuông. Bốn hình cho hiện thẳng hàng trên màn hình tạo thành một bức tranh. Nếu thay đổi thứ tự các hình sẽ nhận được các bức tranh khác nhau. Chương trình dưới đây sẽ cho hiện tất cả các bức tranh khác nhau. Chương trình được tổ chức theo các bước nêu trong 6.1:

- + Lớp cơ sở trừu tượng là lớp HINH (hình) gồm một thuộc tính mau (màu) và một phương thức ảo thuần túy:

```
virtual void draw(int x, int y) = 0 ;
```

- + Các lớp dẫn xuất trực tiếp từ lớp hình là : DTHANG, HTRON và CHUNHAT.

- + Lớp VUONG dẫn xuất từ lớp CHUNHAT.

- + Lớp quản lý chung là lớp picture có thuộc tính là một mảng con trỏ kiểu HINH gồm 4 phần tử dùng để chứa địa chỉ 4 đối tượng: DTHANG, HTRON, CHUNHAT và VUONG. Sử dụng phương thức draw gọi từ 4 phần tử mảng nói trên sẽ nhận được một bức tranh. Bằng cách hoán vị các phần tử này, sẽ nhận được tất cả các bức tranh khác nhau.

```
//CT6-05
// Lop co so truu tuong
// Lop hinh hoc
#include <conio.h>
#include <graphics.h>
class HINH
{
private:
    int mau;
public:
    HINH(int m)
    {
        mau = m;
    }
    int getmau()
    {
        return mau;
    }
    virtual void draw(int x, int y) = 0;
};
class DTHANG : public HINH
{
private:
    int dodai;
```

```

public:
    DTHANG(int d, int m):HINH(m)
    {
        dodai = d ;
    }
    virtual void draw(int x, int y)
    {
        setcolor(getmau()) ;
        line(x,y,x+dodai,y);
    }
};

class CHUNHAT: public HINH
{
private:
    int rong, cao;
public:
    CHUNHAT(int r, int c, int m):HINH(m)
    {
        rong = r; cao = c;
    }
    virtual void draw(int x, int y )
    {
        setcolor(getmau()) ;
        rectangle(x,y,x+rong,y+cao);
        setfillstyle(1,getmau());
        floodfill(x+rong/2,y+cao/2, getmau() );
    }
};

class VUONG : public CHUNHAT
{
public:
    VUONG(int a, int m): CHUNHAT(a,a,m)
    {
    }
};

class HTRON: public HINH
{
private:
    int bk; //Ban kinh
public:
    HTRON(int bkl, int m):HINH(m)
    {
        bk = bkl;
    }
    virtual void draw(int x, int y)
    {
        setcolor(getmau()) ;
        circle(x+bk,y+bk,bk);
        setfillstyle(1,getmau());
        floodfill(x + bk, y + bk,getmau());
    }
};

```

```

class picture
{
    private:
        HINH *h[4];
    public:
        picture(HINH *h0,HINH *h1,HINH *h2,HINH *h3)
        {
            h[0]=h0;
            h[1]=h1;
            h[2]=h2;
            h[3]=h3;
        }
        void paint(int *k);
        void listpaint();
};

void picture::paint(int *k)
{
    for (int i=0; i<4; ++i)
        h[k[i]]->draw(10+i*150, 200);
}

void picture::listpaint()
{
    int k[4],i1,i2,i3,i4;
    for (i1=0;i1<4;++i1)
        for (i2=0;i2<4;++i2)
            if (i2!=i1)
                for (i3=0;i3<4;++i3)
                    if (i3!=i2 && i3!=i1)
                        for (i4=0;i4<4;++i4)
                            if (i4!=i3 && i4!=i2 && i4!=i1)
                                {
                                    k[0]=i1;k[1]=i2;
                                    k[2]=i3;k[3]=i4;
                                    paint(k);
                                    getch();
                                    cleardevice();
                                }
}

    DTIANG dt(120,14);
    HTRON ht(60,RED);
    CHUNHAT cn(120,100,MAGENTA);
    VUONG v(120,CYAN);
};

void main()
{
    int mh=0,mode=0;
    initgraph(&mh,&mode,"");
    picture pic(&dt,&ht,&cn,&v);
    pic.listpaint();
    getch();
    closegraph();
}

```

§ 7. XỬ LÝ CÁC THUẬT TOÁN KHÁC NHAU

Có thể sử dụng tương ứng bội để tổ chức thực hiện các thuật toán khác nhau trên cùng một bài toán như sau:

- + Lớp cơ sở trừu tượng sẽ chứa dữ liệu bài toán và một phương thức ảo.
- + Mỗi lớp dẫn xuất ứng với một thuật toán cụ thể. Phương thức ảo của lớp dẫn xuất sẽ thực hiện một thuật toán cụ thể.
- + Sử dụng một mảng con trỏ của lớp cơ sở và gán cho mỗi phần tử mảng địa chỉ của một đối tượng của lớp dẫn xuất. Sau đó dùng các phần tử mảng con trỏ để gọi tới các phương thức ảo. Bằng cách đó sẽ thực hiện cùng một bài toán theo các thuật toán khác nhau và dễ dàng so sánh hiệu quả của các thuật toán.

Ví dụ sau minh họa việc thực hiện bài toán sắp xếp dãy số nguyên theo thứ tự tăng bằng cách dùng đồng thời 3 thuật toán: Thuật toán lựa chọn (Select_Sort), thuật toán sắp xếp nhanh (Quick_Sort) và thuật toán vun đống (Heap_Sort). Chương trình gồm 4 lớp:

- + Lớp cơ sở trừu tượng:

```
class sort
{
protected:
    int *a;
    void hoan_vi(long i, long j) ;
public:
    virtual void sapxep(int *a1, long n) ;
};
```

Lớp này gồm:

- Một thành phần dữ liệu là con trỏ a trỏ tới một vùng nhớ chứa dãy số nguyên cần sắp xếp.
- Phương thức hoan_vi(i,j) dùng để hoán vị các phần tử a[i] và a[j]. Phương thức này được dùng trong 3 lớp dẫn xuất bên dưới.
- Phương thức ảo sapxep(a1,n) dùng để sắp xếp dãy n số nguyên chứa trong mảng a1.

+ Ba lớp dẫn xuất là: SELECT_SORT, QUICK_SORT và HEAP_SORT. Mỗi lớp đều có phương thức ảo:

```
virtual void sapxep(int *a1, long n) ;
```

để thực hiện việc sắp xếp theo một thuật toán cụ thể.

- + Trong hàm main() sẽ tạo ra một dãy 30000 số nguyên một cách ngẫu nhiên, sau đó lần lượt sử dụng 3 thuật toán sắp xếp để so sánh. Kết quả như sau:

Thời gian sắp xếp theo thuật toán Select sort là: 19.20 giây

Thời gian sắp xếp theo thuật toán Quick sort là: 0.11 giây

Thời gian sắp xếp theo thuật toán Heap sort là: 0.44 giây

Nội dung chương trình như sau:

```
//CT6-06
// Lop co so truu tuong
// Lop sort.
#include <conio.h>
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <iostream.h>
#include <dos.h>
class sort
{
protected:
    int *a;
    void hoan_vi(long i, long j)
    {
        int tq = a[i];
        a[i] = a[j];
        a[j] = tq;
    }
public:
    virtual void sapxep(int *a1, long n)
    {
        a = a1;
    }
};
class select_sort : public sort
{
public:
    virtual void sapxep(int *a1, long n) ;
};
void select_sort::sapxep(int *a1, long n)
{
    long i,j,r;
    sort::sapxep(a1,n);
    for (i=1; i<n; ++i)
    {
        r=i;
        for (j=i+1; j<n; ++j)
            if(a[j] < a[r]) r = j;
        if(r!=i) hoan_vi(i,r);
    }
}
```

```

    }
}
class quick_sort : public sort
{
    private:
        void q_sort(long l, long r);
    public:
        virtual void sapxep(int *a1, long n) ;
} ;
void quick_sort::q_sort(long l, long r)
{
    int x;
    long i, j;
    if (l < r)
    {
        x = a[l]; i = l; j = r+1;
        do
        {
            ++i; --j;
            while (i<r && a[i] < x) ++i ;
            while (a[j] > x) --j ;
            if (i<j) hoan_vi(i,j);
        } while (i<j);
        hoan_vi(l, j);
        q_sort(l, j-1);
        q_sort(j+1, r);
    }
}
void quick_sort::sapxep(int *a1, long n)
{
    sort::sapxep(a1,n);
    q_sort(1,n);
}
class heap_sort : public sort
{
    private:
        void shift(long i, long n);
    public:
        virtual void sapxep(int *a1, long n) ;
} ;
void heap_sort::shift(long i, long n)
{
    long l, r, k;
    l = 2*i; r = l+1;
    if (l>n) return;
    if (l==n)

```

```

    {
        if (a[i]<a[l]) hoan_vi(i,l);
        return;
    }
    if (a[l] > a[r])
        k = l;
    else
        k = r;
    if (a[i]> a[k])
        return;
    else
    {
        hoan_vi(i,k);
        shift(k,n);
    }
}

void heap_sort::sapxep(int *a1, long n)
{
    long i;
    sort::sapxep(a1,n);
    /* Tao dong */
    for (i=n/2 ; i>=1; --i) shift(i,n);
    /* Lap */
    for (i=n ; i>=2; --i)
    {
        hoan_vi(1,i);
        shift(1,i-1);
    }
}

void main()
{
    long i,n;
    struct time t1,t2;
    int *a, k, tg, sec, hund;
    n=30000;
    a=(int*) malloc((n+1)*sizeof(int));
    if (a==NULL)
    {
        puts("\nLoi BN"); getch();
        exit(0);
    }
    sort *s[3];
    select_sort ss;
    quick_sort qs;
    heap_sort hs;
    s[0]=&ss; s[1]=&qs; s[2]=&hs;

```

```

clrscr();
for (k=0; k<3; ++k)
{
    srand(5000);
    for (i 1; i<=n; ++i)
        a[i]=rand();
    gettimeofday(&t1);
    s[k]->sapxep(a,n);
    gettimeofday(&t2);
    tg = (t2.ti_sec - t1.ti_sec)*100 + t2.ti_hund -
t1.ti_hund ;
    sec = tg / 100;
    hund = tg % 100;
    printf("\n Sap xep %d %d %d %d %d",k+1,
t2.ti_sec,t2.ti_hund,t1.ti_sec,t1.ti_hund);
    printf("\n Sap xep %d Thoi gian %d sec %d hund",
k+1, sec, hund);
}
getch();
}

```

THAO TÁC TRÊN CÁC TẬP TIN

Vì trong C++ vẫn sử dụng các hàm của C. Chính vì vậy, chương này sẽ trình bày các thao tác trên tệp như: tạo một tệp mới, ghi dữ liệu từ bộ nhớ lên tệp, đọc dữ liệu từ tệp vào bộ nhớ,... Các thao tác tệp được thực hiện nhờ các hàm thư viện. Các hàm này được chia thành hai nhóm: cấp 1 và cấp 2. Mỗi hàm (dù cấp 1 hay cấp 2) đều có thể truy xuất theo cả hai kiểu nhị phân và văn bản (xem §2). Tuy nhiên việc chọn kiểu phù hợp cho mỗi hàm là cần thiết.

Các hàm cấp 1 (còn gọi là các hàm nhập/xuất hệ thống) thực hiện việc đọc/ghi như DOS.

Các hàm cấp 2 được xây dựng từ các hàm cấp 1 nên dễ sử dụng và có nhiều khả năng hơn.

Do các hàm cấp 2 có nhiều kiểu truy xuất và dễ dùng hơn so với các hàm cấp 1, nên trong chương trình C các hàm cấp 2 được ưa chuộng hơn. Mặt khác vì các hàm cấp 1 ở mức độ sâu hơn, gần DOS hơn nên tốc độ truy nhập sẽ nhanh hơn.

Dưới đây sẽ trình bày hệ thống truy xuất cấp 2. Các hàm vào/ra cấp 1 cũng như một vài vấn đề khác liên quan đến tệp trong C, nếu bạn đọc quan tâm có thể tìm đọc cuốn:

“Kỹ thuật lập trình C – Cơ sở và nâng cao” của GS. Phạm Văn Ất.

Đ1. KIỂU NHẬP XUẤT NHỊ PHÂN VÀ VĂN BẢN

Trước khi giới thiệu các kiểu xuất nhập nhị phân và văn bản chúng ta cũng cần biết cấu trúc chung của một tệp tin trên đĩa. Một tệp tin (dù nó được xây dựng bằng cách nào) đơn giản chỉ là một dãy các byte (có giá trị từ 0 đến 255) ghi trên đĩa. Số byte của dãy chính là độ dài của tệp.

1.1. Kiểu nhị phân

1.1.1. Bảo toàn dữ liệu. Trong quá trình nhập xuất dữ liệu không bị biến đổi. Dữ liệu ghi trên tệp theo các byte nhị phân như trong bộ nhớ.

1.1.2. Mã kết thúc tệp

Trong khi đọc nếu gặp cuối tệp thì ta nhận được mã kết thúc tệp EOF (định nghĩa trong stdio.h bằng -1) và hàm feof cho giá trị khác không. Tại sao lại chọn số -1 làm mã kết thúc tệp? Lý do rất đơn giản: Nếu chưa gặp cuối tệp thì sẽ đọc được một byte có giá trị từ 0 đến 255. Như vậy giá trị -1 sẽ không trùng với bất kỳ byte nào đọc được từ tệp.

1.2. Kiểu văn bản

Kiểu nhập xuất văn bản chỉ khác kiểu nhị phân khi xử lý ký tự chuyển dòng (mã 10) và ký tự mã 26. Đối với các ký tự khác, hai kiểu đều đọc ghi như nhau.

1.2.1. Mã chuyển dòng:

- Khi ghi, một ký tự LF (mã 10) được chuyển thành 2 ký tự CR (mã 13) và LF.

_ Khi đọc, 2 ký tự liên tiếp CR và LF trên tệp chỉ cho ta một ký tự LF.

Ví dụ xét hàm

```
fputc(10,fp);
```

Nếu tệp fp mở theo kiểu nhị phân, hàm sẽ ghi lên tệp một ký tự mã 10. Nhưng nếu fp mở theo kiểu văn bản thì hàm ghi lên tệp hai mã là 13 và 10.

Điều này cốt để phù hợp với DOS, vì các dòng trên tệp văn bản của DOS được kết thúc bởi hai mã 13 và 10.

1.2.2. Mã kết thúc tệp

Trong khi đọc nếu gặp ký tự có mã 26 hoặc cuối tệp thì ta nhận được mã kết thúc tệp EOF (số -1) và hàm feof(fp) cho giá trị khác không (số 1). Điều này cốt để phù hợp với một số hệ soạn thảo (như C, Pascal). Tệp soạn thảo trong các hệ này kết thúc bởi mã 26.

1.2.3. Tệp văn bản của DOS. Kiểu văn bản trong C tạo ra các tệp văn bản có cấu trúc như các tệp văn bản chuẩn của DOS. Các hệ soạn thảo của C, Pascal, Foxpro, Sidekick cũng tạo ra các tệp như vậy. Như vậy có thể dùng tệp văn bản để nhận kết quả từ một chương trình C dùng cho chương trình Pascal. Một ứng dụng khác của kiểu nhập xuất văn bản là: Các kết quả của chương trình có thể đưa ra một tệp văn bản, rồi in tệp này ở bất kỳ chỗ nào khác có máy in.

1.3. Ví dụ minh họa

Chương trình sau tạo 2 tệp có tên là vb và np. Trong chương trình dùng các hàm:

- + fopen để mở tệp,
- + fputc để ghi một ký tự lên tệp,
- + fclose để đóng tệp.

```
#include <stdio.h>
main()
{
    FILE *fwb, *fnp; /* Khai báo 2 biến
                     con trỏ tệp */
    /* Mở tệp vb để ghi theo kiểu văn bản
       gán con trỏ tệp cho biến fwb */
    fwb=fopen("vb", "wt");
    /* Mở tệp np để ghi theo kiểu nhị phân
       gán con trỏ tệp cho biến fnp */
    fnp=fopen("np", "wb");
    /* Ghi các ký tự lên tệp fwb */
    fputc('A', fwb);
    fputc(26, fwb);
    fputc(10, fwb);
    fputc('B', fwb);
    /* Ghi các ký tự lên tệp fnp */
    fputc('A', fnp);
    fputc(26, fnp);
    fputc(10, fnp);
    fputc('B', fnp);
    fclose(fwb); /* Đóng tệp fwb */
    fclose(fnp); /* Đóng tệp fnp */
}
```

Kết quả:

- + Tệp vb có các ký tự ứng với các mã: 65 26 13 10 66
- + Tệp np có các ký tự ứng với các mã: 65 26 10 66

Chú ý: Nếu dùng kiểu văn bản để đọc tệp vb hay tệp np, thì ta chỉ nhận được một ký tự đầu (mã 65) vì khi gặp ký tự thứ hai (mã 26) thì ta nhận được mã kết thúc EOF.

1.4. Một ví dụ khác. Trong 1.3 đã nhắc nhở rằng: muốn đọc tất cả các ký tự của tệp, ta cần dùng hàm fgetc theo kiểu nhị phân. Ta xét thêm một ví dụ nữa để thấy việc chọn kiểu là cần thiết. Xét chương trình:

```
#include <stdio.h>
main()
{
    FILE f; /* Khai báo biến con trỏ tệp */
    /* Mở tệp sl để ghi theo kiểu
       văn bản. Gán tệp sl với con trỏ f */
    f=fopen("sl", "wt");
```

```

/* Ghi 3 dòng lên tệp f */
fprintf(f,"%2d\n%2d\n%2d",56,7,8);
/* Đóng tệp */
fclose(f); /* Đóng tệp f */
}

```

Hàm `fprintf` đưa kết quả ra tệp theo cách như hàm `printf`. Vì tệp `f` mở theo kiểu văn bản nên ký tự xuống dòng '\n' được ghi thành 2 mã 13 và 10. Kết quả là 10 ký tự ứng với các mã sau được ghi lên tệp:

53 54 13 10 32 55 13 10 32 56

(53 là mã của chữ số 5, 32 là mã của khoảng trống,...)

Cấu trúc trên của tệp `sl` phù hợp với DOS, vì vậy khi dùng lệnh `type` (của DOS) có thể cho hiện hay in nội dung của tệp `sl` thành 3 dòng.

Bây giờ nếu ta mở tệp `sl` theo kiểu nhị phân bằng cách dùng câu lệnh:

```
f=fopen("sl","wb");
```

thì tệp `sl` sẽ gồm 8 mã sau:

53 54 10 32 55 10 32 56

Cấu trúc này không phù hợp với DOS và khi dùng lệnh `TYPE` sẽ không nhận được 3 dòng mong muốn.

§2. GIỚI THIỆU CHUNG VỀ CÁC HÀM XỬ LÝ TỆP

Các hàm cấp 2 sử dụng cấu trúc `FILE` và mã kết thúc EOF, tất cả đều được khai báo và định nghĩa trong `<stdio.h>`. Mã EOF bằng (-1) còn cấu trúc `FILE` gồm các thành phần dùng để quản lý tệp tin như:

- + level cho biết có còn dữ liệu trong vùng đệm không,
- + bsize độ lớn vùng đệm (mặc định là 512 byte)
- + flags các cờ trạng thái

...

Mặc dù mỗi hàm đều có thể dùng cả hai kiểu, nhưng như đã nói ở trên, việc lựa chọn kiểu để dùng cho mỗi hàm là cần thiết để tránh nhầm lẫn, sai sót. Dưới đây chúng ta sẽ phân loại các hàm theo sự lựa chọn này.

2.1. Các hàm dùng chung cho cả hai kiểu

1. `fopen` dùng để mở tệp. Trước khi làm việc với tệp cần phải mở nó.
2. `fclose` dùng để đóng tệp. Cần đóng tệp trước khi kết thúc chương trình hay khi không làm việc với tệp nữa.
3. `fcloseall` dùng để đóng tất cả các tệp đang mở.

4. fflush dùng để làm sạch vùng đệm của tệp.
5. fflushall dùng để làm sạch vùng đệm của các tệp đang mở.
6. ferror cho biết có lỗi (khác 0) hay không lỗi (bằng 0).
7. perror thông báo lỗi trên màn hình (khi biết có lỗi).
8. feof cho biết đã gặp cuối tệp hay chưa.
9. unlink và remove dùng để loại tệp trên đĩa.
10. rewind dùng để chuyển con trỏ chỉ vị về đầu tệp.
11. fseek dùng để di chuyển con trỏ chỉ vị đến vị trí bất kỳ trên tệp (hàm này chỉ nên dùng theo kiểu nhị phân).
12. ftell cho biết vị trí hiện tại của con trỏ chỉ vị.

2.2. Các hàm nhập xuất ký tự. Các hàm này có thể dùng trong kiểu nhị phân cũng như văn bản.

13. putc và fputc dùng để ghi ký tự lên tệp.
14. getc và fgetc dùng để đọc ký tự từ tệp.

2.3. Các hàm nhập xuất theo kiểu văn bản. Các hàm này chỉ nên dùng theo kiểu văn bản.

15. fprintf dùng để ghi dữ liệu theo khuôn dạng lên tệp.
16. fscanf dùng để đọc dữ liệu từ tệp theo khuôn dạng.
17. fputs dùng để ghi một chuỗi ký tự lên tệp.
18. fgets dùng để đọc một dãy ký tự từ tệp.

2.4. Các hàm nhập xuất theo kiểu nhị phân. Các hàm này chỉ nên dùng theo kiểu nhị phân.

19. putw dùng để ghi một số nguyên (2 byte) lên tệp.
20. getw dùng để đọc một số nguyên (2 byte) từ tệp.
21. fwrite dùng để ghi một số mẫu tin lên tệp.
22. fread dùng để đọc một số mẫu tin từ tệp.

§3. ĐÓNG MỞ TỆP, XOÁ VÙNG ĐỆM VÀ KIỂM TRA LỖI

Các hàm trong mục này dùng chung cho cả hai kiểu nhị phân và văn bản.

3.1. Hàm fopen: Mở tệp.

+ Dạng hàm:

```
FILE *fopen(const char *tên_tệp, const char *kiểu);
```

+ Các đối:

Đối thứ nhất là tên tệp, đối thứ hai là kiểu truy nhập.

Kiểu có thể có các giá trị sau:

Kiểu	Ý nghĩa
"r" "rt"	Mở một tệp để đọc theo kiểu văn bản. Tệp cần tồn tại nếu không sẽ có lỗi.
"w" "wt"	Mở một tệp mới để ghi theo kiểu văn bản. Nếu tệp đã tồn tại nó bị xoá.
"a" "at"	Mở một tệp để ghi bổ xung theo kiểu văn bản. Nếu tệp chưa tồn tại thì tạo tệp mới.
"rb"	Mở một tệp để đọc theo kiểu nhị phân. Tệp cần tồn tại nếu không sẽ có lỗi.
"wb"	Mở một tệp mới để ghi theo kiểu nhị phân. Nếu tệp đã tồn tại nó bị xoá.
"ab"	Mở một tệp để ghi bổ xung theo kiểu nhị phân. Nếu tệp chưa tồn tại thì tạo tệp mới.
"r+" "r+t"	Mở một tệp để đọc/ghi theo kiểu văn bản. Tệp cần tồn tại nếu không sẽ có lỗi.
"w+" "w+t"	Mở một tệp mới để đọc/ghi theo kiểu văn bản. Nếu tệp đã tồn tại nó bị xoá.
"a+" "a+t"	Mở một tệp để đọc/ghi bổ xung theo kiểu văn bản. Nếu tệp chưa tồn tại thì tạo tệp mới.
"r+b"	Mở một tệp để đọc/ghi theo kiểu nhị phân. Tệp cần tồn tại nếu không sẽ có lỗi.
"w+b"	Mở một tệp mới để đọc/ghi theo kiểu nhị phân. Nếu tệp đã tồn tại nó bị xoá.
"a+b"	Mở một tệp để đọc/ghi bổ xung theo kiểu nhị phân. Nếu tệp chưa tồn tại thì tạo tệp mới.

+ **Công dụng:** Hàm dùng để mở tệp. Nếu thành công hàm cho con trỏ kiểu FILE ứng với tệp vừa mở. Các hàm cấp 2 sẽ làm việc với tệp thông qua con trỏ này. Nếu có lỗi hàm trả về giá trị NULL.

+ **Chú ý:** Trong các kiểu đọc/ghi, cần làm sạch vùng đệm trước khi chuyển từ đọc sang ghi hoặc từ ghi sang đọc. Các hàm fflush và hàm di chuyển đầu từ đều làm được chuyện này.

3.2. Hàm fclose: đóng tệp.

+ Dạng hàm:

```
int fclose(FILE *fp);
```

+ Đối:

fp là con trỏ tương ứng với tệp cần đóng.

+ Công dụng: Hàm dùng để đóng tệp. Nội dung đóng tệp gồm:

- Đẩy dữ liệu còn trong vùng đệm lên đĩa (khi đang ghi)
- Xoá vùng đệm (khi đang đọc)
- Giải phóng biến fp để nó có thể dùng cho tệp khác. Nếu thành công hàm cho giá trị 0, trái lại hàm cho EOF.

3.3. Hàm fcloseall: đóng các tệp đang mở.

+ Dạng hàm:

```
int fcloseall(void);
```

+ Công dụng: Hàm dùng để đóng tất cả các tệp đang mở. Nếu thành công hàm cho giá trị nguyên bằng số tệp đóng được, trái lại hàm cho EOF.

3.4. Hàm fflush: làm sạch vùng đệm.

+ Dạng hàm:

```
int fflush(FILE *fp);
```

+ Đối:

fp là con trỏ tệp.

+ Công dụng: Hàm dùng làm sạch vùng đệm của tệp fp. Nếu thành công hàm cho giá trị 0, trái lại hàm cho EOF.

3.5. Hàm fflushall: làm sạch vùng đệm.

+ Dạng hàm:

```
int fflushall(void);
```

+ Công dụng: Hàm dùng làm sạch vùng đệm của các tệp đang mở. Nếu thành công hàm cho giá trị nguyên bằng số tệp đang mở, trái lại hàm cho EOF.

3.6. Hàm ferror: kiểm tra lỗi.

+ Dạng hàm:

```
int ferror(FILE *fp);
```

+ Đối:

fp là con trỏ tệp.

+ Công dụng: Hàm dùng để kiểm tra lỗi thao tác trên tệp fp. Hàm cho giá trị 0 nếu không lỗi, trái lại hàm cho giá trị khác không.

3.7. Hàm perror: thông báo lỗi hệ thống.

+ Dạng hàm:

```
void perror(const char *s);
```

+ Đối:

s là con trỏ tới một chuỗi ký tự.

+ Công dụng: Hàm in chuỗi s và thông báo lỗi.

3.8. Hàm feof: kiểm tra cuối tệp.

+ Dạng hàm:

```
int feof(FILE *fp);
```

+ Đối: fp là con trỏ tệp.

+ Công dụng: Hàm dùng để kiểm tra cuối tệp. Hàm cho giá trị khác 0 nếu gặp cuối tệp khi đọc, trái lại hàm cho giá trị 0.

3.9. Hàm unlink: xoá tệp.

+ Dạng hàm:

```
int unlink(const char *tên_tệp);
```

+ Đối: là tên tệp cần xoá.

+ Công dụng: Hàm dùng để xoá một tệp trên đĩa. Nếu thành công hàm cho giá trị 0, trái lại hàm cho giá trị EOF.

3.10. Hàm remove: xoá tệp.

+ Dạng hàm:

```
remove(const char *tên_tệp);
```

+ Đối: là tên tệp cần xoá.

+ Công dụng: Hàm dùng để xoá một tệp trên đĩa. Nó hàm macro gọi tới unlink.

Ví dụ: Đoạn chương trình sau sẽ mở một tệp và kiểm tra lỗi.

```
FILE *fp; /* Biến con trỏ tệp */
/* Mở tệp so_lieu để đọc theo kiểu nhị phân.
   Nếu thành công con trỏ tệp so_lieu gán cho biến fp.
   Các hàm sẽ làm việc với tệp so_lieu thông qua fp.
   Vì vậy ta có nói tệp so_lieu hay tệp fp.
*/
fp = fopen("so_lieu", "rb");
/* Kiểm tra lỗi */
if (fp == NULL)
    perror("Lỗi khi mở tệp so_lieu: ");
```

§4. NHẬP XUẤT KÝ TỰ

Các hàm nhập xuất ký tự dùng được cả trong kiểu nhị phân và ký tự, nhưng tác dụng có những điểm khác nhau.

4.1. Các hàm putc và fputc

+ Dạng hàm:

```
int putc(int ch, FILE *fp);  
int fputc(int ch, FILE *fp);
```

+ **Đối:** ch là một giá trị nguyên, fp là con trỏ tệp.

+ **Công dụng:** Hàm ghi lên tệp fp một ký tự có mã bằng:

$m = ch \% 256$

trong đó ch được xem là số nguyên không dấu. Nếu thành công hàm cho mã ký tự được ghi, trái lại hàm cho EOF.

Ví dụ câu lệnh:

```
putc(-1, fp);
```

sẽ ghi lên tệp fp mã 255, vì dạng không dấu của -1 là 65535.

+ Ghi chú:

- Hai hàm trên có ý nghĩa như nhau.
- Trong kiểu văn bản, nếu $m=10$ thì hàm sẽ ghi lên tệp hai mã 13 và 10.

4.2. Các hàm getc và fgetc

+ Dạng hàm:

```
int getc(FILE *fp);  
int fgetc(FILE *fp);
```

+ **Đối:** fp là con trỏ tệp.

+ **Công dụng:** Hàm đọc một ký tự từ tệp fp. Nếu thành công hàm cho mã đọc được (có giá trị từ 0 đến 255). Nếu gặp cuối tệp hay có lỗi hàm cho EOF.

+ Ghi chú:

- Hai hàm trên có ý nghĩa như nhau.
- Trong kiểu văn bản, hàm đọc một lượt cả hai mã 13, 10 và trả về giá trị 10.
- Trong kiểu văn bản, khi gặp mã 26 thì hàm không trả về 26 mà trả về EOF.

4.3. Các ví dụ minh họa

Ví dụ 1: Chương trình sao tệp.

```
/*SAO TEP Dùng mode nhị phân và getc, putc */  
#include "stdio.h"  
main()  
{  
    int c;  
    char t1[14], t2[14];
```

```

FILE *f1,*f2;
printf("\n TEP NGUON: ");
gets(t1);
printf("\n TEP DICH: ");
gets(t2);
f1=fopen(t1,"rb");
if(f1==NULL)
{
    printf("\n TEP %s khong ton tai",t1);
    getch();
    exit(1);
}
f2=fopen(t2,"wb");
/* Việc sao tệp được thực hiện ở đoạn này */
while ((c=fgetc(f1))!=EOF)
    fputc(c,f2);
fclose(f1);
fclose(f2);
}

```

Chương trình thực hiện sao tệp theo thuật toán sau:

1. Đọc một ký tự của tệp f1, kết quả đặt vào biến c.

2. Nếu c bằng EOF thì kết thúc.

Nếu c khác EOF thì ghi c vào tệp f2 rồi trở lại 1.

Nhận xét 1. Nếu trong chương trình trên ta thay bằng kiểu nhập xuất văn bản, thì chỉ các byte đứng trước mã 26 đầu tiên của tệp f1 được sao sang tệp f2.

Nhận xét 2. Nếu dùng hàm feof và thuật toán:

1. Nếu feof(f1) khác không thì kết thúc, trái lại chuyển xuống 2.

2. Đọc một ký tự từ tệp f1, ghi lên tệp f2, rồi trở lại 1, thì nhận được đoạn chương trình:

```

while (!feof(f1))
    fputc(fgetc(f1), f2);

```

Nhưng đoạn chương trình sau không thật đúng. Tệp f2 sẽ dài hơn tệp f1 đúng một byte có giá trị 255. Lý do cũng dễ giải thích. Giả sử tệp f1 có đúng một ký tự mã 65, khi đó thuật toán sẽ diễn ra như sau:

1. Đầu từ đang trỏ vào ký tự A nên feof(f1) = 0, chuyển xuống 2.

2. Đọc ký tự A của f1 và ghi lên f2, trở lại 1.

3. Đầu đọc đặt ở cuối tệp f1 nhưng chưa có thao tác đọc nên feof(f1) vẫn bằng 0, chuyển xuống 2.

4. Đọc một ký tự của f1. Khi đó ta nhận được -1. Ghi -1 lên f2 thì mã 255 sẽ được ghi. Ngoài ra, do khi đọc từ f1 gặp phải cuối tệp nên lúc này feof(f1) khác 0.

Đến đây thuật toán kết thúc.

Ví dụ 2: Chương trình cho hiện lên màn hình các ký tự của tệp. Trên mỗi dòng đặt 10 ký tự. Mỗi ký tự sẽ được thể hiện bởi 2 thông số là mã và dạng ký tự. Sau khi cho hiện 24 dòng thì chương trình tạm dừng. Bấm phím bất kỳ để xem tiếp.

```
/*DOC TEP*/
#include "stdio.h"
#include "conio.h"
main()
{
    int c,i,k;
    char tep[14];
    FILE *fp;
    printf("\n TEP NGUON: "); gets(tep);
    fp=fopen(tep,"rb");
    if(fp==NULL)
    {
        printf("\n TEP %s không tồn tại",tep);
        getch(); exit(1);
    }
    i=0; k=0; clrscr();
    while ((c=fgetc(fp))!=EOF)
    {
        /* In mã */
        printf("%-4d",c);
        if (c<32) c=46;
        printf("%-4c",c); ++i; /* In ký tự */
        if(i==10)
        {
            i=0; ++k;
        }
        if(k==24)
        {
            getch(); k=0; clrscr();
        }
    }
    fclose(fp);
}
```

§5. CÁC HÀM NHẬP XUẤT THEO KIỂU VĂN BẢN

Các hàm trong mục này chỉ nên dùng theo kiểu văn bản.

5.1 Hàm fprintf: ghi dữ liệu theo khuôn dạng

+ **Dạng hàm:**

```
int fprintf(FILE *fp,const char *dk,...);
```

+ **Đối:**

fp là con trỏ tệp,

dk chứa địa chỉ của chuỗi điều khiển,

... là danh sách các đối mà giá trị của chúng cần ghi lên tệp.

Chuỗi điều khiển và danh sách đối có cùng ý nghĩa như trong hàm printf.

+ **Công dụng:** Giá trị các đối được ghi lên tệp fp theo khuôn dạng xác định trong chuỗi điều khiển dk. Nếu thành công, hàm trả về một giá trị nguyên bằng số byte ghi lên tệp. Khi có lỗi hàm cho EOF. Hàm làm việc giống như printf.

+ **Ví dụ:** Xét chương trình:

```
#include <stdio.h>
main()
{
    FILE *f;
    int i;
    f=fopen("text","wt");
    fprintf(f,"Cac dong");
    for(i=1;i<=2;++i)
        fprintf(f,"\nDong%2d",i);
    fclose(f);
}
```

Chương trình trên sẽ tạo ra tệp văn bản text gồm 3 dòng với nội dung như sau:

Cac dong

Dong 1 Dong 2

5.2. Hàm scanf: Đọc dữ liệu từ tệp theo khuôn dạng

Dạng hàm:

```
int fscanf(FILE *fp, const char *dk, ...);
```

+ Đối:

fp là con trỏ tệp,

dk chứa địa chỉ của chuỗi điều khiển,

... là danh sách các đối chứa kết quả đọc được từ tệp.

Chuỗi điều khiển và danh sách đối có cùng ý nghĩa như trong hàm scanf.

+ **Công dụng:** Đọc dữ liệu từ tệp fp, biến đổi theo khuôn dạng (đặc tả) trong dk và lưu kết quả vào các đối. Hàm làm việc giống như scanf. Hàm trả về một giá trị bằng số trường được đọc.

+ **Ví dụ 1:** Giả sử có tệp kiểu văn bản "da_giac.sl" chứa thông tin về một đa giác. Tệp gồm n+1 dòng với nội dung sau:

Dòng 1: n (số đỉnh)

Dòng 2: x1 y1 (toạ độ đỉnh 1)
 Dòng 3: x2 y2 (toạ độ đỉnh 2)
 ...
 Dòng n+1: xn yn (toạ độ đỉnh n)

Chương trình sau sẽ đọc số đỉnh và toạ độ các đỉnh từ tệp "da_giac.sl"

```
#include <stdio.h>
main()
{
    FILE *f; int i,n;
    float x[50],y[50];
    f=fopen("da_giac.sl","rt");
    fscanf(f,"%d",&n);
    for(i=1;i<=n;++i)
        fscanf(f,"%f%f",&x[i],&y[i]);
    fclose(f);
}
```

+ Ví dụ 2: Giả sử có một dãy số nguyên ghi trên tệp "sl.sl". Giữa hai số nguyên có ít nhất một khoảng trống hay các dấu xuống dòng. Yêu cầu đọc và in ra màn hình dãy số nói trên.

Ta phân biệt hai trường hợp:

1. Sau chữ số cuối cùng là mã 26 hay cuối tệp.
2. Sau chữ số cuối cùng có ít nhất một khoảng trống hay các dấu xuống dòng.

Dưới đây là hai chương trình ứng với hai trường hợp nêu trên.

/* Đọc dãy số theo trường hợp 1 */

```
#include <stdio.h>
main()
{
    FILE *f; int c;
    f=fopen("sl.sl","r");
    while(!feof(f))
    {
        fscanf(f,"%d",&c);
        printf("\n%d",c);
    }
    fclose(f);
    getch();
}
```

/* Đọc dãy số theo trường hợp 2 */

```
#include <stdio.h>
main()
{
    FILE *f; int c;
```

```

f=fopen("sl.sl","r");
while(1)
{
    fscanf(f,"%d",&c);
    if(feof(f)) break;
    printf("\n%d",c);
}
fclose(f);
getch();
}

```

5.3. Hàm fputs: ghi một chuỗi ký tự lên tệp.

+ **Dạng hàm:**

```
int fputs(const char *s, FILE *fp);
```

+ **Đối:**

s là con trỏ trỏ tới địa chỉ đầu của một chuỗi ký tự kết thúc bằng dấu '\0'.

fp là con trỏ tệp.

+ **Công dụng:** Ghi chuỗi s lên tệp fp (dấu '\0' không ghi lên tệp). Khi thành công, hàm trả về ký tự cuối cùng được ghi lên tệp. Khi có lỗi hàm cho EOF.

+ **Ví dụ:** Chương trình sau sẽ nhập các dòng ký tự từ bàn phím và ghi lên tệp "dong_vb".

```

/* ghi các dòng văn bản lên tệp */
#include <stdio.h>
#include <conio.h>
main()
{
    int i 0;
    char d[256];
    FILE *f;
    f=fopen("dong_vb", "w");
    clrscr();
    while(1)
    {
        i++;
        printf("\nDong %d (Bam Enter \
ket thuc): ", i);
        gets(d);
        if(d[0]=='\0')
            break;
        if(i>1)
            fputc(10, f);
            fputs(d, f);
    }
    fclose(f);
}

```

5.4. Hàm fgets: đọc một dãy ký tự từ tệp.

+ Dạng hàm:

```
char *fgets(char *s, int n, FILE *fp);
```

+ Đối:

s là con trỏ (kiểu char) trỏ tới một vùng nhớ đủ lớn để chứa chuỗi ký tự đọc từ tệp.

n là số nguyên xác định độ dài cực đại của dãy cần đọc.

fp là con trỏ tệp.

+ **Công dụng:** Đọc một dãy ký tự từ tệp fp chứa vào vùng nhớ s.

Việc đọc kết thúc khi:

- Hoặc đã đọc n-1 ký tự.

- Hoặc gặp dấu xuống dòng (Cặp mã 13 10). Khi đó mã 10 được đưa vào chuỗi kết quả.

- Hoặc kết thúc tệp.

Chuỗi kết quả sẽ được bổ sung thêm dấu hiệu kết thúc chuỗi '\0'.

Khi thành công, hàm trả địa chỉ vùng nhận kết quả. Khi có lỗi hoặc gặp cuối tệp, hàm cho giá trị NULL.

+ **Ví dụ:** Chương trình sau sẽ đọc các dòng ký tự trên tệp "dong_vb" (xem 5.3) và in ra màn hình.

```
/* Đọc các dòng ký tự
trên tệp văn bản */
#include <stdio.h>
#include <conio.h>
main()
{
    int i=0;
    char d[256];
    FILE *f;
    f=fopen("dong_vb", "r");
    clrscr();
    while(!feof(f))
    {
        ++i;
        fgets(d, 256, f);
        printf("Dong %d: %s", i, d);
    }
    fclose(f);
    getch();
}
```

§6. TẬP VĂN BẢN VÀ CÁC THIẾT BỊ CHUẨN

Có thể dùng các hàm nhập xuất văn bản (theo kiểu văn bản) trên các thiết bị chuẩn. C đã định nghĩa các tệp tin và con trỏ tệp ứng với các thiết bị chuẩn như sau:

Tệp	Con trỏ	Thiết bị
in	stdin	Thiết bị vào chuẩn (bàn phím)
out	stdout	Thiết bị ra chuẩn (màn hình)
err	stderr	Thiết bị lỗi chuẩn (màn hình)
prn	stdprn	Thiết bị in chuẩn (máy in)

Khi chương trình C bắt đầu làm việc thì các tệp này được tự động mở, vì vậy có thể dùng các con trỏ nêu trên để nhập xuất trên các thiết bị chuẩn. Dưới đây là một chương trình minh họa:

```
/* Chương trình minh họa cách dùng các con
trò tệp tin chuẩn */
#include "stdio.h"
#include "conio.h"
main()
{
    char ht[25];
    float diem;
    int ns;
    printf("\nHo ten: ");
    fgets(ht,25,stdin);
    printf("\ndiem va ns: ");
    fscanf(stdin,"%f%d",&diem,&ns);
    fputs(ht,stderr);
    fprintf(stdout,"Diem %f ns %d",diem,ns);
}
```

§7. CÁC HÀM NHẬP XUẤT THEO KIỂU NHỊ PHÂN

Các hàm trong mục này chỉ nên dùng theo kiểu nhị phân.

7.1. Hàm putw: ghi một số nguyên.

+ **Dạng hàm:**

```
int putw(int n, FILE *fp);
```

+ **Đối:**

n là giá trị nguyên,

fp là con trỏ tệp.

+ **Công dụng:** Ghi giá trị n lên tệp fp dưới dạng 2 byte. Nếu thành công, hàm trả về số nguyên được ghi. Khi có lỗi hàm trả về EOF.

7.2. Hàm getw: đọc một số nguyên.

+ **Dạng hàm:**

```
int getw(FILE *fp);
```

+ **Đối:**

fp là con trỏ tệp.

+ **Công dụng:** Đọc một số nguyên (2 byte) từ tệp fp. Nếu thành công, hàm trả về số nguyên đọc được. Khi có lỗi hoặc gặp cuối tệp, hàm trả về EOF.

+ **Ví dụ:** Chương trình sau minh họa cách dùng các hàm putw và getw. Đầu tiên ghi một dãy số nguyên lên tệp "so_ng", sau đó đọc các số nguyên từ tệp này và in ra màn hình.

```
/*   Chương trình ghi, đọc số nguyên
    minh họa cách dùng các hàm: putw và getw */
#include "stdio.h"
#include "conio.h"
main()
{
    FILE *f;
    int i;

    /* Ghi các số nguyên */
    f=fopen("so_ng", "wb");
    for(i=1000; i<=1010; ++i)
        putw(i, f);
    fclose(f);

    /* Đọc các số nguyên */
    clrscr();
    f=fopen("so_ng", "rb");
    while((i=getw(f))!=EOF)
        printf("\n%d", i);
    fclose(f);
}
```

7.3. Hàm fwrite:

Ghi các mẫu tin lên tệp.

+ **Dạng hàm:**

```
int fwrite(void *ptr, int size, int n, FILE *fp);
```

+ **Đối:**

ptr là con trỏ trỏ tới vùng nhớ chứa dữ liệu cần ghi,

size là kích thước của mẫu tin theo byte,
n là số mẫu tin cần ghi,
fp là con trỏ tệp.

+ **Công dụng:** Ghi n mẫu tin kích thước size byte từ vùng nhớ ptr lên tệp fp. Hàm trả về một giá trị bằng số mẫu tin thực sự ghi được.

7.4. Hàm fread: đọc các mẫu tin từ tệp.

+ **Dạng hàm:**

```
int fread(void *ptr, int size, int n, FILE *fp);
```

+ **Đối:**

ptr là con trỏ trỏ tới vùng nhớ chứa dữ liệu đọc được,
size là kích thước của mẫu tin theo byte,
n là số mẫu tin cần đọc,
fp là con trỏ tệp.

+ **Công dụng:** Đọc n mẫu tin kích thước size byte từ tệp fp chứa vào vùng nhớ ptr. Hàm trả về một giá trị bằng số mẫu tin thực sự đọc được.

Các ví dụ minh họa fwrite, fread.

Các hàm fread, fwrite thường dùng để đọc ghi các đối tượng có cùng độ lớn như cấu trúc, số thực,... . Dưới đây là 3 ví dụ minh họa.

Ví dụ 1: Chương trình dưới đây thực hiện việc sao tệp bằng các hàm fread và fwrite.

```
/* SAO TEP dung fread, fwrite */
#include "stdio.h"
main()
{
    int n;
    char t1[14], t2[14], c[1000];
    FILE *f1, *f2;
    printf("\n TEP NGUON: ");
    gets(t1);
    printf("\n TEP DICH: ");
    gets(t2);
    f1=fopen(t1, "rb");
    if(f1==NULL)
    {
        printf("\n TEP %s khong ton tai", t1);
        getch();
        exit(1);
    }
    f2=fopen(t2, "wb");
```

```

while ( (n=fread(c,1,1000,f1))>0 )
    fwrite(c,1,n,f2);
fclose(f1);
fclose(f2);
}

```

Ví dụ 2: Chương trình dưới đây minh họa cách dùng các hàm fread và fwrite để ghi đọc một dãy n phần tử (nguyên, thực, cấu trúc).

```

/* Dùng fread và fwrite để ghi đọc một dãy n phần tử */
#include "stdio.h"
#include "conio.h"
typedef struct
{
    char ht[25];
    int ns;
} hs;
/* Khai báo và khởi đầu mảng cấu trúc ds */
hs ds[50]={
    {"",0},
    {"Pham Thu Huong",1976},
    {"Nguyen Viet Dung",1978},
    {"Thanh Tam",1974}
};

main()
{
    FILE *fp;
    float x[50]; int a[50];
    int i;
    /* Gán giá trị cho a[i] và x[i] */
    for(i=1;i<=10;++i)
        a[i]=100*i;
    for(i=1;i<=5;++i)
        x[i]=1.0/i;
    /* Ghi mảng a, x và ds từ
       phần tử có chỉ số 1 */
    fp=fopen("mang.sl","wb");
    fwrite(a+1,sizeof(int),10,fp);
    fwrite(x+1,sizeof(float),5,fp);
    fwrite(ds+1,sizeof(hs),3,fp);
    fclose(fp);
    /* Đọc từ tệp và đưa ra mảng từ
       phần tử chỉ số 11 */
    fp=fopen("mang.sl","rb");
    fread(a+11,sizeof(int),10,fp);
    fread(x+11,sizeof(float),5,fp);
    fread(ds+11,sizeof(hs),3,fp);
    fclose(fp);
    /* In */
    for(i=11;i<=20;++i)
        printf("\n%d", a[i]);
}

```

```

        for(i=11;i<=15;++i)
            printf("\n%f", x[i]);
        for(i=11; i<=13; ++i)
            printf("\n%s %d", ds[i].ht,ds[i].ns);
    }

```

Ví dụ 3: Chương trình dưới đây minh họa cách dùng các hàm fread và fwrite để ghi đọc các cấu trúc (chưa biết rõ số lượng là bao nhiêu)

```

/* đọc/ghi cấu trúc bằng fread fwrite */
#include "stdio.h"
#include "conio.h"
typedef struct
{
    char ht[25];
    int ns;
} hs;
main()
{
    FILE *fp;
    hs h;
    /* Nhập số liệu từ bàn phím rồi ghi lên tệp */
    fp=fopen("hs.sl", "wb");
    while(1)
    {
        printf("\nHọ ten (Bấm Enter kết thúc): ");
        gets(h.ht);
        if(h.ht[0]=='\0')
            break;
        printf("\nNam sinh ");
        scanf("%d%c",&h.ns);
        fwrite(&h,sizeof(hs),1,fp);
    }
    fclose(fp);
    /* Đọc dữ liệu từ tệp và in ra màn hình */
    fp=fopen("hs.sl", "rb");
    while( fread(&h,sizeof(hs),1,fp)>0)
        printf("\n%s %d",h.ht,h.ns);
    fclose(fp);
}

```

§8. NHẬP XUẤT NGẪU NHIÊN

Mỗi tệp khi đang mở có một con trỏ chỉ vị (File position locator) dùng để xác định vị trí đọc ghi trên tệp. Khi mở tệp tin để đọc hay ghi, con trỏ chỉ vị luôn luôn ở đầu tệp tin (byte 0). Nhưng nếu tệp được mở theo mode "a" thì con trỏ chỉ vị ở cuối tệp để ghi thêm dữ liệu vào tệp này. Việc nhập xuất dữ liệu được bắt đầu từ vị trí hiện tại của con trỏ chỉ vị và sau khi hoàn thành thì con trỏ này dịch chuyển một số byte chính bằng số byte đã đọc hay ghi.

Như vậy việc nhập xuất được tiến hành tuần tự theo chiều từ đầu đến cuối tệp tin.

Trong mục này sẽ nói cách di chuyển con trỏ chỉ vị đến vị trí mong muốn, nhờ đó ta có thể nhập xuất tại bất kỳ chỗ nào trên tệp tin. Cách làm này gọi là nhập xuất ngẫu nhiên (Random Access I/O) rất tiện lợi cho việc sửa chữa, bổ sung dữ liệu trực tiếp trên tệp.

8.1. Hàm **rewind**: chuyển con trỏ chỉ vị về đầu tệp.

+ **Dạng hàm:**

```
void rewind(FILE *fp);
```

+ **Đối:**

fp là con trỏ tệp.

+ **Công dụng:** Chuyển con trỏ chỉ vị của tệp fp về đầu tệp. Khi đó việc nhập xuất trên tệp fp được thực hiện từ đầu tệp.

8.2. Hàm **fseek**: di chuyển con trỏ chỉ vị đến vị trí mong muốn.

+ **Dạng hàm:**

```
int fseek(FILE *fp, long sb, int xp);
```

+ **Đối:**

fp là con trỏ tệp.

sb là số byte cần di chuyển,

xp cho biết vị trí xuất phát mà việc dịch chuyển được bắt đầu từ đây. xp có thể nhận các giá trị sau:

xp = SEEK_SET hay 0: Xuất phát từ đầu tệp.

xp = SEEK_CUR hay 1: Xuất phát từ vị trí hiện tại của con trỏ chỉ vị.

xp = SEEK_END hay 2: Xuất phát từ cuối tệp.

+ **Công dụng:** Hàm di chuyển con trỏ chỉ vị của tệp fp từ vị trí xác định bởi xp qua một số byte bằng giá trị tuyệt đối của sb. Chiều di chuyển là về cuối tệp nếu sb dương, trái lại sẽ di chuyển về phía đầu tệp.

Khi thành công hàm trả về giá trị 0. Khi có lỗi hàm trả về giá trị khác không.

+ **Chú ý:** Không nên dùng fseek trên kiểu văn bản, vì sự chuyển đổi ký tự sẽ làm cho việc định vị thiếu chính xác.

8.3. Hàm **ftell**: cho biết vị trí hiện tại của con trỏ chỉ vị.

+ **Dạng hàm:**

```
long ftell(FILE *fp);
```

+ **Đổi:**

fp là con trỏ tệp.

+ **Công dụng:** Khi thành công hàm cho biết vị trí hiện tại của con trỏ chỉ vị (byte thứ mấy trên tệp fp). Số thứ tự của byte được tính từ 0. Khi có lỗi hàm trả về -1L.

+ **Minh họa:** Các ví dụ sau sẽ giải thích thêm cách làm việc của hàm fseek và ftell.

Giả sử tệp fp có 3 ký tự lần lượt là A, B và C.

1. Sau câu lệnh

```
fseek(fp, 0, SEEK_END);
```

thì con trỏ chỉ vị ở cuối tệp và ftell(fp) = 3.

2. Sau câu lệnh

```
fseek(fp, -1, SEEK_END);
```

thì con trỏ chỉ vị đặt tại C và ftell(fp) = 2.

3. Sau câu lệnh

```
fseek(fp, 0, SEEK_SET);
```

thì con trỏ chỉ vị đặt tại A và ftell(fp) = 0.

Hai ví dụ minh họa.

+ **Ví dụ 1:** Chương trình dưới đây dùng các hàm fseek và ftell để xác định độ dài của tệp bất kỳ.

```
/* Dung fseek, ftell xác định do dai tep tin */
#include "stdio.h"
#include "conio.h"
main()
{
    FILE *fp;
    long n;
    char tep[14];
    clrscr();
    printf("\nTen tep: "); gets(tep);
    fp = fopen(tep, "rb");
    if (fp == NULL)
    {
        printf("\nTen %s khong ton tai ", tep);
        exit(1);
    }
    fseek(fp, 0, SEEK_END);
    n = ftell(fp);
    fclose(fp);
    printf("\nDo dai cua tep %s la %ld byte", tep, n);
}
```

+ **Ví dụ 2:** Chương trình dưới đây dùng các hàm fseek, ftell, fread và fwrite để tổ chức một hệ quản trị dữ liệu với 4 chức năng: tạo mới, bổ sung, sửa chữa và xem dữ liệu.

```

/*
Dùng các mode wb, ab, rb, r+b và các hàm f
write, fread, fseek, ftell để tổ chức cơ sở dữ liệu
*/
#include "conio.h"
#include "stdio.h"
void taomoi(void);
void bosung(void);
void suachua(void);
void xem(void);
typedef struct
{
    char ht[25];
    int bl;
    float nc;
} person;
long size    sizeof(person);
main()
{
    int sg;
    while (1)
    {
        clrscr();
        printf("\n1.Tao moi");
        printf("\n2.Bo sung");
        printf("\n3.Sua chua");
        printf("\n4.Xem du lieu");
        printf("\n5.Ket thuc");
        sg=getch();
        if(sg=='1')
            taomoi();
        else if (sg=='2')
            bosung();
        else if (sg=='3')
            suachua();
        else if (sg=='4')
            xem();
        else
        {
            clrscr();
            exit(1);
        }
    }
}
void taomoi(void)
{
    person cb;
    FILE *fp;
    int n=0;
    fp=fopen("dscb.sl", "wb");
    while(1)

```

```

        {
            clrscr();
            printf("\nNguoi thu %d\n\n",n+1);
            printf("\nlo ten (Bam Enter ket thuc): ");
            gets(cb.ht);
            if(cb.ht[0]==0) break;
            printf("\nBac luong: ");
            scanf("%d",&cb.bl);
            printf("\nNgay cong: ");
            scanf("%f%c",&cb.nc);
            ++n;
            fwrite(&cb,size,1,fp);
        }
    fclose(fp);
}

void bosung(void)
{
    person cb;
    FILE *fp;
    long n;
    fp=fopen("dscb.sl","ab");
    /* Xac dinh n = so ban ghi */
    fseek(fp,0,SEEK_END);
    n=ftell(fp)/size;
    while(1)
    {
        clrscr();
        printf("\nNguoi thu %ld\n\n",n+1);
        printf("\nlo ten (Bam Enter ket thuc): ");
        gets(cb.ht);
        if(cb.ht[0]==0) break;
        printf("\nBac luong: ");
        scanf("%d",&cb.bl);
        printf("\nNgay cong: ");
        scanf("%f%c",&cb.nc);
        ++n;
        fwrite(&cb,size,1,fp);
    }
    fclose(fp);
}

void suachua(void)
{
    person cb;
    FILE *fp; long i,n;
    if( (fp=fopen("dscb.sl","r+b")) ==NULL)
    {
        printf("\nChua co du lieu");
        return;
    }
    /* Xac dinh n = so ban ghi */
    fseek(fp,0,SEEK_END);
    n =ftell(fp)/size;
    while(1)
    {

```

```

clrscr();
printf("\nSua nguoi thu (n<=%ld)",n);
scanf("%ld%c",&i);
if(i>0&&i<=n)
{
    fseek(fp,(i-1)*size,SEEK_SET);
    fread(&cb,size,1,fp);
    printf("\nSo lieu hien tai:");
    printf("\nHo ten: %s",cb.ht);
    printf("\nBac luong: %d",cb.bl);
    printf("\nNgay cong: %0.2f",cb.nc);
    printf("\n\nSua la:");
    printf("\nHo ten (Bam Enter \
ket thuc): ");
    gets(cb.ht);
    if(cb.ht[0]==0) break;
    printf("\nBac luong: ");
    scanf("%d",&cb.bl);
    printf("\nNgay cong: ");
    scanf("%f%c",&cb.nc);
    printf("\nCo Ghi sua ? - C/K");
    if(toupper(getch())=='C')
    {
        fseek(fp,m,SEEK_SET);
        fwrite(&cb,size,1,fp);
    }
}
fclose(fp);
}

void xem(void)
{
    person cb;
    FILE *fp;
    int i 0;
    if( (fp=fopen("dsch.sl","rb"))==NULL)
    {
        printf("\nChua co du lieu");
        return;
    }
    clrscr();
    while( fread(&cb,size,1,fp)>0 )
    {
        ++i;
        printf("\n\nNguoi %d",i);
        printf("\nHo ten: %s",cb.ht);
        printf("\nBac luong: %d",cb.bl);
        printf("\nNgay cong: %0.2f",cb.nc);
    }
    fclose(fp);
    getch();
}

```

§9. TẠO TỆP, ĐÓNG MỞ TỆP VÀ KIỂM TRA LỖI

9.1. Hàm creat: Xây dựng tệp mới.

+ Dạng hàm:

```
int creat(const char *fname, int amode);
```

+ Đối:

Đối thứ nhất là tên tệp,

Đối thứ hai là thuộc tính tệp. Các giá trị thường dùng của amode (định nghĩa trong sys/stat.h):

S_IREAD Tệp chỉ đọc. Sau này không thể xoá, sửa, bổ sung.

S_IWRITE Tệp để ghi. Sau này có thể xoá, sửa, bổ sung.

+ Công dụng: Xây dựng tệp mới có tên cho bởi đối thứ nhất và có thuộc tính cho bởi amode. Trong trường hợp tệp đã tồn tại:

- Nếu tệp để ghi, thì nó bị xoá,

- Nếu tệp chỉ đọc, thì bị lỗi.

Khi có lỗi, hàm trả về -1.

Khi thành công, hàm trả về số hiệu tệp (handle). Số hiệu này có thể dùng trong hàm write để ghi thông tin lên tệp (ngay cả khi amode = S_IREAD). Kiểu ghi được xác định bởi biến _fmode (định nghĩa trong fcntl.h và stdlib.h). Giá trị mặc định của _fmode là O_TEXT (kiểu văn bản). Muốn ghi theo kiểu nhị phân thì ta cần xác định lại _fmode (trước khi dùng hàm creat) bằng câu lệnh: _fmode = O_BINARY;

+ Ví dụ về creat: Chương trình sau dùng creat để tạo tệp mới và ghi lên đó 5 số nguyên. Tệp có thuộc tính chỉ đọc nên nó được bảo vệ. Sau này nếu muốn sửa, ghi bổ sung hay xoá ta cần thay đổi thuộc tính của nó bằng hàm chmod hay _chmod.

```
#include <io.h>
#include <fcntl.h>
#include <sys/stat.h>
int a[5]={10,2,10,4,5};
main()
{
    int fd;
    /* Tạo tệp mới sl.sl để ghi theo
    kiểu nhị phân */
    _fmode = O_BINARY;
    fd=creat("sl.sl",S_IREAD);
    /* Ghi 5 số nguyên lên tệp */
    write(fd,a,10);
    close(fd);      /* Đóng tệp */
}
```

9.2. Hàm `_creat`: Xây dựng tệp mới.

+ **Dạng hàm:**

```
int _creat(const char *fname, int attrib);
```

+ **Đối:**

Đối thứ nhất là tên tệp,

Đối thứ hai là thuộc tính tệp. Các giá trị thường dùng của attrib (định nghĩa trong dos.h):

FA_RDONLY Tệp chỉ đọc, không thể xóa hoặc sửa chữa.

FA_ARCH Tệp để ghi.

FA_HIDDEN Tệp ẩn, không hiện trong lệnh DIR của DOS.

Chú ý: Giá trị của attrib có thể là một tổ hợp (có nghĩa) của các giá trị trên, ví dụ nếu

```
attrib = FA_HIDDEN | FA_ARCH
```

thì thuộc tính tệp là: ẩn và có thể ghi.

+ **Công dụng:**

Xây dựng tệp mới có tên cho bởi đối thứ nhất và có thuộc tính cho bởi attrib. Trong trường hợp tệp đã tồn tại:

- Nếu tệp để ghi, thì nó bị xóa,

- Nếu tệp chỉ đọc, thì bị lỗi.

Khi có lỗi, hàm trả về -1.

Khi thành công, hàm trả về số hiệu tệp (handle). Số hiệu này có thể dùng trong hàm write để ghi thông tin lên tệp (ngay cả khi attrib FA_RDONLY). Kiểu ghi luôn luôn là nhị phân.

+ **Ví dụ về `_creat`:** Chương trình sau dùng `_creat` để tạo tệp mới và ghi lên đó 5 số nguyên. Tệp có thuộc tính để ghi nên sau này có thể sửa, ghi bổ sung hay xóa.

```
#include <io.h>
#include <dos.h>
int a[5]={10,2,10,4,5};
main()
{
    int fd;
    /* Tạo tệp mới sl.sl để ghi
       theo kiểu nhị phân */
    fd=_creat("sl.sl",FA_ARCH);
    /* Ghi 5 số nguyên lên tệp */
    write(fd,a,10);
    close(fd); /* Đóng tệp */
}
```

9.3. Hàm open

Mở tệp đã có hoặc xây dựng tệp mới để đọc ghi

+ **Dạng hàm:**

```
int open(const char *fname,  
         int access, [, unsigned amode]);
```

+ **Đối:**

- **Đối thứ nhất fname** xác định tên tệp cần mở.
- **Đối thứ hai access** xác định quyền và kiểu truy nhập, nó có thể nhận từ các giá trị sau (định nghĩa trong `fcntl.h`)

Giá trị	Ý nghĩa
O_RDONLY	Đọc
O_WRONLY	Ghi
O_RDWR	Đọc/ghi
O_APPEND	Ghi bổ sung
O_TRUNC	Xoá tệp nếu nó tồn tại (giữ nguyên thuộc tính)
O_TEXT	Kiểu nhập xuất văn bản
O_BINARY	Kiểu nhập xuất nhị phân
O_CREAT	Nếu tệp chưa có thì nó được tạo. (Khi đó cần đưa thêm đối thứ ba).

Chú ý 1: Đối thứ hai access có thể là một tổ hợp có nghĩa bất kỳ của các giá trị nêu trên (dùng phép |), ví dụ

O_WRONLY|O_TRUNC|O_BINARY

Chú ý 2: Nếu trong đối access không mô tả rõ O_TEXT hay O_BINARY thì kiểu truy nhập được xác định theo giá trị của biến `_fmode` (xem hàm `creat`)

- Đối thứ ba amode (xem hàm `creat`) dùng để xác định thuộc tính của tệp mới được tạo, nó thường nhận hai giá trị sau:

S_IREAD Tệp chỉ đọc, không thể xoá hoặc sửa chữa.

S_IWRITE Tệp có thể ghi.

+ **Công dụng:**

1. Không có O_CREAT: Mở một tệp đã tồn tại để truy nhập theo cách xác định bởi access. Lỗi xảy ra khi:

- Tập không tồn tại.
- Mở để ghi một tập chỉ đọc.

2. Có O_CREAT: Nếu tập chưa có thì nó được tạo. Lỗi xảy ra khi:

- Mở để ghi một tập chỉ đọc.

+ Giá trị hàm:

Khi thành công hàm trả về số hiệu tệp, nó sẽ dùng trong các hàm để truy nhập tới tệp. Khi có lỗi hàm cho giá trị -1.

+ Ví dụ về open:

Câu lệnh:

```
fd=open("sl.sl",O_WRONLY|O_BINARY|O_TRUNC|O_CREAT,  
S_IWRITE);
```

sẽ mở tệp sl.sl để ghi theo kiểu nhị phân. Nếu tệp đã tồn tại thì nó bị xoá. Nếu tệp chưa có thì nó được xây dựng.

Câu lệnh:

```
fd=open("sl.sl",O_RDONLY|O_BINARY);
```

sẽ mở tệp đã tồn tại sl.sl để đọc theo kiểu nhị phân.

9.4. Hàm _open: Mở tệp đã tồn tại để đọc ghi.

+ Dạng hàm:

```
int open(const char *fname, int oflag);
```

+ Đối:

- Đối thứ nhất fname xác định tên tệp cần mở,
- Đối thứ hai oflags xác định quyền và kiểu truy nhập, nó có ý nghĩa như đối access trong hàm open, ngoại trừ 2 điểm sau: kiểu mặc định là O_BINARY (chứ không phải O_TEXT) và phương án O_CREAT không có tác dụng tạo tệp mới.

+ Công dụng: Mở một tệp đã tồn tại để truy nhập theo cách xác định bởi oflags. Lỗi xảy ra khi:

- Tệp không tồn tại.
- Mở để ghi một tệp chỉ đọc.

+ Giá trị hàm:

Khi thành công hàm trả về số hiệu tệp, nó sẽ dùng trong các hàm để truy nhập tới tệp. Khi có lỗi hàm cho giá trị -1.

+ Ví dụ về open:

Câu lệnh:

```
fd= open("sl.sl",O_WRONLY|O_TRUNC);
```

sẽ mở và xoá tệp sl.sl (đã tồn tại) để ghi lại từ đầu (theo kiểu nhị phân).

Câu lệnh:

```
fd=_open("sl.sl",O_RDONLY);
```

sẽ mở tệp đã tồn tại sl.sl để đọc theo kiểu nhị phân.

9.5. Các hàm close và _close: đóng tệp.

+ Dạng hàm:

```
int close(int fd);  
int _close(int fd);
```

+ Đối:

fd là số hiệu tệp nhận được trong các hàm creat, _creat open và _open.

+ **Công dụng:** Đóng tệp có số hiệu fd. Số hiệu fd được giải phóng và nó có thể dùng cho tệp khác.

Khi thành công hàm trả về số 0. Khi có lỗi hàm cho -1.

9.6. Hàm chmod: thay đổi thuộc tính tệp.

+ Dạng hàm:

```
int chmod(const char *fname, int amode);
```

+ Đối:

Đối fname xác định tên tệp cần đổi thuộc tính.

Đối amode cho thuộc tính mới của tệp. Các giá trị của amode ở đây giống như amode trong hàm creat:

S IREAD Tệp chỉ đọc.

S WRITE Tệp để ghi.

+ **Công dụng:** Đổi thuộc tính tệp fname theo amode.

Hàm cho giá trị 0 nếu thành công và -1 nếu có lỗi.

9.7. Hàm chmod: Nhận hay đổi thuộc tính tệp theo kiểu DOS.

+ Dạng hàm:

```
int _chmod(const char *fname,  
            int func [,int attrib]);
```

+ Đối:

Đối fname xác định tên tệp cần nhận hay đổi thuộc tính.

Nếu func = 0, hàm cho biết thuộc tính của tệp, và không cần dùng đối thứ ba.

Nếu func = 1, thay đổi thuộc tính, dùng đối attrib để xác định thuộc tính mới. Các giá trị của attrib ở đây giống như attrib trong _creat:

FA_RDONLY Tập chỉ đọc,
FA_ARCH Tập để ghi.
FA_HIDDEN Tập ẩn.

+ **Công dụng:** Cho biết thuộc tính của tệp fname (func=0) hoặc đổi thuộc tính tệp theo attrib (func=1).

Hàm cho giá trị 0 nếu thành công và -1 nếu có lỗi.

9.8. Hàm perror: thông báo lỗi hệ thống.

(Hàm này đã trình bày trong §3.7)

+ **Dạng hàm:**

```
#include <stdio.h>
void perror(const char *s);
```

+ **Đối:**

s là con trỏ tới một chuỗi ký tự.

+ **Công dụng:** Hàm in chuỗi s và thông báo lỗi.

§10. CÁC HÀM write, read VÀ lseek

Để nhập xuất dữ liệu trên tệp, hệ thống cấp 1 cung cấp 3 hàm: write, read và lseek, chúng làm việc giống như các hàm fwrite, fread và fseek. Các hàm này nên dùng với kiểu nhập xuất nhị phân.

10.1. Hàm write: ghi một dãy các byte lên tệp.

+ **Dạng hàm:**

```
unsigned write(int fd, void *ptr, unsigned n);
```

+ **Đối:**

fd là số hiệu tệp,

ptr là con trỏ tới vùng nhớ chứa dữ liệu,

n là số byte.

+ **Công dụng:** Ghi n byte từ vùng nhớ ptr lên tệp có số hiệu fd. Khi thành công hàm trả về một số bằng số byte ghi được.

Khi có lỗi hàm trả về -1.

+ **Nhận xét:** Khi giá trị trả về của hàm khác n là có lỗi.

10.2. Hàm read: đọc một dãy các byte từ tệp.

+ **Dạng hàm:**

```
unsigned read(int fd, void *ptr, unsigned n);
```

+ **Đối:**

fd là số hiệu tệp.

ptr là con trỏ trỏ tới vùng nhớ nhận dữ liệu đọc được,

n là số byte.

+ Công dụng:

Đọc n byte từ tệp có số hiệu fd chứa vào vùng nhớ ptr. Khi thành công hàm trả về một số bằng số byte đọc được. Khi có lỗi hàm trả về -1.

+ Nhận xét:

Khi gặp cuối tệp thì giá trị trả về của hàm có thể nhỏ hơn n.

10.3. Hàm lseek: di chuyển con trỏ chỉ vị.

+ Dạng hàm:

```
long lseek(int fd, long sb, int xp);
```

+ Đối:

fd là số hiệu tệp,

sb là số byte cần di chuyển,

xp cho biết vị trí xuất phát mà việc dịch chuyển được bắt đầu từ đây. xp có thể nhận các giá trị sau:

xp = SEEK_SET hay 0: Xuất phát từ đầu tệp.

xp = SEEK_CUR hay 1: Xuất phát từ vị trí hiện tại của con trỏ chỉ vị.

xp = SEEK_END hay 2: Xuất phát từ cuối tệp.

+ Công dụng:

Hàm di chuyển con trỏ chỉ vị của tệp fp từ vị trí xác định bởi xp qua một số byte bằng giá trị tuyệt đối của sb. Chiều di chuyển là về cuối tệp nếu sb dương, trái lại sẽ di chuyển về phía đầu tệp.

Khi thành công hàm trả về một giá trị kiểu long bằng vị trí mới của con trỏ chỉ vị (số thứ tự của byte mà con trỏ chỉ vị trỏ tới). Khi có lỗi hàm trả về -1L.

+ Chú ý:

Không nên dùng lseek trên kiểu văn bản, vì sự chuyển đổi ký tự sẽ làm cho việc định vị thiếu chính xác.

+ Ví dụ:

Để xác định độ dài của tệp có số hiệu fd ta có thể dùng hàm lseek như sau:

```
long do_dai;  
do_dai = lseek(fd, 0, 2);
```

CÁC DÒNG TIN (STREAM)

C đã cung cấp một thư viện các hàm nhập xuất như `printf`, `scanf`, `gets`, `getch()`, `puts`, `push()`, `sprintf`, `fscanf`, `fopen`, `fwrite`, `fread`,... Các hàm này làm việc khá hiệu quả nhưng không thích ứng với cách tổ chức chương trình hướng đối tượng.

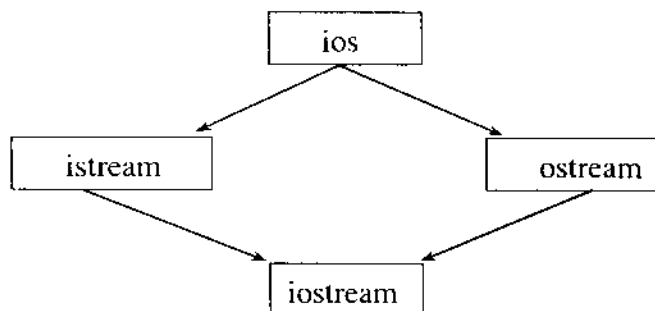
C++ sử dụng khái niệm dòng tin (stream) và đưa ra các lớp dòng tin để tổ chức việc nhập xuất. Dòng tin có thể xem như một dãy các byte. Thao tác nhập là lấy (đọc) các byte từ dòng tin (khi đó gọi là dòng nhập - input) vào bộ nhớ. Thao tác xuất là đưa các byte từ bộ nhớ ra dòng tin (khi đó gọi là dòng xuất - output). Các thao tác này là độc lập thiết bị. Để thực hiện việc nhập, xuất lên một thiết bị cụ thể, chúng ta chỉ cần gắn dòng tin với thiết bị này.

§1. CÁC LỚP STREAM

Có 4 lớp quan trọng cần nhớ là:

- + Lớp cơ sở ios
- + Từ lớp ios dẫn xuất đến 2 lớp istream và ostream
- + Hai lớp istream và ostream lại dẫn xuất tới lớp iostream

Sơ đồ kế thừa giữa các lớp như sau:



Lớp ios

- + Thuộc tính của lớp: Trong lớp ios định nghĩa các thuộc tính được sử dụng làm các cờ định dạng cho việc nhập xuất và các cờ kiểm tra lỗi (xem bên dưới).

+ Các phương thức: Lớp ios cung cấp một số phương thức phục vụ việc định dạng dữ liệu nhập xuất, kiểm tra lỗi (xem bên dưới).

Lớp istream

Lớp này cung cấp toán tử nhập >> và nhiều phương thức nhập khác (xem bên dưới) như các phương thức: get, getline, read, ignore, peek, seekg, tellg,...

Lớp ostream

Lớp này cung cấp toán tử xuất << và nhiều phương thức xuất khác (xem bên dưới) như các phương thức: put, write, flush, seekp, tellp,...

Lớp iostream

Lớp này thừa kế các phương thức nhập xuất của các lớp istream và ostream.

§2. DÒNG CİN VÀ TOÁN TỬ NHẬP

Dòng cin là một đối tượng kiểu istream đã định nghĩa trong C⁺⁺. Đó là dòng vào (input) chuẩn gắn với bàn phím (tương tự như stdin của C). Các thao tác nhập trên dòng cin đồng nghĩa với nhập dữ liệu từ bàn phím.

Do cin là một đối tượng của lớp istream nên với cin chúng ta có thể sử dụng toán tử nhập >> và các phương thức nhập của các lớp ios và istream.

Cách dùng toán tử nhập để đọc dữ liệu từ dòng cin như sau:

```
cin >> Tham_số ;
```

Trong đó Tham_số có thể là:

- Biến hoặc phần tử mảng nguyên để nhận một số nguyên
- Biến hoặc phần tử mảng thực để nhận một số thực
- Biến hoặc phần tử mảng ký tự để nhận một ký tự
- Con trỏ ký tự để nhận một dãy các ký tự khác trống

Chú ý: Các toán tử nhập có thể viết nối đuôi để nhập nhiều giá trị trên một dòng lệnh như sau:

```
cin >> Tham_số_1 >> Tham_số_2 >> ... >> Tham_số_k ;
```

Cách thức nhập như sau: Bỏ qua các ký tự trắng (dấu cách, dấu tab, dấu chuyển dòng) đứng trước nếu có và sau đó đọc vào các ký tự tương ứng với kiểu yêu cầu. Cụ thể đối với từng kiểu như sau:

Khi nhập số nguyên sẽ bỏ qua các ký tự trắng đứng trước nếu có, sau đó bắt đầu nhận các ký tự biểu thị số nguyên. Việc nhập kết thúc khi gặp một ký tự trắng hoặc một ký tự không thể hiểu là thành phần của số nguyên. Ví

dụ nếu trên dòng vào (gõ từ bàn phím) chứa các ký tự <space> <space> 123X2 và Tham_số (bên phải cin) là biến nguyên n thì n sẽ nhận giá trị 123. Con trỏ nhập sẽ dừng tại ký tự X.

Phép nhập một số thực cũng tiến hành tương tự: Bỏ qua các khoảng trắng đứng trước nếu có, sau đó bắt đầu nhận các ký tự biểu thị số Thực. Việc nhập kết thúc khi gặp một ký tự trắng hoặc một ký tự không thể hiểu là thành phần của số thực.

Phép nhập một ký tự cũng vậy: Bỏ qua các khoảng trắng đứng trước nếu có, sau đó nhận một ký tự khác ký tự trắng. Ví dụ nếu gõ <space><space>XY thì ký tự X được nhận và con trỏ nhập dừng tại ký tự Y.

Phép nhập một dãy ký tự: Bỏ qua các khoảng trắng đứng trước nếu có, sau đó bắt đầu nhận từ một ký tự khác ký tự trắng. Việc nhập kết thúc khi gặp một ký tự trắng.

Ví dụ 1: Xét đoạn chương trình:

```
char ten[10], que[12];
char ch;
int n;
float x;
cin >> n >> x >> ch >> ten >> que ;
```

Nếu gõ các ký tự:

```
123<s>3.14<s><s>ZHONG<s>HAI<s>PHONG<Enter>
```

(để cho gọn sẽ ký hiệu <s> là <space>)

thì kết quả nhập như sau:

```
n=123
x=3.14
ch='Z'
ten="HONG"
que  "HAI"
```

Con trỏ nhập sẽ dừng tại ký tự <space> trước từ PHONG. Các ký tự còn lại sẽ được nhận trong các câu lệnh nhập tiếp theo.

Ví dụ 2: Xét đoạn chương trình:

```
int m;
float y;
cin >> m >> y;
```

Nếu gõ:

```
<s><s>456<s><s>4.5<Enter>
```

thì kết quả nhập là:

```
m = 456
```

```
y = 4.5
```

Ký tự <Enter> vẫn còn lại trên dòng nhập.

§3. NHẬP KÍ TỰ VÀ CHUỖI KÍ TỰ TỪ BÀN PHÍM

Chúng ta nhận thấy toán tử nhập >> chỉ tiện lợi khi dùng để nhập các giá trị số (nguyên, thực). Để nhập ký tự và chuỗi ký tự nên dùng các phương thức sau (định nghĩa trong lớp istream):

```
cin.get      cin.getline  cin.ignore
```

3.1. Phương thức get có 3 dạng (thực chất có 3 phương thức cùng có tên get):

Dạng 1:

```
int cin.get() ;
```

dùng để đọc một ký tự (kể cả khoảng trắng). Cách thức đọc của cin.get có thể minh họa qua ví dụ sau: Xét các câu lệnh

```
char ch;
```

```
ch = cin.get();
```

+ Nếu gõ

```
ABC<Enter>
```

thì biến ch nhận mã ký tự A, các ký tự BC<Enter> còn lại trên dòng vào.

+ Nếu gõ

```
A<Enter>
```

thì biến ch nhận mã ký tự A, ký tự <Enter> còn lại trên dòng vào.

+ Nếu gõ

```
<Enter>
```

thì biến ch nhận mã ký tự <Enter> (bằng 10) và dòng vào rỗng.

Dạng 2:

```
istream& cin.get(char &ch) ;
```

dùng để đọc một ký tự (kể cả khoảng trắng) và đặt vào một biến kiểu char được tham chiếu bởi ch.

Chú ý:

+ Cách thức đọc của cin.get dạng 2 cũng giống như dạng 1

+ Do cin.get() dạng 2 trả về tham chiếu tới cin, nên có thể sử dụng các phương thức get() dạng 2 nối đuôi nhau. Ví dụ 2 nếu khai báo

```
char ch1, ch2;
```

thì 2 câu lệnh:

```
cin.get(ch1);
```

```
cin.get(ch2);
```

có thể viết chung trên một câu lệnh sau:

```
cin.get(ch1).get(ch2);
```

Dạng 3:

```
istream& cin.get(char *str, int n, char delim = '\n');
```

dùng để đọc một dãy ký tự (kể cả khoảng trắng) và đưa vào vùng nhớ do str trỏ tới. Quá trình đọc kết thúc khi xảy ra một trong 2 tình huống sau:

- + Gặp ký tự giới hạn (cho trong delim). Ký tự giới hạn mặc định là '\n' (Enter)

- + Đã nhận đủ (n-1) ký tự

Chú ý:

- + Ký tự kết thúc chuỗi '\0' được bổ sung vào dãy ký tự nhận được
- + ký tự giới hạn vẫn còn lại trên dòng nhập để dành cho các lệnh nhập tiếp theo.

Chú ý:

- + Cũng giống như get() dạng 2, có thể viết các phương thức get() dạng 3 nối đuôi nhau trên một dòng lệnh.

- + Ký tự <Enter> còn lại trên dòng nhập có thể làm trôi phương thức get() dạng 3. Ví dụ xét đoạn chương trình:

```
char ht[25], qq[20], cq[30];  
cout << "\nHọ tên: ";  
cin.get(ht, 25);  
cout << "\nQuê quán: ";  
cin.get(qq, 20);  
cout << "\nCơ quan: ";  
cin.get(cq, 30);  
cout << "\n" << ht << " " << qq << " " << cq
```

Đoạn chương trình dùng để nhập họ tên, quê quán và cơ quan. Nếu gõ:

```
Pham Thu Huong<Enter>
```

thì câu lệnh get đầu tiên sẽ nhận được chuỗi "Pham Thu Huong" cất vào mảng ht. Ký tự <Enter> còn lại sẽ làm trôi 2 câu lệnh get tiếp theo. Do đó câu lệnh cuối cùng sẽ chỉ in ra Pham Thu Huong.

Để khắc phục tình trạng trên, có thể dùng một trong các cách sau:

+ Dùng phương thức `get()` dạng 1 hoặc dạng 2 để lấy ra ký tự <Enter> trên dòng nhập trước khi dùng `get` (dạng 3).

+ Dùng phương thức `ignore` để lấy ra một số ký tự không cần thiết trên dòng nhập trước khi dùng `get` dạng 3. Phương thức này viết như sau:

```
cin.ignore(n) ; // Lấy ra (loại ra hay bỏ qua)
                // n ký tự trên dòng nhập.
```

Như vậy để có thể nhập được cả quê quán và cơ quan, cần sửa lại đoạn chương trình trên như sau:

```
char ht[25], qq[20], cq[30];
cout << "\nHọ tên: " ;
cin.get(ht,25);
cin.get(); // Nhận <Enter>
cout << "\nQuê quán: " ;
cin.get(qq,20);
ignore(1); // Bỏ qua <Enter>
cout << "\nCơ quan: " ;
cin.get(cq,30);
cout << "\n" << ht << " " << qq << " " << cq
```

3.2. Phương thức `getline`

Tương tự như `get` dạng 3, có thể dùng `getline` để nhập một dãy ký tự từ bàn phím. Phương thức này được mô tả như sau:

```
istream& cin.getline(char *str, int n,
                     char delim = '\n');
```

Phương thức đầu tiên làm việc như `get` dạng 3, sau đó nó loại <Enter> ra khỏi dòng nhập (ký tự <Enter> không đưa vào dãy ký tự nhận được). Như vậy có thể dùng `getline` để nhập nhiều chuỗi ký tự (mà không lo ngại các câu lệnh nhập tiếp theo bị trôi).

Ví dụ đoạn chương trình nhập họ tên, quê quán và cơ quan bên trên có thể viết như sau (bằng cách dùng `getline`):

```
char ht[25], qq[20], cq[30];
cout << "\nHọ tên: " ;
cin.getline(ht,25);
cout << "\nQuê quán: " ;
cin.getline(qq,20);
cout << "\nCơ quan: " ;
cin.get(cq,30);
cout << "\n" << ht << " " << qq << " " << cq
```

Chú ý: Cũng giống như `get()` dạng 2 và `get()` dạng 3, có thể viết các

phương thức `getline()` nối đuôi nhau trên một dòng lệnh. Ví dụ đoạn chương trình trên có thể viết lại như sau:

```
char ht[25], qq[20], cq[30];
cout << "\nHọ tên, Quê quán và Cơ quan: " ;
cin.getline(ht, 25).getline(qq, 20).get(cq, 30);
cout << "\n" << ht << " " << qq << " " << cq
```

3.3. Phương thức `ignore`

Phương thức `ignore` dùng để bỏ qua (loại bỏ) một số ký tự trên dòng nhập. Trong nhiều trường hợp, đây là việc làm cần thiết để không làm ảnh hưởng đến các phép nhập tiếp theo.

Phương thức `ignore` được mô tả như sau:

```
istream& cin.ignore(int n=1);
```

Phương thức sẽ bỏ qua (loại bỏ) n ký tự trên dòng nhập.

3.4. Nhập đồng thời giá trị số và ký tự

Như đã nói trong §2, toán tử nhập `>>` bao giờ cũng để lại ký tự `<Enter>` trên dòng nhập. Ký tự `<Enter>` này sẽ làm trôi các lệnh nhập ký tự hoặc chuỗi ký tự bên dưới. Do vậy cần dùng:

hoặc `ignore()`

hoặc `get()` dạng 1

hoặc `get()` dạng 2

để loại bỏ ký tự `<Enter>` còn sót lại ra khỏi dòng nhập trước khi thực hiện việc nhập ký tự hoặc chuỗi ký tự.

3.5. Ví dụ: Chương trình dưới đây sử dụng lớp `TSINH` (Thí sinh) với 2 phương thức xuất và nhập.

```
//CT7_04.CPP
// Nhập dữ liệu số và ký tự
#include <iostream.h>
#include <conio.h>
struct TS
{
    int sobd;
    char ht[25];
    float dt, dl, dh, td;
};
class TSINH
{
private:
    TS *ts;
    int solts;
```

```

public:
    TSINH()
    {
        ts=NULL;
        sots=0;
    }
    TSINH(int n)
    {
        ts=new TS[n+1];
        sots=n;
    }
    ~TSINH()
    {
        if (sots)
        {
            sots 0;
            ts = NULL;
        }
    }
    void nhap();
    void xuat();
};

void TSINH::nhap()
{
    if (sots)
        for (int i=1; i<=sots; ++i)
        {
            cout << "\nThi sinh "<< i << ": " ;
            cout << "\nSo bao danh: " ;
            cin >> ts[i].sobd;
            cin.ignore();
            cout << "Ho ten: " ;
            cin.get(ts[i].ht,25);
            cout << "Diem toan, ly, hoa: " ;
            cin >> ts[i].dt >> ts[i].dl >> ts[i].dh;
            ts[i].td = ts[i].dt + ts[i].dl + ts[i].dh;
        }
}

void TSINH::xuat()
{
    if (sots)
    {
        cout << "\nDanh sach thi sinh:" ;
        for (int i=1; i<=sots; ++i)
            cout << "\nHo ten: " << ts[i].ht << " So BD: "
                << ts[i].sobd << " Tong diem: "<< ts[i].td;
    }
}

void main()

```

```

{
    int n;
    clrscr();
    cout << "\nSo thi sinh: ";
    cin>>n;
    TSINH *t = new TSINH(n);
    t->nhap();
    t->xuat();
    getch();
    delete t;
}

```

54. DÒNG COUT VÀ TOÁN TỬ XUẤT

4.1. Dòng cout

Dòng cout là một đối tượng kiểu ostream đã định nghĩa trong C++. Đó là dòng xuất (output) chuẩn gắn với màn hình (tương tự như stdout của C). Các thao tác xuất trên dòng cout đồng nghĩa với xuất dữ liệu ra màn hình.

Do cout là một đối tượng của lớp ostream nên với cout chúng ta có thể sử dụng toán tử xuất << và các phương thức xuất của các lớp ios và ostream.

4.2. Toán tử xuất

C++ định nghĩa chồng toán tử dịch trái << để gửi các ký tự ra dòng xuất.

Cách dùng toán tử xuất để xuất dữ liệu từ bộ nhớ ra dòng cout như sau:

```
cout << Tham_số ;
```

Trong đó Tham_số biểu thị một giá trị cần xuất ra màn hình. Giá trị sẽ được biến đổi thành một dãy ký tự trước khi đưa ra dòng xuất. Kiểu của Tham_số có thể như sau:

- Nguyên (xuất giá trị nguyên)
- Thực (xuất giá trị thực)
- ký tự - char (xuất một ký tự)
- con trỏ ký tự - char* (xuất chuỗi ký tự)

Chú ý: Các toán tử xuất có thể viết nối đuôi nhau (để xuất nhiều giá trị) trên một dòng lệnh như sau:

```
cout << Tham_số_1 << Tham_số_2 << ... << Tham_số_k;
```

Chú ý: Toán tử xuất được định nghĩa chồng (trùng tên) với toán tử dịch trái và nó cũng có mức độ ưu tiên như toán tử dịch trái. Xem phụ lục 1 chúng ta thấy toán tử xuất có thứ tự ưu tiên lớn hơn các toán tử trong biểu

thức điều kiện. Vì vậy nếu dùng toán tử xuất để in một biểu thức điều kiện như sau:

```
int a=5, b=10;
cout << "\nMax= " << a>b?a:b ;
```

thì Trình biên dịch sẽ báo lỗi. Để tránh lỗi cần dùng các dấu ngoặc tròn để bao biểu thức điều kiện như sau:

```
int a=5, b=10;
cout << "\nMax= " << (a>b?a:b) ;
```

Tóm lại: Nên bao các biểu thức trong 2 dấu ngoặc tròn.

4.3. Định dạng (tạo khuôn dạng cho) dữ liệu xuất

Việc định dạng dữ liệu xuất hay tạo khuôn dạng cho dữ liệu xuất là một việc cần thiết. Ví dụ cần in các giá trị thực trên 10 vị trí trong đó có 2 vị trí dành cho phần phân.

Bản thân toán tử xuất chưa có khả năng định dạng, mà cần sử dụng các công cụ sau:

- + Các phương thức định dạng
- + Các cờ định dạng
- + Các hàm và bộ phận định dạng

Mục sau sẽ trình bày cách định dạng giá trị xuất.

§5. CÁC PHƯƠNG THỨC ĐỊNH DẠNG

5.1. Nội dung định dạng giá trị xuất

Nội dung định dạng là xác định các thông số:

- Độ rộng quy định
- Độ chính xác
- Ký tự độn
- Và các thông số khác

+ **Độ rộng thực tế của giá trị xuất:** Như đã nói ở trên, C++ sẽ biến đổi giá trị cần xuất thành một chuỗi ký tự rồi đưa chuỗi này ra màn hình. Ta sẽ gọi số ký tự của chuỗi này là độ rộng thực tế của giá trị xuất. Ví dụ với các câu lệnh:

```
int n=4567, m=-23 ;
float x = -3.1416 ;
char ht[] = "Tran Van Thong" ;
```

thì:

Độ rộng thực tế của n là 4, của m là 3, của x là 7, của ht là 14.

+ **Độ rộng quy định** là số vị trí tối thiểu trên màn hình dành để in giá trị. Theo mặc định, độ rộng quy định bằng 0. Chúng ta có thể dùng phương thức `cout.width()` để thiết lập rộng này. Ví dụ câu lệnh:

```
cout.width(8);
```

sẽ thiết lập độ rộng quy định là 8.

+ **Mối quan hệ giữa độ rộng thực tế và độ rộng quy định**

- Nếu độ rộng thực tế lớn hơn hoặc bằng độ rộng quy định thì số vị trí trên màn hình chứa giá trị xuất sẽ bằng độ rộng thực tế.

- Nếu độ rộng thực tế nhỏ hơn độ rộng quy định thì số vị trí trên màn hình chứa giá trị xuất sẽ bằng độ rộng quy định. Khi đó sẽ có một số vị trí dư thừa. Các vị trí dư thừa sẽ được độn (lấp đầy) bằng khoảng trống.

+ **Xác định ký tự độn:** Ký tự độn mặc định là dấu cách (khoảng trống). Tuy nhiên có thể dùng phương thức `cout.fill()` để chọn một ký tự độn khác. Ví dụ với các câu lệnh sau:

```
int n=123; // Độ rộng thực tế là 3
cout.fill('*'); // Ký tự độn là *
cout.width(5); // Độ rộng quy định là 5
cout << n;
```

thì kết quả in ra là:

```
**123
```

+ **Độ chính xác** là số vị trí dành cho phần phân (khi in số thực). Độ chính xác mặc định là 6. Tuy nhiên có thể dùng phương thức `cout.precision()` để chọn độ chính xác. Ví dụ với các câu lệnh:

```
float x = 34.455; // Độ rộng thực tế 6
cout.precision(2); // Độ chính xác 2
cout.width(8); // Độ rộng quy ước 8
cout.fill('0'); // Ký tự độn là số 0
cout << x;
```

thì kết quả in ra là:

```
0034.46
```

5.2. Các phương thức định dạng

1. Phương thức

```
int cout.width()
```

cho biết độ rộng quy định hiện tại.

2. Phương thức

```
int cout.width(int n)
```

thiết lập độ rộng quy định mới là n và trả về độ rộng quy định trước đó.

Chú ý: Độ rộng quy định n chỉ có tác dụng cho một giá trị xuất. Sau đó C++ lại áp dụng độ rộng quy định bằng 0.

Ví dụ với các câu lệnh:

```
int m=1234, n=56;  
cout << "\nAB"  
cout.width(6);  
cout << m ;  
cout << n ;
```

thì kết quả in ra là:

```
AB 123456  
(giữa B và số 1 có 2 dấu cách).
```

3. Phương thức

```
int cout.precision()
```

cho biết độ chính xác hiện tại (đang áp dụng để xuất các giá trị thực).

4. Phương thức

```
int cout.precision(int n)
```

thiết lập độ chính xác sẽ áp dụng là n và cho biết độ chính xác trước đó. Độ chính xác được thiết lập sẽ có hiệu lực cho tới khi gặp một câu lệnh thiết lập độ chính xác mới.

5. Phương thức

```
char cout.fill()
```

cho biết ký tự độn hiện tại đang được áp dụng.

6. Phương thức

```
char cout.fill(char ch)
```

quy định ký tự độn mới sẽ được dùng là ch và cho biết ký tự độn đang dùng trước đó. Ký tự độn được thiết lập sẽ có hiệu lực cho tới khi gặp một câu lệnh chọn ký tự độn mới.

Ví dụ xét chương trình:

```
//CT1_06.CPP  
// Các phương thức định dạng  
#include <iostream.h>  
#include <conio.h>  
void main()  
{  
clrscr();  
float x=-3.1551, y=-23.45421;  
cout.precision(2);  
cout.fill('*');
```

```

cout << "\n" ;
cout.width(8);
cout << x;
cout << "\n" ;
cout.width(8);
cout << y;
getch();
}

```

Sau khi thực hiện, chương trình in ra màn hình 2 dòng sau:

```

***-3.16
**-23.45

```

§6. CỜ ĐỊNH DẠNG

6.1. Khái niệm chung về cờ

Mỗi cờ chứa trong một bit. Cờ có 2 trạng thái:

Bật (on) - có giá trị 1

Tắt (off) - có giá trị 0

(Trong 6.3 sẽ trình bày các phương thức dùng để bật, tắt các cờ)

Các cờ có thể chứa trong một biến kiểu long. Trong tệp <iostream.h> đã định nghĩa các cờ sau:

ios::left	ios::right	ios::internal
ios::dec	ios::oct	ios::hex
ios::fixed	ios::scientific	ios::showpos
ios::uppercase	ios::showpoint	ios::showbase

6.2. Công dụng của các cờ

Có thể chia các cờ thành các nhóm:

Nhóm 1 gồm các cờ định vị (căn lề) :

ios::left ios::right ios::internal

Cờ ios::left: Khi bật cờ ios::left thì giá trị in ra nằm bên trái vùng quy định, các ký tự độn nằm sau, ví dụ:

```

35***
-89**

```

Cờ ios::right: Khi bật cờ ios::right thì giá trị in ra nằm bên phải vùng quy định, các ký tự độn nằm trước, ví dụ:

```

***35
**-89

```

Chú ý: Mặc định cờ ios::right bật.

Cờ ios::internal: Cờ ios::internal có tác dụng giống như cờ ios::right chỉ khác là dấu (nếu có) in đầu tiên, ví dụ:

***35

_*89

Chương trình sau minh họa cách dùng các cờ định vị:

```
//CT7_06.CPP
// Các phương thức định dạng
// Cờ định vị
#include <iostream.h>
#include <conio.h>
void main()
{
    clrscr();
    float x=-87.1551, y=23.45421;
    cout.precision(2);
    cout.fill('*');
    cout.setf(ios::left); // Bật cờ ios::left
    cout << "\n" ;
    cout.width(8);
    cout << x;
    cout << "\n" ;
    cout.width(8);
    cout << y;
    cout.setf(ios::right); // Bật cờ ios::right
    cout << "\n" ;
    cout.width(8);
    cout << x;
    cout << "\n" ;
    cout.width(8);
    cout << y;
    cout.setf(ios::internal); // // Bật cờ ios::internal
    cout << "\n" ;
    cout.width(8);
    cout << x;
    cout << "\n" ;
    cout.width(8);
    cout << y;
    getch();
}
```

Sau khi thực hiện chương trình in ra 6 dòng như sau:

-87.16**

23.45***

******-87.16

*******23.45

-**87.16

*******23.45

Nhóm 2 gồm các cờ định dạng số nguyên:

`ios::dec ios::oct ios::hex`

+ Khi `ios::dec` bật (mặc định): Số nguyên được in dưới dạng cơ số 10

+ Khi `ios::oct` bật : Số nguyên được in dưới dạng cơ số 8

+ Khi `ios::hex` bật : Số nguyên được in dưới dạng cơ số 16

Nhóm 3 gồm các cờ định dạng số thực:

`ios::fixed ios::scientific ios::showpoint`

Mặc định: Cờ `ios::fixed` bật (on) và cờ `ios::showpoint` tắt (off).

+ Khi `ios::fixed` bật và cờ `ios::showpoint` tắt thì số thực in ra dưới dạng thập phân, số chữ số phần phân (sau dấu chấm) được tính bằng độ chính xác n nhưng khi in thì bỏ đi các chữ số 0 ở cuối.

Ví dụ nếu độ chính xác $n = 4$ thì:

Số thực -87.1500 được in: -87.15

Số thực 23.45425 được in: 23.4543

Số thực 678.0 được in: 678

+ Khi `ios::fixed` bật và cờ `ios::showpoint` bật thì số thực in ra dưới dạng thập phân, số chữ số phần phân (sau dấu chấm) được in ra đúng bằng độ chính xác n .

Ví dụ nếu độ chính xác $n = 4$ thì:

Số thực -87.1500 được in: -87.1500

Số thực 23.45425 được in: 23.4543

Số thực 678.0 được in: 678.0000

+ Khi `ios::scientific` bật và cờ `ios::showpoint` tắt thì số thực in ra dưới dạng mũ (dạng khoa học). Số chữ số phần phân (sau dấu chấm) của phần định trị được tính bằng độ chính xác n nhưng khi in thì bỏ đi các chữ số 0 ở cuối.

Ví dụ nếu độ chính xác $n = 4$ thì:

Số thực -87.1500 được in: -8.715e+01

Số thực 23.45425 được in: 2.3454e+01

Số thực 678.0 được in: 6.78e+02

+ Khi `ios::scientific` bật và cờ `ios::showpoint` bật thì số thực in ra dưới dạng mũ. Số chữ số phần phân (sau dấu chấm) của phần định trị được in đúng bằng độ chính xác `n`.

Ví dụ nếu độ chính xác `n = 4` thì:

Số thực -87.1500 được in: -8.7150e+01

Số thực 23.45425 được in: 2.3454e+01

Số thực 678.0 được in: 6.7800e+01

Nhóm 4 gồm các hiển thị:

`ios::showpos` `ios::showbase` `ios::uppercase`

Cờ `ios::showpos`

+ Nếu cờ `ios::showpos` tắt (mặc định) thì dấu cộng không được in trước số dương.

+ Nếu cờ `ios::showpos` bật thì dấu cộng được in trước số dương.

Cờ `ios::showbase`

+ Nếu cờ `ios::showbase` bật thì số nguyên hệ 8 được in bắt đầu bằng ký tự 0 và số nguyên hệ 16 được bắt đầu bằng các ký tự 0x.

Ví dụ nếu `a = 40` thì:

dạng in hệ 8 là: 050

dạng in hệ 16 là 0x28

+ Nếu cờ `ios::showbase` tắt (mặc định) thì không in 0 trước số nguyên hệ 8 và không in 0x trước số nguyên hệ 16. Ví dụ nếu `a = 40` thì:

dạng in hệ 8 là: 50

dạng in hệ 16 là 28

Cờ `ios::uppercase`

+ Nếu cờ `ios::uppercase` bật thì các chữ số hệ 16 (như A, B, C, ...) được in dưới dạng chữ hoa.

+ Nếu cờ `ios::uppercase` tắt (mặc định) thì các chữ số hệ 16 (như A, B, C, ...) được in dưới dạng chữ thường.

6.3. Các phương thức bật tắt cờ

Các phương thức này định nghĩa trong lớp `ios`.

+ Phương thức

`long cout.setf(long f);`

sẽ bật các cờ liệt kê trong `f` và trả về một giá trị `long` biểu thị các cờ đang bật. Thông thường giá trị `f` được xác định bằng cách tổ hợp các cờ trình bày trong mục 6.1.

Ví dụ câu lệnh:

```
cout.setf(ios::showpoint | ios::scientific);
```

sẽ bật các cờ ios::showpoint và ios::scientific.

+ Phương thức

```
long cout.unsetf(long f);
```

sẽ tắt các cờ liệt kê trong f và trả về một giá trị long biểu thị các cờ đang bật. Thông thường giá trị f được xác định bằng cách tổ hợp các cờ trình bày trong mục 6.1.

Ví dụ câu lệnh:

```
cout.unsetf(ios::showpoint | ios::scientific);
```

sẽ tắt các cờ ios::showpoint và ios::scientific.

+ Phương thức

```
long cout.flags(long f);
```

có tác dụng giống như cout.setf(long). Ví dụ câu lệnh:

```
cout.flags(ios::showpoint | ios::scientific);
```

sẽ bật các cờ ios::showpoint và ios::scientific.

+ Phương thức

```
long cout.flags();
```

sẽ trả về một giá trị long biểu thị các cờ đang bật.

§7. CÁC BỘ PHẬN ĐỊNH DẠNG VÀ CÁC HÀM ĐỊNH DẠNG

7.1. Các bộ phận định dạng (định nghĩa trong <iostream.h>)

Các bộ phận định dạng gồm:

```
dec    // như cờ ios::dec
oct    // như cờ ios::oct
hex    // như cờ ios::hex
endl   // xuất ký tự '\n' (chuyển dòng)
flush  // đẩy dữ liệu ra thiết bị xuất
```

Chúng có tác dụng như cờ định dạng nhưng được viết nổi đuôi trong toán tử xuất nên tiện sử dụng hơn.

Ví dụ xét chương trình đơn giản sau:

```
//CT7_08.CPP
// Bộ phận định dạng
#include <iostream.h>
```

```
#include <conio.h>
void main()
{
    clrscr();
    cout.setf(ios::showbase)
    cout << "ABC" << endl << hex << 40 << " " << 41;
    getch();
}
```

Chương trình sẽ in 2 dòng sau ra màn hình:

```
ABC
0x28 0x29
```

7.2. Các hàm định dạng (định nghĩa trong <iomanip.h>)

Các hàm định dạng gồm:

```
setw(int n)           // như cout.width(int n)
setprecision(int n)   // như cout.precision(int n)
setfill(char ch)      // như cout.fill(char ch)
setiosflags(long l)   // như cout.setf(long f)
resetiosflags(long l) // như cout.unsetf(long f)
```

Các hàm định dạng có tác dụng như các phương thức định dạng nhưng được viết nối đuôi trong toán tử xuất nên tiện sử dụng hơn.

Chú ý 1: Các hàm định dạng (cũng như các bộ phận định dạng) cần viết trong các toán tử xuất. Một hàm định dạng đứng một mình như một câu lệnh sẽ không có tác dụng định dạng.

Chú ý 2: Muốn sử dụng các hàm định dạng cần bổ sung vào đầu chương trình câu lệnh:

```
#include <iomanip.h>
```

Ví dụ có thể thay phương thức

```
cout.setf(ios::showbase);
```

trong chương trình của mục 7.1 bằng hàm

```
cout << setiosflags(ios::showbase);
```

(chú ý hàm phải viết trong toán tử xuất)

Như vậy chương trình trong 7.1 có thể viết lại theo các phương án sau:

Phương án 1:

```
#include <iostream.h>
#include <iomanip.h>
```

```
#include <conio.h>
void main()
{
    clrscr();
    cout << setiosflags(ios::showbase) ;
    cout << "ABC" << endl << hex << 40 << " " << 41;
    getch();
}
Phương án 2:
#include <iostream.h>
#include <iomanip.h>
#include <conio.h>
void main()
{
    clrscr();
    cout << "ABC" << endl << setiosflags(ios::showbase)
    << hex << 40 << " " << 41;
    getch();
}
```

Dưới đây là ví dụ khác về việc dùng các hàm và bộ phận định dạng. Các câu lệnh:

```
int i = 23;
cout << i << endl << setiosflags(ios::showbase)
    << hex << i << dec << setfill('*')
    << endl << setw(4) << i << setfill('0')
    << endl << setw(5) << i ;
```

sẽ in ra màn hình như sau:

```
23
0x17
**23
00023
```

7.3. Ví dụ: Chương trình dưới đây minh họa cách dùng các hàm định dạng và phương thức định dạng để in danh sách thí sinh dưới dạng bảng với các yêu cầu sau: Số báo danh in 4 ký tự (chèn thêm số 0 vào trước ví dụ 0003), tổng điểm in với đúng một chữ số phần phân.

```
//CT7_08.CPP
// Bộ phận định dạng
// Hàm định dạng
// Cơ định dạng
#include <iostream.h>
#include <iomanip.h>
#include <conio.h>
```

```

struct TS
{
    int sobd;
    char ht[25];
    float dt, dl, dh, td;
};
class TSINH
{
private:
    TS *ts;
    int sots;
public:
    TSINH()
    {
        ts = NULL;
        sots=0;
    }
    TSINH(int n)
    {
        ts=new TS[n+1];
        sots=n;
    }
    ~TSINH()
    {
        if (sots)
        {
            sots=0;
            ts = NULL;
        }
    }
    void nhap();
    void sapxep();
    void xuat();
};
void TSINH::nhap()
{
    if (sots)
        for (int i=1; i<=sots; ++i)
        {
            cout << "\nThi sinh "<< i << ": " ;
            cout << "\nSo bao danh: " ;
            cin >> ts[i].sobd;
            cin.ignore();
            cout << "Ho ten: " ;
            cin.get(ts[i].ht, 25);
            cout << "Diem toan, ly, hoa: " ;
            cin >> ts[i].dt >> ts[i].dl >> ts[i].dh;
            ts[i].td = ts[i].dt + ts[i].dl + ts[i].dh;
        }
}
void TSINH::sapxep()

```

```

{
    int i,j;
    for (i=1; i< sots; ++i)
        for (j=i+1; j<= sots; ++j)
            if (ts[i].td < ts[j].td)
            {
                TS tg;
                tg=ts[i];
                ts[i]=ts[j];
                ts[j]=tg;
            }
}

void TSINH::xuat()
{
    if (sots)
    {
        cout << "\nDanh sach thi sinh:" ;
        cout.precision(1);
        cout << setiosflags(ios::left);
        cout << "\n" << setw(20) << "Ho ten" << setw(8)
            << "So BD" << setw(10) << "Tong diem";
        for (int i=1; i<=sots; ++i)
            cout << "\n" << setw(20)<<setiosflags(ios::left) <<
ts[i].ht
                <<          setw(4)          <<          setfill('0')          <<
setiosflags(ios::right)
                << ts[i].sobd << "          " << setfill(32)
                << setiosflags(ios::left|ios::showpoint)
                << setw(10) << ts[i].td;
    }
}

void main()
{
    int n;
    clrscr();
    cout << "\nSo thi sinh: ";
    cin>>n;
    TSINH *t = new TSINH(n);
    t->nhap();
    t->sapxep();
    t->xuat();
    getch();
    delete t;
}

```

§8. CÁC DÒNG TIN CHUẨN

Có 4 dòng tin (đối tượng của các lớp Stream) đã định nghĩa trước, được cài đặt khi chương trình khởi động. Hai trong số đó đã nói ở trên là:

cin dòng input chuẩn gắn với bàn phím, giống như stdin của C.

cout dòng output chuẩn gắn với màn hình, giống như stdout của C.

Hai dòng tin chuẩn khác là:

cerr dòng output lỗi chuẩn gắn với màn hình, giống như stderr của C.

clog giống cerr nhưng có thêm bộ đệm.

Chú ý 1: Có thể dùng các dòng cerr và clog để xuất ra màn hình như đã dùng đối với cout.

Chú ý 2: Vì clog có thêm bộ đệm, nên dữ liệu được đưa vào bộ đệm. Khi đầy bộ đệm thì đưa dữ liệu từ bộ đệm ra dòng clog. Vì vậy trước khi kết thúc xuất cần dùng phương thức:

```
clog.flush();
```

để đẩy dữ liệu từ bộ đệm ra clog.

Chương trình sau minh họa cách dùng dòng clog. Chúng ta nhận thấy, nếu bỏ câu lệnh clog.flush() thì sẽ không nhìn thấy kết quả xuất ra màn hình khi chương trình tạm dừng bởi câu lệnh getch().

```
// Dùng clog và flush
#include <iostream.h>
#include <conio.h>
void main()
{
    clrscr();
    float x=-87.1500, y=23.45425, z=678.0;
    clog.setf(ios::scientific);
    clog.precision(4);
    clog.fill('*');
    clog << "\n";
    clog.width(10);
    clog << x;
    clog << "\n";
    clog.width(10);
    clog << y;
    clog << "\n";
    clog.width(10);
    clog << z;
    clog.flush();
    getch();
}
```

59. XUẤT RA MÁY IN

Trong số 4 dòng tin chuẩn không dòng nào gắn với máy in. Như vậy không thể dùng các dòng này để xuất dữ liệu ra máy in. Để xuất dữ liệu ra máy in (cũng như nhập, xuất trên tệp) cần tạo ra các dòng tin mới và cho nó gắn với thiết bị cụ thể. C++ cung cấp 3 lớp stream để làm điều này, đó là các lớp:

ifstream dùng để tạo dòng nhập

ofstream dùng để tạo dòng xuất

fstream dùng để tạo dòng nhập, dòng xuất hoặc dòng nhập-xuất

Mỗi lớp có 4 hàm tạo dùng để khai báo các dòng tin (đối tượng dòng tin). Trong mục sau sẽ nói thêm về các hàm tạo này.

Để tạo một dòng xuất và gắn nó với máy in ta có thể dùng một trong các hàm tạo sau:

```
ofstream Tên_dòng_tin(int fd);
```

```
ofstream Tên_dòng_tin(int fd, char *buf, int n);
```

Trong đó:

+ Tên_dòng_tin là tên biến đối tượng kiểu ofstream hay gọi là tên dòng xuất do chúng ta tự đặt.

+ fd (file descriptor) là chỉ số tệp tin. Chỉ số tệp tin định sẵn đối với stdprn (máy in chuẩn) là 4.

+ Các tham số buf và n xác định một vùng nhớ n byte do buf trỏ tới. Vùng nhớ sẽ được dùng làm bộ đệm cho dòng xuất.

Ví dụ 1 câu lệnh:

```
ofstream prn(4);
```

sẽ tạo dòng tin xuất prn và gắn nó với máy in chuẩn. Dòng prn sẽ có bộ đệm mặc định. Dữ liệu trước hết chuyển vào bộ đệm, khi đầy bộ đệm thì dữ liệu sẽ được đẩy từ bộ đệm ra dòng prn. Để chủ động yêu cầu đẩy dữ liệu từ bộ đệm ra dòng prn có thể sử dụng phương thức flush hoặc bộ phận định dạng flush. Cách viết như sau:

```
prn.flush(); // Phương thức
```

```
prn << flush; // Bộ phận định dạng
```

Các câu lệnh sau sẽ xuất dữ liệu ra prn (máy in) và ý nghĩa của chúng như sau:

```
prn << "\nTong = " << (4+9); // Đưa một dòng vào bộ đệm
```

```
prn << "\nTich =" << (4*9); // Đưa tiếp dòng thứ 2 vào bộ đệm
```

```
prn.flush(); // Đẩy dữ liệu từ bộ đệm ra máy in (in 2 dòng)
```

Các câu lệnh dưới đây cũng xuất dữ liệu ra máy in nhưng sẽ in từng dòng một:

```
prn << "\nTong = " << (4+9) << flush ; // In một dòng
prn << "\nTich = " << (4*9) ; << flush // In dòng thứ hai
```

Ví dụ 2: Các câu lệnh

```
char buf[1000] ;
ofstream prn(4,buf,1000) ;
```

sẽ tạo dòng tin xuất prn và gắn nó với máy in chuẩn. Dòng xuất prn sử dụng 1000 byte của mảng buf làm bộ đệm. Các câu lệnh dưới đây cũng xuất dữ liệu ra máy in:

```
prn << "\nTong = " << (4+9) ; // Đưa dữ liệu vào bộ đệm
prn << "\nTich = " << (4*9) ; // Đưa dữ liệu vào bộ đệm
prn.flush() : // Xuất 2 dòng (ở bộ đệm) ra máy in
```

Chú ý: Trước khi kết thúc chương trình, dữ liệu từ bộ đệm sẽ được tự động đẩy ra máy in.

Chương trình minh họa: Chương trình dưới đây tương tự như chương trình trong mục 7.3 (chỉ sửa đổi phương thức xuất) nhưng thay việc xuất ra màn hình bằng xuất ra máy in.

```
//CT7_08B.CPP
// Xuất ra máy in
// Bỏ phân định dạng
// Hàm định dạng
#include <iostream.h>
#include <iomanip.h>
#include <conio.h>
struct TS
{
    int sobd;
    char ht[25];
    float dt,dl,dh,td;
} ;
class TSINH
{
private:
    TS *ts;
    int sots;
public:
    TSINH()
    {
        ts=NULL;
        sots=0;
    }
};
```

```

    }
    TSINH(int n)
    {
        ts=new TS[n+1];
        sots=n;
    }
    ~TSINH()
    {
        if (sots)
        {
            sots=0;
            ts = NULL;
        }
    }
    void nhap();
    void sapxep();
    void xuat();
} ;

void TSINH::nhap()
{
    if (sots)
        for (int i=1; i<=sots; ++i)
        {
            cout << "\nThi sinh "<< i << ": " ;
            cout << "\nSo bao danh: " ;
            cin >> ts[i].sobd;
            cin.ignore();
            cout << "Ho ten: " ;
            cin.get(ts[i].ht,25);
            cout << "Diem toan, ly, hoa: " ;
            cin >> ts[i].dt >> ts[i].dl >> ts[i].dh;
            ts[i].td = ts[i].dt + ts[i].dl + ts[i].dh;
        }
}

void TSINH::sapxep()
{
    int i, j;
    for (i=1; i< sots; ++i)
        for (j=i+1; j<= sots; ++j)
            if (ts[i].td < ts[j].td)
            {
                TS tg;
                tg=ts[i];
                ts[i]=ts[j];
                ts[j]=tg;
            }
}

```

```

    }
    void TSINH::xuat()
    {
        ostream prn(4);
        if (sots)
        {
            prn << "\nDanh sach thi sinh:" ;
            prn.precision(1);
            prn << setiosflags(ios::left);
            prn << "\n" << setw(20) << "Ho ten" << setw(8)
                << "So BD" << setw(10) << "Tong diem";
            for (int i=1; i<=sots; ++i)
                prn << "\n" << setw(20) << setiosflags(ios::left)
<<ts[i].ht <<
                    setw(4) << setfill('0') << setiosflags(ios::right) <<
ts[i].sobd
                    << " " << setfill(32)
<<setiosflags(ios::left|ios::showpoint)
                    <<setw(10) << ts[i].td;
        }
    }
}
void main()
{
    int n;
    clrscr();
    cout << "\nSo thi sinh: ";
    cin>>n;
    TSINH *t = new TSINH(n);
    t->nhap();
    t->sapxep();
    t->xuat();
    getch();
    delete t;
}

```

§10. LÀM VIỆC VỚI TỆP

10.1. Các lớp dùng để nhập, xuất dữ liệu lên tệp

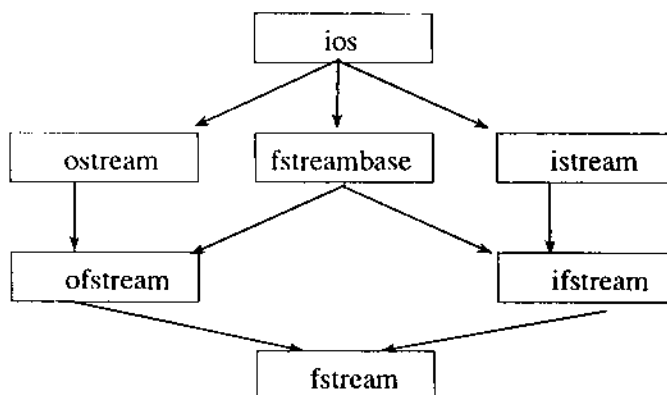
Như đã nói ở trên, C++ cung cấp 4 dòng tin chuẩn để làm việc với bàn phím và màn hình. Muốn nhập xuất lên tệp chúng ta cần tạo các dòng tin mới (khai báo các đối tượng Stream) và gắn chúng với một tệp cụ thể. C++ cung cấp 3 lớp stream để làm điều này, đó là các lớp:

ofstream dùng để tạo các dòng xuất (ghi tệp)

ifstream dùng để tạo các dòng nhập (đọc tệp)

fstream dùng để tạo các dòng nhập, dòng xuất hoặc dòng nhập-xuất

Sơ đồ dẫn xuất các lớp như sau:



10.2. Ghi dữ liệu lên tệp

Thủ tục ghi dữ liệu lên tệp như sau:

1. Dùng lớp ofstream để tạo ra một dòng xuất và gắn nó với một tệp cụ thể. Khi đó việc xuất dữ liệu ra dòng này đồng nghĩa với việc ghi dữ liệu lên tệp.
2. Thực hiện xuất dữ liệu ra dòng xuất vừa tạo như thể xuất dữ liệu ra dòng xuất chuẩn cout.

10.3. Đọc dữ liệu từ tệp

Thủ tục đọc dữ liệu từ tệp như sau:

1. Dùng lớp ifstream để tạo ra một dòng nhập và gắn nó với một tệp cụ thể. Khi đó việc nhập dữ liệu từ dòng này đồng nghĩa với việc đọc dữ liệu từ tệp.
2. Thực hiện nhập dữ liệu từ dòng nhập vừa tạo như thể nhập dữ liệu từ dòng nhập chuẩn cin.

10.4. Đọc - ghi dữ liệu đồng thời trên tệp

Thủ tục đọc-ghi dữ liệu đồng thời trên tệp như sau:

1. Dùng lớp fstream để tạo ra một dòng nhập-xuất và gắn nó với một tệp cụ thể.
2. Thực hiện nhập dữ liệu từ dòng nhập-xuất vừa tạo như thể nhập dữ liệu từ dòng nhập chuẩn cin.
3. Thực hiện xuất dữ liệu ra dòng nhập-xuất vừa tạo như thể xuất dữ liệu ra dòng xuất chuẩn cout.

Nhận xét: Như vậy:

1. Việc xuất dữ liệu ra máy in hoặc lên tệp được thực hiện hoàn toàn giống như xuất dữ liệu ra dòng xuất chuẩn cout (màn hình).

2. Việc đọc dữ liệu từ tệp được thực hiện hoàn toàn giống như nhập dữ liệu từ dòng nhập chuẩn cin (bàn phím).

§11. GHI DỮ LIỆU LÊN TỆP

11.1. Lớp ofstream

Để ghi dữ liệu lên tệp chúng ta sử dụng lớp ofstream. Lớp ofstream thừa kế các phương thức của các lớp ios và ostream. Nó cũng thừa kế phương thức:

`close`

của lớp fstreambase. Ngoài ra lớp ofstream có thêm các hàm tạo và các phương thức sau:

1. Hàm tạo:

```
ofstream() ; // Không đối
```

dùng để tạo một đối tượng ofstream (dòng xuất), chưa gắn với tệp.

2. Hàm tạo:

```
ofstream(const char *fn, int mode = ios::out,  
         int prot = filebuf::openprot);
```

dùng để tạo một đối tượng ofstream, mở tệp có tên fn để ghi và gắn đối tượng vừa tạo với tệp được mở.

+ Tham số fn cho biết tên tệp.

+ Tham số mode có giá trị mặc định là ios::out (mở để ghi). Tham số này có thể là một hợp của các giá trị sau:

ios::binary ghi theo kiểu nhị phân (mặc định theo kiểu văn bản)

ios::out ghi tệp, nếu tệp đã có thì nó bị xóa

ios::app ghi bổ sung vào cuối tệp

ios::ate chuyển con trỏ tệp tới cuối tệp sau khi mở tệp

ios::trunc xóa nội dung của tệp nếu nó tồn tại

ios::nocreate nếu tệp chưa có thì không làm gì (bỏ qua)

ios::noreplace nếu tệp đã có thì không làm gì (bỏ qua)

+ Tham số thứ ba prot quy định cấp bảo vệ của dòng tin, tham số này có thể bỏ qua vì nó đã được gán một giá trị mặc định.

3. Hàm tạo:

```
ofstream(int fd);
```

dùng để tạo một đối tượng ofstream và gắn nó với một tệp có chỉ số fd đang mở.

(Đề mở và lấy chỉ số (số hiệu) tệp có thể dùng hàm `_open`, xem cuốn Kỹ thuật Lập trình C của tác giả).

4. Hàm tạo:

```
ofstream(int fd, char *buf, int n);
```

dùng để tạo một đối tượng `ofstream`, gắn nó với một tệp có chỉ số `fd` đang mở và sử dụng một vùng nhớ `n` byte do `buf` trỏ tới làm bộ đệm.

5. Phương thức:

```
void open(const char *fn, int mode = ios::out,  
          int prot = filebuf::openprot);
```

dùng để mở tệp có tên `fn` để ghi và gắn nó với đối tượng `ofstream`. Các tham số của phương thức có cùng ý nghĩa như trong hàm tạo thứ 2.

11.2. Các cách ghi tệp

Có 2 cách chính sau:

- + *Cách 1*: Dùng hàm tạo 2 để xây dựng một dòng xuất, mở một tệp để ghi và gắn tệp với dòng xuất. Sau đó dùng toán tử xuất `<<` và các phương thức để xuất dữ liệu ra dòng xuất vừa tạo như thể xuất dữ liệu ra `cout` (xem các mục trên).

- + *Cách 2*: Dùng hàm tạo 1 để xây dựng một dòng xuất. Sau đó dùng phương thức `open` để mở một tệp cụ thể và cho gắn với dòng xuất vừa xây dựng. Khi không cần làm việc với tệp này nữa, chúng ta có thể dùng phương thức `close` để chấm dứt mọi ràng buộc giữa dòng xuất và tệp. Sau đó có thể gắn dòng xuất với tệp khác. Theo cách này, có thể dùng một dòng xuất (đối tượng `ofstream`) để xuất dữ liệu lên nhiều tệp khác nhau.

11.3. Ví dụ

Chương trình 1: Chương trình dưới đây sẽ nhập danh sách n thí sinh. Thông tin thí sinh gồm: Họ tên, tỉnh hoặc thành phố cư trú, số báo danh, các điểm toán lý hoá. Dữ liệu thí sinh được ghi trên 2 tệp: Tệp `DS1.DL` ghi thí sinh theo thứ tự nhập từ bàn phím, tệp `DS2.DL` ghi thí sinh theo thứ tự giảm của tổng điểm. Cấu trúc của 2 tệp như sau:

Dòng đầu ghi một số nguyên bằng số thí sinh.

Các dòng tiếp theo ghi dữ liệu của thí sinh. Mỗi thí sinh ghi trên 2 dòng, dòng 1 ghi họ tên trên 24 vị trí và tên tỉnh trên 20 vị trí. Dòng 2 ghi số báo danh (6 vị trí), các điểm toán, lý, hoá và tổng điểm (mỗi điểm ghi trên 6 vị trí trong đó một vị trí chứa phân phân). Chương trình sử dụng lớp `TS` (Thí sinh) có 3 phương thức: Nhập, sắp xếp và ghi tệp. Cách ghi tệp sử dụng ở đây là cách 1: Dùng hàm tạo dạng 2 của lớp `ofstream`.

Chương trình 2 ngay bên dưới cũng giải quyết cùng bài toán nêu trên nhưng sử dụng cách ghi tệp thứ 2 (dùng hàm tạo 1 và phương thức open)

Một điều đáng nói ở đây là việc nhập một chuỗi ký tự (như họ tên và tên tính) bằng các phương thức get hoặc getline chưa được thuận tiện, vì 2 lý do sau: thứ nhất là các phương thức này có thể bị ký tự chuyển dòng (còn sót trên cin) làm trôi. Thứ hai là các phương thức này có thể để lại một số ký tự trên dòng cin (nếu số ký tự gõ nhiều hơn so với quy định) và các ký tự này sẽ gây ảnh hưởng đến các phép nhập tiếp theo. Để khắc phục các nhược điểm trên, chúng ta đưa vào 2 chương trình trên hàm getstr để nhập chuỗi ký tự từ bàn phím.

```
//CT7_10.CPP
// Ghi Tệp
#include <iostream.h>
#include <iomanip.h>
#include <fstream.h>
#include <conio.h>
#include <stdlib.h>
#include <ctype.h>
void getstr(char *str,int n)
{
    char tg[21];
    while(1) // Bỏ qua Enter và nhập tối đa n-1 ký tự
    {
        cin.get(str,n);
        if (str[0])
            break;
        else
            cin.ignore();
    }
    while(1) //Loại các ký tự còn lại ra khỏi dòng nhập cin
    {
        cin.get(tg,20);
        if (tg[0]!='\n')
        {
            cin.ignore();
            break;
        }
    }
}
struct TSINH
{
    char ht[25];
    char ttinh[21];
    int sobd;
    float dt,dl,dh,td;
};
class TS
```

```

{
    private:
        int sots;
        TSINH *ts;
    public:
        TS()
        {
            sots=0;
            ts = NULL;
        }
        void nhap();
        void sapxep();
        void ghitep(char *ttep);
};

void TS::nhap()
{
    cout << "\n So thi sinh: " ;
    cin >> sots ;
    int n=sots;
    ts = new TSINH[n+1];
    for (int i=1; i<=n; ++i)
    {
        cout << "\n Nhap thi sinh thu: " << i << endl;
        cout << "Họ ten: " ;
        getstr(ts[i].ht,25);
        cout << "Tinh hoac thanh pho: " ;
        getstr(ts[i].ttinh,21);
        cout << "So bao danh: " ;
        cin >> ts[i].sobd ;
        cout << "Cac diem toan, ly, hoa: " ;
        cin >> ts[i].dt >> ts[i].dl >> ts[i].dh ;
        ts[i].td =ts[i].dt + ts[i].dl + ts[i].dh ;
    }
}

void TS::sapxep()
{
    int n = sots;
    for (int i=1; i< n; ++i)
        for (int j=i+1; j<= n; ++j)
            if (ts[i].td < ts[j].td)
            {
                TSINH tg = ts[i];
                ts[i] = ts[j];
                ts[j] = tg;
            }
}

void TS::ghitep(char *ttep)
{
    ofstream f(ttep);
    f << sots ;
    f << setprecision(1) << setiosflags(ios::showpoint);
}

```

```

for (int i=1; i<=sots; ++i)
{
    f << endl << setw(24) << ts[i].ht <<
        setw(20) << ts[i].ttinh ;
    f << endl << setw(6) << ts[i].sobd
        << setw(6) << ts[i].dt
        << setw(6) << ts[i].dl
        << setw(6) << ts[i].dh
        << setw(6) << ts[i].td ;
}
f.close();
}
void main()
{
    clrscr();
    TS t;
    t.nhap();
    t.ghitep("DS1.DL");
    t.sapxep();
    t.ghitep("DS2.DL");
    cout << "\n Hoan thanh";
    getch();
}

```

Chương trình 2:

```

//CT7 11.CPP
// Ghi Tep
#include <iostream.h>
#include <iomanip.h>
#include <fstream.h>
#include <conio.h>
#include <stdlib.h>
#include <ctype.h>
void getstr(char *str,int n)
{
    char tg[21];
    while(1)
    {
        cin.get(str,n);
        if (str[0])
            break;
        else
            cin.ignore();
    }
    while(1)
    {
        cin.get(tg,20);
        if (tg[0]==0)
        {
            cin.ignore();
            break;
        }
    }
}

```

```

    }
}
}
struct TSINH
{
    char ht[25];
    char ttinh[21];
    int sobd;
    float dt,dl,dh,td;
};
class TS
{
private:
    int sots;
    TSINH *ts;
public:
    TS()
    {
        sots=0;
        ts = NULL;
    }
    void nhap();
    void sapxep();
    void ghitep(char *ttep);
};
void TS::nhap()
{
    cout << "\n So thi sinh: " ;
    cin >> sots ;
    int n=sots;
    ts = new TSINH[n+1];
    for (int i=1; i<=n; ++i)
    {
        cout << "\n Nhap thi sinh thu: " << i << endl;
        cout << "Ho ten: " ;
        getstr(ts[i].ht,25);
        cout << "Tinh hoac thanh pho: " ;
        getstr(ts[i].ttinh,21);
        cout << "So bao danh: " ;
        cin >> ts[i].sobd ;
        cout << "Cac diem toan, ly, hoa: " ;
        cin >> ts[i].dt >> ts[i].dl >> ts[i].dh ;
        ts[i].td =ts[i].dt + ts[i].dl + ts[i].dh ;
    }
}
void TS::sapxep()
{
    int n = sots;
    for (int i=1; i< n; ++i)
        for (int j=i+1; j<= n; ++j)
            if (ts[i].td < ts[j].td)

```

```

        {
            TSINH tg = ts[i];
            ts[i] = ts[j];
            ts[j] = tg;
        }
    }
}

void TS::ghitep(char *ttep)
{
    ofstream f;
    f.open(ttep, ios::out | ios::noreplace);
    if (f.bad())
    {
        cout << "\nTep " << ttep << " da ton tai";
        cout << "\nCo ghi dc? - C/K";
        int ch=getch();
        if (toupper(ch)=='C')
        {
            f.close();
            f.open(ttep) ;
        }
        else
            exit(1);
    }
    f << sots ;
    f << setprecision(1) << setiosflags(ios::showpoint);
    for (int i=1; i<=sots; ++i)
    {
        f << endl << setw(24) << ts[i].ht << setw(20) <<
ts[i].ttinh ;
        f << endl << setw(6) << ts[i].sobd
        << setw(6) << ts[i].dt
        << setw(6) << ts[i].dl
        << setw(6) << ts[i].dh
        << setw(6) << ts[i].td ;
    }
    f.close();
}

void main()
{
    clrscr();
    TS t;
    t.nhap();
    t.ghitep("DS1.DL");
    t.sapxep();
    t.ghitep("DS2.DL");
    cout << "\n Hoan thanh";
    getch();
}

```

§12. ĐỌC DỮ LIỆU TỪ TẬP

12.1. Lớp ifstream

Để đọc dữ liệu từ tập chúng ta sử dụng lớp ifstream. Lớp ifstream thừa kế các phương thức của các lớp ios và istream. Nó cũng thừa kế phương thức:

`close`

của lớp fstreambase. Ngoài ra lớp ifstream có thêm các hàm tạo và các phương thức sau:

1. Hàm tạo:

```
ifstream() ; // Không đối
```

dùng để tạo một đối tượng ifstream (dòng nhập), chưa gắn với tập.

2. Hàm tạo:

```
ifstream(const char *fn, int mode = ios::in,  
          int prot = filebuf::openprot);
```

dùng để tạo một đối tượng ifstream, mở tập có tên fn để đọc và gắn đối tượng vừa tạo với tập được mở.

+ Tham số fn cho biết tên tập.

+ Tham số mode có giá trị mặc định là ios::in (mở để đọc). Tham số này có thể là một hợp của các giá trị sau:

ios::binary đọc theo kiểu nhị phân (mặc định theo kiểu văn bản)

ios::ate chuyển con trỏ tập tới cuối tập sau khi mở tập

+ Tham số thứ ba prot quy định cấp bảo vệ của dòng tin, tham số này có thể bỏ qua vì nó đã được gán một giá trị mặc định.

3. Hàm tạo:

```
ifstream(int fd);
```

dùng để tạo một đối tượng ifstream và gắn nó với một tập có chỉ số fd đang mở.

(Để mở và lấy chỉ số (số hiệu) tập có thể dùng hàm `_open`, xem cuốn Kỹ thuật Lập trình C của tác giả)

4. Hàm tạo:

```
ifstream(int fd, char *buf, int n);
```

dùng để tạo một đối tượng ifstream, gắn nó với một tập có chỉ số fd đang mở và sử dụng một vùng nhớ n byte do buf trỏ tới làm bộ đệm.

5. Phương thức:

```
void open(const char *fn, int mode = ios::in,  
          int prot = filebuf::openprot);
```

dùng để mở tệp có tên `fin` để đọc và gắn nó với đối tượng `ifstream`. Các tham số của phương thức có cùng ý nghĩa như trong hàm tạo thứ 2.

12.2. Các cách đọc tệp

Có 2 cách chính sau:

+ *Cách 1*: Dùng hàm tạo 2 để xây dựng một dòng nhập, mở một tệp để đọc và gắn tệp với dòng nhập. Sau đó dùng toán tử nhập `>>` và các phương thức để nhập dữ liệu từ dòng nhập vừa tạo như thẻ nhập dữ liệu từ `cin` (xem các mục trên)

+ *Cách 2*: Dùng hàm tạo 1 để xây dựng một dòng nhập. Sau đó dùng phương thức `open` để mở một tệp cụ thể và cho gắn với dòng nhập vừa xây dựng. Khi không cần làm việc với tệp nữa, chúng ta có thể dùng phương thức `close` để chấm dứt mọi ràng buộc giữa dòng nhập và tệp. Sau đó có thể gắn dòng nhập với tệp khác. Theo cách này, có thể dùng một dòng nhập (đối tượng `ifstream`) để nhập dữ liệu từ nhiều tệp khác nhau.

12.3. Kiểm tra sự tồn tại của tệp, kiểm tra cuối tệp

+ Khi mở một tệp để đọc mà tệp không tồn tại thì sẽ phát sinh lỗi, khi đó phương thức `bad` trả về giá trị khác không. Ví dụ để kiểm tra xem tệp `DSTS` (Danh sách thí sinh) đã tồn tại hay không có thể dùng đoạn chương trình:

```
ifstream fin("DSTS");
if (fin.bad())
{
    cout << "\nTep DSTS không tồn tại";
    exit(1);
}
```

+ Trong quá trình đọc, con trỏ tệp sẽ chuyển dần về cuối tệp. Khi con trỏ tệp đã ở cuối tệp (hết dữ liệu) mà vẫn thực hiện một lệnh đọc thì phương thức `eof` sẽ cho giá trị khác không. Chương trình dưới đây dùng phương thức `eof` để xác định độ dài (số byte) của tệp `TC.EXE` (chú ý cần dùng kiểu đọc nhị phân):

```
//CT7_14.CPP
// Đo dài tệp
#include <iostream.h>
#include <fstream.h>
#include <conio.h>
#include <stdlib.h>
void main()
{
```

```

clrscr();
long dd=0;
char ch;
ifstream f("TC.EXE",ios::in | ios::binary);
if (f.bad())
{
    cout << "\nTep TC.EXE khong ton tai";
    getch();
    exit(1);
}
while(f.get(ch),!f.eof()) fidd;
cout << "\n Do dai TC.EXE: " << dd;
getch();
}

```

12.4. Ví dụ

Chương trình dưới đây sẽ:

+ Đọc danh sách thí sinh từ tệp DS1.DL do chương trình trong mục §11 tạo ra.

+ In danh sách thí sinh vừa đọc.

+ Sắp xếp dãy thí sinh (vừa nhập từ tệp) theo thứ tự giảm của tổng điểm.

+ Ghi danh sách thí sinh sau khi sắp xếp lên tệp DS3.DL

+ Đọc danh sách thí sinh từ tệp DS3.DL

+ In danh sách thí sinh đọc từ tệp DS3.DL

Chương trình sử dụng lớp TS (Thí sinh) có 4 phương thức:

```

void xuất();
void sapxep();
void ghitep(char *ttep);
void doctep(char *ttep);
//CT7_12.CPP
// Đọc tệp
#include <iostream.h>
#include <iomanip.h>
#include <fstream.h>
#include <conio.h>
#include <stdlib.h>
#include <ctype.h>
struct TSINH
{
    char ht[25];
    char ttinh[21];
    int sobd;
    float dt,dl,dh,td;
} ;

```

```

class TS
{
private:
    int sots;
    TSINH *ts;
public:
    TS()
    {
        sots=0;
        ts = NULL;
    }
    void xuat();
    void sapxep();
    void ghitep(char *ttep);
    void doctep(char *ttep);
};

void TS::xuat()
{
    cout << "\n\nSo thi sinh: " << sots;
    cout << setprecision(1) << setiosflags(ios::showpoint);
    for (int i=1; i<=sots; ++i)
    {
        cout << "\nThi sinh thu: " << i ;
        cout << "\nlo ten: " << ts[i].ht ;
        cout << "\nTinh - thanh pho: " << ts[i].ttinh ;
        cout << "\nSo bao danh: " << ts[i].sobd ;
        cout << "\nCac diem toan, ly, hoa: "
            << setw(5) << ts[i].dt
            << setw(5) << ts[i].dl
            << setw(5) << ts[i].dh ;
        cout << "\nTong diem: " << ts[i].td ;
    }
}

void TS::sapxep()
{
    int n = sots;
    for (int i=1; i< n; ++i)
        for (int j=i+1; j<= n; ++j)
            if (ts[i].td < ts[j].td)
            {
                TSINH tg = ts[i];
                ts[i] = ts[j];
                ts[j] = tg;
            }
}

void TS::ghitep(char *ttep)
{
    ofstream f;
    f.open(ttep, ios::out|ios::noreplace);
    if (f.bad())
    {

```

```

cout << "\nTep " << ttep << " da ton tai";
cout << "\nCo ghi de? - C/K";
int ch=getch();
if (toupper(ch)=='C')
{
    f.close();
    f.open(ttep) ;
}
else
    exit(1);
}
f << sots ;
f << setprecision(1) << setiosflags(ios::showpoint);
for (int i=1; i<=sots; ++i)
{
    f << endl << setw(24) << ts[i].ht << setw(20) <<
ts[i].ttinh ;
    f << endl << setw(6) << ts[i].sobd
    << setw(6) << ts[i].dt
    << setw(6) << ts[i].dl
    << setw(6) << ts[i].dh
    << setw(6) << ts[i].td ;
}
f.close();
}
void TS::doctep(char *ttep)
{
    ifstream f;
    f.open(ttep);
    if (f.bad())
    {
        cout << "\nTep " << ttep << " khong ton tai";
        getch();
        exit(1);
    }
    f >> sots ;
    f.ignore();
    if (ts!=NULL) delete ts;
    ts = new TSINH[sots+1];
    for (int i=1; i<=sots; ++i)
    {
        f.get(ts[i].ht,25).get(ts[i].ttinh,21); ;
        f >> ts[i].sobd >> ts[i].dt >> ts[i].dl
        >> ts[i].dh >> ts[i].td ;
        f.ignore();
    }
    f.close();
}
void main()
{
    clrscr();

```

```

TS t;
t.doctep("DS1.DL");
t.xuat();
t.sapxep();
t.qhitep("DS3.DL");
t.doctep("DS3.DL");
t.xuat();
cout << "\n Hoàn thành";
getch();
}

```

§13. ĐỌC GHI ĐỒNG THỜI TRÊN TẬP

13.1. Lớp fstream

Để đọc ghi đồng thời trên tập, chúng ta sử dụng lớp fstream. Lớp fstream thừa kế các phương thức của các lớp ofstream và ifstream. Ngoài ra lớp fstream có các hàm tạo và phương thức sau:

1. Hàm tạo:

```
fstream() ; // Không đối
```

dùng để tạo một đối tượng fstream (dòng nhập-xuất), chưa gắn với tập.

2. Hàm tạo:

```
fstream(const char *fn, int mode,
        int prot = filebuf::openprot);
```

dùng để tạo một đối tượng fstream, mở tập có tên fn và gắn đối tượng vừa tạo với tập được mở.

+ Tham số fn cho biết tên tập.

+ Tham số mode quy định các kiểu truy nhập và có thể là tổ hợp của các giá trị sau:

ios::binary đọc-ghi theo kiểu nhị phân (mặc định theo kiểu văn bản).

ios::out ghi tập, nếu tập đã có thì nó bị xóa

ios::in đọc tập

ios::app ghi bổ sung vào cuối tập

ios::ate chuyển con trỏ tập về cuối sau khi mở

ios::trunc xóa nội dung của tập nếu nó tồn tại

ios::nocreate nếu tập chưa có thì không làm gì (bỏ qua)

ios::noreplace nếu tập đã có thì không làm gì (bỏ qua)

Chú ý:

+ Tham số mode không có giá trị mặc định.

+ Tham số thứ ba prot quy định cấp bảo vệ của dòng tin, tham số này có thể bỏ qua vì nó đã được gán một giá trị mặc định.

3. Hàm tạo:

```
fstream(int fd);
```

dùng để tạo một đối tượng fstream và gán nó với một tệp có chỉ số fd đang mở.

(Để mở và lấy chỉ số (số hiệu) tệp có thể dùng hàm _open, xem cuốn Kỹ thuật Lập trình C của tác giả)

4. Hàm tạo:

```
fstream(int fd, char *buf, int n);
```

dùng để tạo một đối tượng fstream, gán nó với một tệp có chỉ số fd đang mở và sử dụng một vùng nhớ n byte do buf trỏ tới làm bộ đệm.

5. Phương thức:

```
void open(const char *fn, int mode,  
          int prot = filebuf::openprot);
```

dùng để mở tệp có tên fn và gán nó với đối tượng fstream. Các tham số của phương thức có cùng ý nghĩa như trong hàm tạo thứ 2.

Chú ý: Tham số mode không có giá trị mặc định.

13.2. Các cách đọc-ghi đồng thời trên tệp

Có 2 cách chính sau:

+ *Cách 1:* Dùng hàm tạo 2 để xây dựng một dòng nhập-xuất, mở một tệp để đọc-ghi và gán tệp với dòng nhập-xuất. Sau đó dùng toán tử nhập >>, toán tử xuất << và các phương thức nhập, xuất để nhập, xuất dữ liệu ra dùng nhập-xuất vừa tạo (như đối với các dòng chuẩn cin và cout).

Ví dụ:

```
fstream f("DU_LIEU", ios::in | ios::out);
```

+ *Cách 2:* Dùng hàm tạo 1 để xây dựng một dòng nhập-xuất. Sau đó dùng phương thức open để mở một tệp cụ thể (để đọc và ghi) và cho gán với dòng nhập-xuất vừa xây dựng. Khi không cần làm việc với tệp này nữa, chúng ta có thể dùng phương thức close để chấm dứt mọi ràng buộc giữa dòng nhập-xuất và tệp. Sau đó có thể gán dòng nhập-xuất với tệp khác. Theo cách này, có thể dùng một dòng nhập-xuất (đối tượng fstream) để đọc-ghi dữ liệu từ nhiều tệp khác nhau.

Ví dụ:

```
fstream f;
```

```
f.open("DU_LIEU", ios::in | ios::out);
```

13.3. Di chuyển con trỏ tệp

13.3.1. Để di chuyển con trỏ tệp trên dòng xuất, chúng ta sử dụng các phương thức sau (của lớp ostream) :

1. Phương thức

`ostream& seekp(long n) ;`

sẽ chuyển con trỏ tệp tới vị trí (byte) thứ n (số thứ tự các byte tính từ 0).

2. Phương thức

`ostream& seekp(long offset, seek_dir dir) ;`

sẽ chuyển con trỏ tệp tới vị trí **offset** kể từ vị trí xuất phát **dir**. Giá trị của **offset** có thể âm, còn **dir** có thể nhận một trong các giá trị sau:

`ios::beg` xuất phát từ đầu tệp

`ios::end` xuất phát từ cuối tệp

`ios::cur` xuất phát từ vị trí hiện tại của con trỏ tệp

3. Phương thức

`long tellp() ;`

cho biết vị trí hiện tại của con trỏ tệp.

13.3.2. Để di chuyển con trỏ tệp trên dòng nhập, chúng ta sử dụng các phương thức sau (của lớp istream):

4. Phương thức

`istream& seekg(long n) ;`

sẽ chuyển con trỏ tệp tới vị trí (byte) thứ n (số thứ tự các byte tính từ 0)

5. Phương thức

`istream& seekg(long offset, seek_dir dir) ;`

sẽ chuyển con trỏ tệp tới vị trí **offset** kể từ vị trí xuất phát **dir**. Giá trị của **offset** có thể âm, còn **dir** có thể nhận một trong các giá trị sau:

`ios::beg` xuất phát từ đầu tệp

`ios::end` xuất phát từ cuối tệp

`ios::cur` xuất phát từ vị trí hiện tại của con trỏ tệp

6. Phương thức

`long tellg() ;`

cho biết vị trí hiện tại của con trỏ tệp.

13.3.3. Để di chuyển con trỏ tệp trên dòng nhập-xuất, chúng ta có thể sử dụng cả 6 phương thức nêu trên.

13.4. Ví dụ

Ví dụ 1. Trong §12 đã viết chương trình xác định độ dài của tệp TC.EXE. Dưới đây là một phương án khác đơn giản hơn:

```

fstream f("TC.EXE");
f.seekg(0,ios::end);
do_dai = f.tellg();

```

Ví dụ 2. Chương trình dưới đây sẽ nhập danh sách n thí sinh từ bàn phím và ghi lên tệp. Sau đó đưa con trỏ tệp về đầu tệp và bắt đầu đọc dữ liệu thí sinh từ tệp để in ra màn hình. Thông tin thí sinh gồm: Họ tên, tỉnh hoặc thành phố cư trú, số báo danh, các điểm toán lý hoá.

```

//CT7_13.CPP
// ghi - đọc đồng thời
#include <iostream.h>
#include <iomanip.h>
#include <fstream.h>
#include <conio.h>
#include <stdlib.h>
#include <ctype.h>
#include <stdio.h>
void main()
{
    char ht[25], ttinh[21], ttep[40];
    int sobd, stt ;
    float dt, dl, dh, td;
    fstream f;
    cout << "\nTen tep: " ;
    cin >> ttep;
    f.open(ttep, ios::out|ios::in|ios::noreplace);
    if (f.bad())
    {
        cout << "\nTep " << ttep << " da ton tai";
        cout << "\nCo ghi de? - C/K";
        int ch=getch();
        if (toupper(ch)=='C')
        {
            f.close();
            f.open(ttep, ios::out|ios::in|ios::trunc) ;
        }
        else
            exit(1);
    }
    stt=0 ;
    f << setprecision(1) << setiosflags(ios::showpoint);
    while(1)
    {
        ++stt;
        cout << "\nNhap thi sinh thu: " << stt ;
        cout << "\nHo ten (neu rong thi ket thuc nhap) : ";
        cin.ignore();
        cin.getline(ht, 25);
    }
}

```

```

    if (ht[0]==0) break;
    cout << "Tinh - thanh pho: ";
    cin.getline(ttinh,21);
    cout << "SoBD, diem toan, diem ly, diem hoa: " ;
    cin >> sobd >> dt >> dl >> dh ;
    td = dt + dl + dh ;
    if (stt>1) f << endl;
    f << setw(24) << ht << setw(20) << ttinh ;
    f << endl << setw(6) << sobd
      << setw(6) << dt
      << setw(6) << dl
      << setw(6) << dh
      << setw(6) << td ;
  }
  f.seekg(0);
  stt=0;
  clrscr();
  cout << "\nDanh sach thi sinh\n";
  cout << setprecision(1) <<
  setiosflags(ios::showpoint);
  while(1)
  {
    f.getline(ht,25).getline(ttinh,21);
    if (f.eof()) break;
    ++stt;
    f >> sobd >> dt >> dl >> dh >> td;
    f.ignore();
    cout << "\nThi sinh thu: " << stt ;
    cout << "\nlo ten: " << ht;
    cout << " \nTinh - thanh pho: " << ttinh;
    cout << "\nSo bao danh: " << sobd;
    cout << "\nDiem toan, ly, hoa va tong diem: "
      << setw(6) << dt << setw(6) << dl << setw(6) << dh
      << setw(6) << td ;
  }
  f.close();
  cout << "\n Hoan thanh";
  getch();
}

```

§14. XỬ LÝ LỖI

Khi làm việc với tệp không phải mọi việc đều trôi chảy mà thường xảy ra nhiều điều trục trặc, chẳng hạn:

1. Mở một tệp để đọc nhưng tệp không tồn tại.
2. Đọc dữ liệu nhưng con trỏ tệp đã ở cuối tệp
3. Ghi dữ liệu nhưng hết không gian đĩa (đĩa đầy).
4. Tạo tệp nhưng đĩa hỏng, hoặc đĩa cấm ghi hoặc đĩa đầy.

5. Dùng tên tệp không hợp lệ

6. Định thực hiện một thao tác nhưng tệp lại không được mở ở mode phù hợp để thực hiện thao tác đó.

Tóm lại khi làm việc với tệp thường gặp nhiều lỗi khác nhau, nếu không biết cách phát hiện xử lý thì chương trình sẽ dẫn đến rối loạn hoặc cho kết quả sai. Trong lớp ios của C++ có nhiều phương thức cho phép phát hiện lỗi khi làm việc với tệp. Đó là:

1. Phương thức

```
int eof() ;
```

Nếu con trỏ tệp đã ở cuối tệp mà lại thực hiện một lệnh đọc dữ liệu thì phương thức eof() trả về giá trị khác không, trái lại phương thức có giá trị bằng 0.

2. Phương thức

```
int fail() ;
```

Phương thức fail() trả về giá trị khác không nếu lần nhập xuất cuối cùng có lỗi, trái lại phương thức có giá trị bằng 0.

3. Phương thức

```
int bad() ;
```

Phương thức bad() trả về giá trị khác không khi một phép nhập xuất không hợp lệ hoặc có lỗi mà chưa phát hiện được, trái lại phương thức có giá trị bằng 0.

4. Phương thức

```
int good() ;
```

Phương thức good() trả về giá trị khác không nếu mọi việc đều tốt đẹp (không có lỗi nào xảy ra). Khi có một lỗi nào đó thì phương thức có giá trị bằng 0.

5. Phương thức

```
void clear() ;
```

dùng để tắt tất cả các bit lỗi.

Ví dụ 1. Khi mở tệp để đọc cần kiểm tra xem tệp có tồn tại không. Để làm điều đó, chúng ta có thể dùng đoạn chương sau:

```
char ten_tep[40] ;
cout << "\n Cho biết tên tệp: " ;
cin >> ten_tep ;
ifstream f(ten_tep);
if (f.bad())
{
    cout << "\n Tệp << ten_tep << "không tồn tại" ;
    exit(1) ;
}
```

Ví dụ 2. Khi tạo tệp mới để ghi cần kiểm tra xem có tạo được tệp hay không. Để làm điều đó, chúng ta có thể dùng đoạn chương sau:

```
char ten_tep[40] ;  
cout << "\n Cho biết tên tệp: " ;  
cin >> ten_tep ;  
ofstream f(ten_tep);  
if (f.bad())  
{  
    cout << "\n Không tạo được tệp << ten_tep ;  
    exit(1) ;  
}
```

Ví dụ 3. Để xác định độ dài của tệp, có thể dùng phương thức eof() và thuật toán sau:

- + Đọc một byte (chú ý phải đọc theo kiểu nhị phân)
 - + Nếu việc đọc thành công (eof()=0) thì cộng thêm một vào bộ đếm.
- Nếu việc đọc không thành (eof() != 0) thì kết thúc vòng lặp.

Thuật toán trên được thể hiện bằng đoạn chương trình sau:

```
ifstream f(ten_tep, ios::in | ios::binary) ;  
long dem = 0 ; char ch;  
while (1)  
{  
    f.get(ch) ;  
    if (!eof()) dem++ ;  
    else break;  
}
```

§15. NHẬP XUẤT NHỊ PHÂN

15.1. Chọn kiểu nhập xuất nhị phân

Kiểu nhập xuất mặc định là văn bản. Để chọn kiểu nhập xuất nhị phân, thì trong tham số mode (của hàm tạo dạng 2 và phương thức open) cần chứa giá trị:

ios::binary

Ví dụ muốn mở tệp "DSTS.DL" để đọc-ghi theo kiểu nhị phân và gắn tệp với dòng nhập-xuất fs, ta dùng câu lệnh sau:

```
fstream fs("DSTS.DL", ios::in | ios::out | ios::binary) ;
```

15.2. Đọc, ghi ký tự

- + Để ghi một ký tự lên tệp có thể dùng phương thức:

```
ostream & put(char) ;
```

+ Để đọc một ký tự từ tệp có thể dùng phương thức:

```
istream & get(char &) ;
```

Cần chú ý rằng: Cách đọc ghi ký tự theo kiểu văn bản khác với cách đọc ghi ký tự theo kiểu nhị phân (xem chương 10, cuốn Kỹ thuật lập trình C của tác giả)

Ví dụ để sao tệp có thể dùng thuật toán đơn giản sau:

+ Đọc một ký tự từ tệp nguồn

+ Nếu đọc thành công (phương thức `cof()` = 0) thì ghi lên tệp đích và lại tiếp tục đọc ký tự tiếp theo.

+ Nếu đọc không thành công (phương thức `eof()` != 0) thì kết thúc.

Chú ý là phải dùng kiểu nhập xuất nhị phân thì thuật toán mới cho kết quả chính xác. Chương trình sao tệp dưới đây viết theo thuật toán trên.

```
//CT7_15.CPP
// Sao tep
#include <iostream.h>
#include <fstream.h>
#include <conio.h>
#include <stdlib.h>
void main()
{
    clrscr();
    char tep_nguon[40], tep_dich[40] ;
    char ch;
    fstream fnguồn, fdich;
    cout << "\nTen tep nguon: " ; cin >> tep_nguon;
    cout << "\nTen tep dich: " ; cin >> tep_dich;
    fnguồn.open(tep_nguon, ios::in | ios::binary);
    fdich.open(tep_dich, ios::out | ios::binary);
    if (fnguồn.bad() || fdich.bad() )
    {
        cout << "\n Loi mo tep nguon hoac dich " ;
        getch();
        exit(1);
    }
    while(fnguồn.get(ch), !fnguồn.eof())
    fdich.put(ch) ;
    fnguồn.close();
    fdich.close();
    cout << "\nHoan thanh" ;
    getch();
}
```

15.3. Đọc, ghi một dãy ký tự theo kiểu nhị phân

+ Phương thức:

```
ostream & write(char *buf, int n) ;
```

sẽ xuất n ký tự (byte) chứa trong buf ra dòng xuất.

+ Phương thức:

```
istream & read(char *buf, int n) ;
```

sẽ nhập n ký tự (byte) từ dòng nhập và chứa vào buf.

+ Phương thức

```
int gcount
```

cho biết số ký tự thực sự đọc được trong phương thức read.

Chú ý: Các phương thức write, read chỉ làm việc một cách chính xác trong kiểu nhập-xuất nhị phân.

Dưới đây là chương trình sao tệp sử dụng các phương thức write, read và gcount.

```
//CT7_16.CPP
// Sao tệp dung write, read va gcount
#include <iostream.h>
#include <fstream.h>
#include <conio.h>
#include <stdlib.h>
void main()
{
    clrscr();
    char tep_ngon[40], tep_dich[40] ;
    char buf[5000];
    int n;
    fstream fngon, fdich;
    cout << "\nTen tep ngon: " ; cin >> tep_ngon;
    cout << "\nTen tep dich: " ; cin >> tep_dich;
    fngon.open(tep_ngon, ios::in | ios::binary);
    fdich.open(tep_dich, ios::out | ios::binary);
    if (fngon.bad() || fdich.bad() )
    {
        cout << "\n Loi mo tep ngon hoac dich " ;
        getch();
        exit(1);
    }
    while(fngon.read(buf, 5000), (n=fngon.gcount()))
        fdich.write(buf, n) ;
    fngon.close();
    fdich.close();
    cout << "\nHoan thanh" ;
    getch();
}
```

§16. ĐỌC GHI ĐỒNG THỜI THEO KIỂU NHỊ PHÂN

Chương trình dưới đây minh họa cách đọc ghi đồng thời trên tệp theo kiểu nhị phân. Chương trình sử dụng các phương thức write, read, các phương thức di chuyển con trỏ tệp và các phương thức kiểm tra lỗi. Chương trình gồm 3 chức năng:

1. Nhập một danh sách thí sinh mới và ghi vào tệp TS.DL
2. Bổ sung thí sinh vào tệp TS.DL
3. Xem sửa thí sinh trên tệp TS.DL

```
//CT7 18.CPP
// Doc tep
#include <iostream.h>
#include <iomanip.h>
#include <fstream.h>
#include <conio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#include <stdio.h>
struct TSINH
{
    char ht[25];
    int sobd;
    float td;
};
class TS
{
private:
    TSINH ts;
    char ten_tep[40];
    int sots;
    static int size;
public:
    TS(char *ttep);
    void tao_ds();
    void bo_sung();
    void xem_sua();
};
int TS::size = sizeof(TSINH);
TS::TS(char *ttep)
{
    strcpy(ten_tep, ttep);
    fstream f;
    f.open(ten_tep, ios::binary|ios::in|ios::ate);
    if (!f.good())
        sots = 0 ;
    else
```

```

    {
        sots=f.tellg()/size ;
    }
}
void TS::tao_ds()
{
    fstream f;
    f.open(ten_tep,ios::binary|ios::out|ios::noreplace);
    if (!f.good())
    {
        cout << "\nDanh sach da ton tai" ;
        cout << "\nCo tao lai khong? - C/K" ;
        char ch=getch();
        if (toupper(ch) != 'C')
            return;
        else
        {
            f.close();
            f.open(ten_tep,ios::binary|ios::out|ios::trunc);
        }
    }
    sots=0;
    while(1)
    {
        cout << "\nThi sinh thu: " << (sots+1) ;
        cout << "\nHo ten (Bam Enter de ket thuc): ";
        fflush(stdin);
        gets(ts.ht);
        if (ts.ht[0]==0) break;
        cout << "\nSo bao danh: ";
        cin >> ts.sobd;
        cout << "\nTong diem: ";
        cin >> ts.td;
        f.write((char*)&ts,size) ;
        sots++ ;
    }
    f.close();
}
void TS::bo_sung()
{
    fstream f;
    f.open(ten_tep,ios::binary|ios::app|ios::nocreate);
    if (!f.good())
    {
        cout << "\nDanh sach chua tao" ;
        cout << "\nCo tao moi khong? - C/K" ;
        char ch=getch();
        if (toupper(ch) != 'C')
            return;
        else
        {

```

```

        f.close();
        f.open(ten_tep, ios::binary|ios::out);
    }
}
int stt=0;
while(1)
{
    cout << "\nBo sung thi sinh thu: " << (stt+1);
    cout << "\nHo ten (Bam Enter de ket thuc): ";
    fflush(stdin);
    gets(ts.ht);
    if (ts.ht[0]==0) break;
    cout << "\nSo bao danh: ";
    cin >> ts.sobd;
    cout << "\nTong diem: ";
    cin >> ts.td;
    f.write((char*)&ts, size) ;
    ++stt;
}
sots += stt ;
f.close();
}
void TS::xem_sua()
{
    fstream f; int ch;
    f.open(ten_tep, ios::binary|ios::out|ios::in|ios::nocreate);
    if (!f.good())
    {
        cout << "\nDanh sach chua tao" ;
        getch();
        return ;
    }
    cout << "\nDanh sach gom: " << sots << "thi sinh" ; int stt;
    while(1)
    {
        cout << "\nCan xem-sua thi sinh thu (Bam 0 de ket
thuc): " ;
        cin >> stt ;
        if (stt<1 || stt > sots) break;
        f.seekg((stt-1)*size, ios::beg);
        f.read((char*)&ts, size);
        cout << "\nHo ten : " << ts.ht;
        cout << "\nSo ba danh: " << ts.sobd ;
        cout << "\nTong diem: " << ts.td ;
        cout << "\nCo sua khong? - C/K" ;
        ch=getch();
        if (toupper(ch)=='C')
        {
            f.seekg(-size, ios::cur) ;
            cout << "\nHo ten: ";

```

```

        fflush(stdin); gets(ts.ht);
        cout << "\nSo bao danh: ";
        cin >> ts.sobd;
        cout << "\nTong diem: ";
        cin >> ts.td;
        f.write((char*)&ts,size) ;
    }
}
f.close();
}
void main()
{
    int chon;
    clrscr();
    TS t("TS.DL");
    while(1)
    {
        clrscr();
        cout << "\n1. Tao danh sach thi sinh moi" ;
        cout << "\n2. Bo sung danh sach thi sinh" ;
        cout << "\n3. Xem-sua danh sach thi sinh" ;
        cout << "\n4. Ket thuc chuong trinh      " ;
        chon = getch();
        chon = chon - 48;
        clrscr();
        if (chon==1) t.tao_ds();
        else if(chon==2) t.bo_sung();
        else if(chon==3) t.xem_sua();
        else break;
    }
    clrscr();
    cout << "\n Hoan thanh";
    getch();
}

```

§17. XÂY DỰNG TOÁN TỬ NHẬP XUẤT ĐỐI TƯỢNG TRÊN TẬP

Trong các mục trên đã trình bày cách dùng các toán tử nhập >> và xuất << để ghi dữ liệu kiểu chuẩn (nguyên, thực, ký tự, chuỗi ký tự) trên tệp. Mục này trình bày cách xây dựng các toán tử dùng để đọc ghi các đối tượng của một lớp bất kỳ do người dùng định nghĩa.

Giả sử chúng ta muốn sử dụng các toán tử nhập xuất để đọc ghi các đối tượng của lớp TS. Khi đó ta đưa vào các hàm bạn toán tử nhập xuất như sau:

```

class TS
{

```

```

private:
    // Khai báo các thuộc tính
public:
    friend fstream& operator<<(fstream& fs,const TS &t);
    friend fstream& operator>>(fstream& fs,TS &t);
    ...
};

```

Về kiểu ghi: Có thể xây dựng các toán tử để thực hiện các phép đọc ghi theo kiểu văn bản cũng như nhị phân.

Ví dụ 1: Ghi theo kiểu văn bản

Chương trình dưới đây minh họa cách xây dựng và sử dụng các toán tử nhập xuất đối tượng trên màn hình, bàn phím và tệp. Chương trình đưa vào lớp TS (Thí sinh) và các hàm toán tử cho phép nhập xuất các đối tượng TS trên màn hình, bàn phím và tệp. Chương trình gồm các nội dung sau:

- + Tạo tệp TS.DL dùng để đọc và ghi theo kiểu văn bản.
- + Nhập 3 thí sinh từ bàn phím và chứa vào 3 biến đối tượng t1, t2, t3.
- + Ghi nội dung của 3 biến đối tượng t1, t2, t3 lên tệp TS.DL
- + Đọc các đối tượng từ tệp TS.DL và chứa vào 3 biến t4, t5, t6
- + In các biến đối tượng t4, t5, t6 ra màn hình
- + Chuyển con trỏ về đầu tệp, dùng chu trình while để lần lượt đọc các đối tượng từ tệp và in ra màn hình. Dùng phương thức eof để kiểm tra xem đã đọc hết dữ liệu hay chưa.

```

//CT7_17.CPP
// Các toán tử đọc ghi đối tượng trên Tệp
#include <iostream.h>
#include <iomanip.h>
#include <fstream.h>
#include <conio.h>
#include <stdlib.h>
#include <ctype.h>
class TS
{
private:
    char ht[25];
    float td;
public:
    friend ostream& operator<<(ostream& os,const TS &t);
    friend istream& operator>>(istream& is,TS &t);
    friend istream& operator<<(fstream& fs,const TS &t);
    friend fstream& operator>>(fstream& fs,TS &t);
};

```

```

fstream& operator>>(fstream& fs,TS &t)
{
    fs.getline(t.ht,25);
    fs >> t.td;
    fs.ignore();
    return fs;
}
ostream& operator<<(ostream& os,const TS &t)
{
    os << "\nHo ten: " << t.ht ;
    os << "\nTong diem: " << t.td;
    return os;
}
fstream& operator<<(fstream& fs,const TS &t)
{
    fs << t.ht << endl;
    fs << t.td << endl;
    return fs;
}
istream& operator>>(istream& is,TS &t)
{
    cout << "\nHo ten: " ;
    is.get(t.ht,25);
    cout << "Tong diem: " ;
    is >> t.td ;
    is.ignore();
    return is;
}
void main()
{
    clrscr();
    fstream f("TS.DL",ios::out | ios::in | ios::trunc);
    TS t1,t2,t3,t4,t5,t6,t;
    cin >> t1 >> t2 >> t3;
    f << t1 << t2 <<t3;
    f.seekg(0);
    f>>t4>>t5>>t6;
    cout << t4 << t5 << t6;
    f.seekg(0);
    while (f>>t,!f.eof())
        cout << t;
    f.close();
    cout << "\n Xong";
    getch();
}

```

Ví dụ 2 : Ghi theo kiểu nhị phân

Chương trình dưới đây cũng có các chức năng như chương trình trong ví dụ 1 bên trên, nhưng cách ghi đọc tệp theo kiểu nhị phân.

```

//CT7_19.CPP
// Các toán tử đọc ghi đối tượng trên tệp
// Kiểu nhi phân
#include <iostream.h>
#include <iomanip.h>
#include <fstream.h>
#include <conio.h>
#include <stdlib.h>
#include <ctype.h>
class TS
{
private:
    char ht[25];
    float td;
    static int size;
public:
    friend ostream& operator<<(ostream& os,const TS &t);
    friend istream& operator>>(istream& is,TS &t);
    friend fstream& operator<<(fstream& fs,const TS &t);
    friend fstream& operator>>(fstream& fs,TS &t);
};
int TS::size= sizeof(TS);
fstream& operator>>(fstream& fs,TS &t)
{
    fs.read( (char*)&t, t.size);
    return fs;
}
fstream& operator<<(fstream& fs,const TS &t)
{
    fs.write( (char*)&t, t.size);
    return fs;
}
ostream& operator<<(ostream& os,const TS &t)
{
    os << t.ht << endl;
    os << t.td << endl;
    return os;
}
istream& operator>>(istream& is,TS &t)
{
    cout << "\nHo ten: " ;
    is.get(t.ht,25);
    cout << "Tong diem: " ;
    is >> t.td ;
    is.ignore();
    return is;
}
void main()
{
    clrscr();
    fstream f("THU.DT",ios::binary | ios::out|ios::in|ios::trunc);
    TS t1,t2,t3,t4,t5,t6,t;

```

```

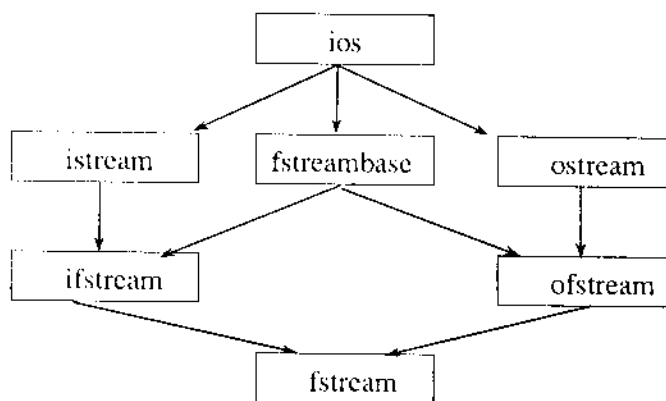
cin >> t1 >> t2 >> t3;
f << t1 << t2 << t3;
f.seekg(0);
f>>t4>>t5>>t6;
cout << t4 << t5 << t6;
f.seekg(0);
while( f>>t, !f.eof() )
    cout << t;
f.close();
cout << "\n Xong";
getch();
}

```

§18. HỆ THỐNG CÁC LỚP STREAM

Mục này hệ thống lại các lớp stream mà chúng ta đã sử dụng bên trên để tổ chức xuất nhập trên màn hình, bàn phím, máy in và tệp

18.1. Sơ đồ quan hệ giữa các lớp



18.2. Các phương thức của lớp ios

1. int bad()
2. void clear(int=0)
3. int eof()
4. int fail()
5. int fill()
6. int fill(char)
7. long flags()
8. long flags(long)
9. int good()
10. int precision()
11. int precision(int)
12. long setf(long)
13. long setf(long setbits, long field)
14. long unsetf(long)

- 15. `int width()`
- 16. `int width(int)`

18.3. Các phương thức của lớp `istream`

- 1. `operator>>`
- 2. `int gcount()`
- 3. `int get()`
- 4. `istream& get(char*, int, char = '\n')`
- 5. `istream& get(char&)`
- 6. `istream& getline(char*, int, char = '\n')`
- 7. `istream& ignore(int n = 1, int delim = EOF)`
- 8. `int peek()`
- 9. `istream& putback(char)`
- 10. `istream& read(char*, int)`
- 11. `istream& seekg(long)`
- 12. `istream& seekg(long, seek_dir)`
- 13. `long tellg()`

18.4. Các phương thức của lớp `ostream`

- 1. `operator<<`
- 2. `ostream& flush()`
- 3. `ostream& put(char)`
- 4. `ostream& seekp(long)`
- 5. `ostream& seekp(long, seek_dir)`
- 6. `long tellp()`
- 7. `ostream& write(char*, int)`

18.5. Các phương thức của lớp `fstreambase`

`void close()`

18.6. Các phương thức của lớp `ifstream`

- 1. `ifstream()`
- 2. `ifstream(const char*, int = ios::in, int = filebuf::openprot)`
- 3. `ifstream(int)`
- 4. `ifstream(int, char*, int)`
- 5. `void open(const char*, int = ios::in, int = filebuf::openprot)`

18.7. Các phương thức của lớp `ofstream`

- 1. `ofstream()`
- 2. `ofstream(const char*, int = ios::out, int = filebuf::openprot)`
- 3. `ofstream(int)`
- 4. `ofstream(int, char*, int)`
- 5. `void open(const char*, int = ios::out, int = filebuf::openprot)`

18.8. Các phương thức của lớp `fstream`

- 1. `fstream()`
- 2. `fstream(const char*, int, int = filebuf::openprot)`
- 3. `fstream(int)`
- 4. `fstream(int, char*, int)`
- 5. `void open(const char*, int, int = filebuf::openprot)`

ĐỒ HỌA

Trong chương này sẽ giới thiệu các hàm dùng để vẽ các đường và hình cơ bản như đường tròn, cung elip, hình quạt, đường gãy khúc, hình đa giác, đường thẳng, đường chữ nhật, hình chữ nhật, hình hộp chữ nhật,... Ngoài ra còn đề cập tới các vấn đề rất lý thú khác như: Xử lý văn bản trên màn hình đồ họa, cửa sổ và kỹ thuật tạo ảnh di động. Các hàm đồ họa được khai báo tệp graphics.h.

§1. KHÁI NIỆM ĐỒ HỌA

Để hiểu kỹ thuật lập trình đồ họa, đầu tiên phải hiểu các yếu tố cơ bản của đồ họa. Từ trước đến nay chúng ta chủ yếu làm việc với kiểu văn bản. Nghĩa là màn hình được thiết lập để hiển thị 25 dòng, mỗi dòng có thể chứa 80 ký tự. Trong kiểu văn bản, các ký tự hiển thị trên màn hình đã được phần cứng của máy PC ấn định trước và ta không thể nào thay đổi được kích thước, kiểu chữ.

Ở màn hình đồ họa, ta có thể xử lý đến từng chấm nhỏ (pixel) trên màn hình và do vậy muốn vẽ bất kỳ thứ gì cũng được. Sự bài trí và số pixel trên màn hình được gọi là độ phân giải (resolution). Do mỗi kiểu màn hình đồ họa có một cách xử lý đồ họa riêng nên TURBO C cung cấp một tệp tin điều khiển riêng cho từng kiểu đồ họa. Bảng 1 cho thấy các kiểu đồ họa và các tệp tin điều khiển chúng.

Ngoài các tệp có đuôi BGI chứa chương trình điều khiển đồ họa, TURBO C còn cung cấp các tệp tin đuôi CHER chứa các Font chữ để vẽ các kiểu chữ khác nhau trên màn hình đồ họa. Đó là các tệp:

GOTH.CHR

LITT.CHR

SANS.CHR

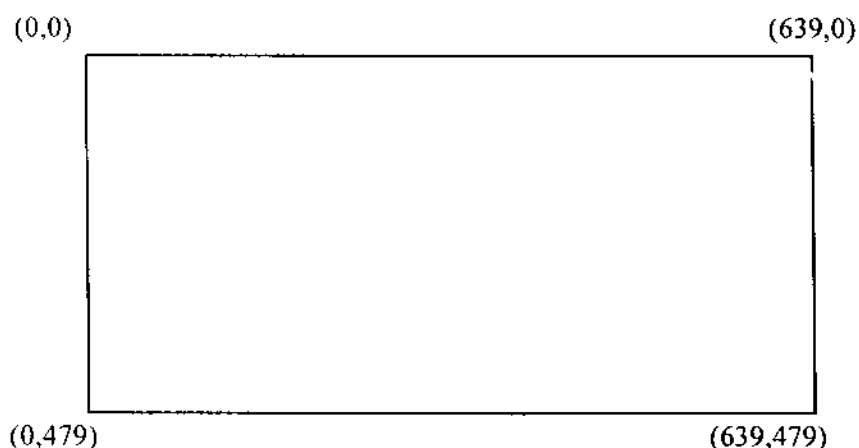
TRIP.CHR

Bảng 1. Các tệp tin điều khiển đồ họa của TURBO C

Tên tệp tin	Kiểu màn hình đồ họa
ATT.BGI	ATT & T6300 (400 dòng)
CGA.BGI	IBM CGA, MCGA và các máy tương thích
EGAVGA.BGI	IBM EGA, VGA và các máy tương thích
HERC.BGI	Hercules monochrome và các máy tương thích
IBM8514.BGI	IBM 8514 và các máy tương thích
PC3270.BGI	IBM 3270 PC

Màn hình đồ họa gồm nhiều điểm ảnh được sắp xếp trên các đường thẳng nằm ngang và thẳng đứng. Điều này đúng cho tất cả các kiểu màn hình đồ họa của máy tính. Khác biệt chủ yếu giữa chúng là kích thước và số các điểm ảnh. Trong kiểu CGA (độ phân giải thấp), điểm ảnh có kích thước lớn, chiều ngang có 320 điểm ảnh, còn theo chiều đứng có 200 điểm ảnh. Màn hình VGA có độ phân giải cao hơn: Điểm ảnh nhỏ hơn, trên mỗi hàng có 640 điểm ảnh và trên mỗi cột có 480 điểm ảnh. Điểm ảnh càng nhỏ thì số điểm ảnh trên màn hình càng nhiều và chất lượng đồ họa càng cao.

Mỗi kiểu đồ họa dùng một hệ tọa độ riêng. Hệ tọa độ cho màn hình VGA là 640 x 480 (hình 1)



Hình 1. Hệ tọa độ VGA

Nhờ hệ tọa độ này, ta có thể tác động hay tham chiếu đến bất kỳ điểm ảnh nào trên màn hình đồ họa.

Nếu dùng màn hình CGA thì góc phải dưới có tọa độ (319, 199). Độc lập với kiểu đồ họa đang sử dụng, các hàm getmaxx và getmaxy bao giờ cũng cho tọa độ x và y lớn nhất trong kiểu đồ họa đang dùng.

Một chương trình đồ họa thường gồm các phần sau:

- Khởi động hệ thống đồ họa.
- Xác định mẫu nền (màu màn hình), mẫu đường vẽ, mẫu tô và kiểu (mẫu) tô.
- Vẽ, tô mẫu các hình mà ta mong muốn.
- Các thao tác đồ họa khác như cho hiện các dòng chữ...
- Đóng hệ thống đồ họa để trở về mode văn bản.

§2. KHỞI ĐỘNG HỆ ĐỒ HỌA

Mục đích của việc khởi động hệ thống đồ họa là xác định thiết bị đồ họa (màn hình) và một đồ họa sẽ sử dụng trong chương trình. Để làm điều này ta dùng hàm

```
void initgraph(int *graphdriver, int *graphmode, char *driverpath);
```

trong đó: driverpath là đường dẫn của thư mục chứa các tệp tin điều khiển đồ họa, graphdriver, graphmode cho biết màn hình và một đồ họa sẽ sử dụng trong chương trình. Bảng 2 cho các giá trị khả dĩ của graphdriver và graphmode.

Ví dụ 1: Giả sử máy tính của ta có màn hình EGA, các tệp tin đồ họa chứa trong thư mục C:\TC, khi đó ta có thể khởi động hệ thống đồ họa như sau:

```
#include "graphics.h"
void main()
{
    int mh=EGA, mode= EGALO;
    initgraph(&mh, &mode, "C:\TC");
    . . .
}
```

Bảng 2. Các giá trị khả dĩ của graphdriver, graphmode

graphdriver	graphmode	Độ phân giải
Detect (0)		
CGA (1)	CGAC0 (0)	320 x 200
	CGAC1 (1)	320 x 200
	CGAC2 (2)	320 x 200
	CGAC3 (3)	320 x 200
	CGAHi (4)	640 x 200

MCGA (2)	MCGA0 (0)	320 x 200
	MCGA1 (1)	320 x 200
	MCGA2 (2)	320 x 200
	MCGA3 (3)	320 x 200
	MCGAMed (4)	640 x 200
	MCGAHi (5)	640 x 480
EGA (3)	EGALO (0)	640 x 200
	EGAHi (1)	640 x 350
EGA64 (4)	EGA64LO (0)	640 x 200
	EGA64Hi (1)	640 x 350
EGAMONO (5)	EGAMONOHl (0)	640 x 350
VGA (9)	VGALO (0)	640 x 200
	VGAMED (1)	640 x 350
	VGAHI (2)	640 x 480
HERCMONO (7)	HERCMONOHl	720 x 348
ATT400 (8)	ATT400C0 (0)	320 x 200
	ATT400C1 (1)	320 x 200
	ATT400C2 (2)	320 x 200
	ATT400C3 (3)	320 x 200
	ATT400MED (4)	640 x 400
	ATT400HI (5)	640 x 400
PC3270 (10)	PC3270HI (0)	720 x 350
IBM8514 (6)	IBM8514LO (0)	640 x 480, 256 màu
	IBM8514HI (1)	1024 x 768, 256 màu

Chú ý 1: Bảng 2 cho các hằng và giá trị của chúng mà các biến graphdriver, graphmode có thể nhận. Chẳng hạn hằng DETECT có giá trị 0, hằng VGA có giá trị 9, hằng VGALO có giá trị 0... Khi lập trình ta có thể dùng tên hằng hoặc giá trị tương ứng của chúng. Chẳng hạn các phép gán trong ví dụ 1 có thể viết theo một cách khác tương đương như sau:

```
mh=3;
```

```
mode=0;
```

Chú ý 2: Bảng 2 cho thấy độ phân giải phụ thuộc cả vào màn hình và mode. Ví dụ trong màn hình EGA nếu dùng mode EGALO thì độ phân giải là 640 x 200, hàm getmaxx cho giá trị 639, hàm getmaxy cho giá trị 199.

Nếu cũng màn hình EGA mà dùng mode EGAHI thì độ phân giải là 640x350, hàm getmaxx cho giá trị 639, hàm getmaxy cho giá trị 349.

Chú ý 3: Nếu không biết chính xác kiểu màn hình đang sử dụng thì ta gán cho biến graphdriver hằng DETECT hay giá trị 0. Khi đó kết quả của hàm initgraph sẽ là:

- Kiểu của màn hình đang sử dụng được phát hiện, giá trị số của nó được gán cho biến graphdriver.

- Mode đồ họa ở độ phân giải cao nhất ứng với màn hình đang sử dụng cũng được phát hiện và giá trị số của nó được gán cho biến graphmode.

Như vậy việc dùng hằng số DETECT chẳng những có thể khởi động được hệ thống đồ họa của màn hình hiện có theo mode có độ phân giải cao nhất, mà còn giúp ta xác định chính xác kiểu màn hình đang sử dụng.

Ví dụ 2: Chương trình dưới đây xác định kiểu màn hình đang sử dụng.

```
#include "graphics.h"
#include "stdio.h"
#include "conio.h"
void main()
{
    int mh=0, mode= 0;
    initgraph(&mh, &mode, "");
    closegraph();
    printf("\n Giá trị số của màn hình là: %d", mh);
    getch();
}
```

Nếu chương trình cho kết quả:

Giá trị số của màn hình là: 3

thì ta có thể khẳng định loại màn hình đang dùng là EGA.

Chú ý 4: Nếu chuỗi dùng để xác định driverpath là một chuỗi rỗng (như trong ví dụ 2) thì chương trình dịch sẽ tìm các tệp điều khiển đồ họa trên thư mục chủ (thư mục hiện hành).

§3. LỖI ĐỒ HỌA

Khi khởi động hệ thống đồ họa nếu máy không tìm thấy các chương trình điều khiển đồ họa thì sẽ phát sinh lỗi đồ họa và việc khởi động coi như không thành. Lỗi đồ họa còn phát sinh khi dùng các hàm. Trong mọi trường hợp, hàm graphresult cho biết có lỗi hay không lỗi và đó là lỗi gì. Bảng 3 cho các mã lỗi mà hàm này phát hiện được. Ta có thể dùng hàm

grapherrormsg với mã lỗi do hàm graphresult trả về để biết được đó là lỗi gì, ví dụ:

```
int maloi;
maloi = graphresult();
printf("\nLỗi đồ họa là: %s", grapherrormsg(maloi));
```

Bảng 3. Các mã lỗi của graphresult

Hằng	Trị	Lỗi phát hiện
grOk	0	Không có lỗi
grNoInitGraph	-1	Chưa khởi động hệ đồ họa
grNotDetected	-2	Không có phần cứng đồ họa
grFileNotFound	-3	Không tìm thấy trình điều khiển đồ họa
grInvalidDriver	-4	Trình điều khiển không hợp lệ
grNoLoadMem	-5	Không đủ RAM cho đồ họa
grNoScanMem	-6	Vượt vùng RAM trong Scan fill
grNoFloodMem	-7	Vượt vùng RAM trong flood fill
grFontNotFound	-8	Không tìm thấy tập tin Font
grNoFontMem	-9	Không đủ RAM để nạp Font
grInvalidMode	-10	Kiểu đồ họa không hợp lệ cho trình điều khiển
grError	-11	Lỗi đồ họa tổng quát
grIOerror	-12	Lỗi đồ họa vào ra
grInvalidFont	-13	Tập tin Font không hợp lệ
grInvalidFontNum	-14	Số hiệu Font không hợp lệ

§4. MÀU VÀ MẪU

Dưới đây là các hàm để chọn màu và mẫu.

1. Để thiết lập màu nền ta sử dụng hàm

```
void setbkcolor(int color);
```

2. Để thiết lập màu đường vẽ ta dùng hàm

```
void setcolor(int color);
```

3. Để thiết lập mẫu (kiểu) tô và màu tô ta dùng hàm

```
void setfillstyle(int pattern, int color);
```

Trong cả 3 trường hợp color xác định mã của mẫu. Các giá trị khả dĩ của color cho trong bảng 4, pattern xác định mã của mẫu tô (xem bảng 5).

Mẫu tô và mẫu tô sẽ được sử dụng trong các hàm picslice, fillpoly, bar, bar3d và floodfill (xem §5 dưới đây).

4. Chọn giải mẫu: Để thay đổi giải mẫu đã được định nghĩa trong bảng 4 ta dùng hàm

```
void setpalette(int colornum, int color);
```

Ví dụ câu lệnh

```
setpalette(0, Lightcyan);
```

biến mẫu đầu tiên trong bảng mẫu thành xanh lơ nhạt (Lightcyan). Các mẫu khác không bị ảnh hưởng.

Bảng 4. Các giá trị khả dĩ của color

Tên hằng	Giá trị số	Mẫu hiển thị
BLACK	0	đen
BLUE	1	Xanh da trời
GREEN	2	Xanh lá cây
CYAN	3	Xanh lơ
RED	4	Đỏ
MAGENTA	5	Tím
BROWN	6	Nâu
LIGHTGRAY	7	Xám nhạt
DARKGRAY	8	Xám sẫm
LIGHTBLUE	9	Xanh da trời nhạt
LIGHTGREEN	10	Xanh lá cây nhạt
LIGHTCYAN	11	Xanh lơ nhạt
LIGHTRED	12	Đỏ nhạt
LIGHTMAGENTA	13	Tím nhạt
YELLOW	14	Vàng
WHITE	15	Trắng

5. Để nhận giải mẫu hiện hành ta dùng hàm

```
void getpalette (struct palettetype *palette);
```

ở đây palettetype là kiểu đã định nghĩa trước như sau:

```
#define MAXCOLORS 15
```

```

struct palettetype
{
    unsigned char size;
    unsigned char colors[MAXCOLORS+1];
};

```

ở đây: size là số lượng màu trong palette, colors là mảng chứa màu với chỉ số mảng chạy từ 0 đến size - 1.

Bảng 5. Các giá trị khả dĩ của pattern

Tên hằng	Giá trị số	Mô tả kiểu tô
EMPTY_FILL	0	Tô bằng màu nền
SOLID_FILL	1	Tô bằng đường nét liền
LINE_FILL	2	Tô bằng - - -
LTSLASH_FILL	3	Tô bằng ///
SLASH_FILL	4	Tô bằng /// in đậm
BKSLASH_FILL	5	Tô bằng \\\ in đậm
LTBKSLASH_FILL	6	Tô bằng \\
HATCH_FILL	7	Tô bằng đường gạch bóng nhật
XIHATCH_FILL	8	Tô bằng đường gạch bóng chữ thập
INTERLEAVE_FILL	9	Tô bằng đường đứt quãng
WIDE_DOT_FILL	10	Tô bằng dấu chấm thưa
CLOSE_DOT_FILL	11	Tô bằng dấu chấm mau

6. Hàm getcolor trả về màu đã xác định trước đó bằng hàm setcolor.

7. Hàm getbkcolor trả về màu đã xác định trước đó bằng hàm setbkcolor.

8. Hàm getmaxcolor trả về số lượng màu cực đại thuộc giải màu hiện đang có hiệu lực. Trên 256 K EGA, hàm getmaxcolor luôn cho giá trị 15.

§5. VẼ VÀ TÔ MÀU

Có thể chia các đường và hình thành bốn nhóm chính:

- Đường tròn và hình tròn
- Đường gấp khúc và hình đa giác
- Đường thẳng
- Hình chữ nhật

5.1. Đường tròn và hình tròn

Nhóm này gồm cung tròn, đường tròn, cung ellipse và hình quạt.

1. Cung tròn: Để vẽ một cung tròn ta dùng hàm

```
void arc(int x, int y, int gd, int gc, int r);
```

ở đây:

(x, y) là tọa độ của tâm cung tròn

r là bán kính

gd là góc đầu

gc là góc cuối

Chú ý: Trong tất cả các hàm dưới đây, góc tính theo độ và có giá trị từ 0 đến 360.

2. Đường tròn: Để vẽ một đường tròn ta dùng hàm

```
void circle(int x, int y, int r);
```

ở đây:

(x, y) là tọa độ của tâm

r là bán kính đường tròn

3. Cung ellipse: Để vẽ một cung Ellipse ta dùng hàm

```
void ellipse(int x, int y, int gd, int gc, int xr, int yr);
```

ở đây:

(x, y) là tọa độ của tâm cung Ellipse

gd là góc đầu

gc là góc cuối

xr là bán trục ngang

yr là bán trục đứng

4. Hình quạt: Để vẽ và tô màu một hình quạt ta dùng hàm

```
void pieslice(int x, int y, int gd, int gc, int r);
```

ở đây:

(x,y) là tọa độ tâm hình quạt

gd là góc đầu

gc là góc cuối

r là bán kính

Ví dụ 1: Chương trình dưới đây sẽ vẽ: Một cung tròn ở góc phần tư thứ nhất, một cung ellipse ở góc phần tư thứ ba, một đường tròn và một hình quạt quét từ 90 đến 360 độ.

```

#include <graphics.h>
#include <conio.h>
main()
{
    int mh, mode;
    /* Khởi động đồ họa, màn hình EGA, mode EGALO */
    mh=EGA;
    mode=EGALO;
    initgraph(&mh, &mode, "C:\\TC\\BGI");
    /* Mẫu nền Green, mẫu đường vẽ
       White, mẫu tô Red, kiểu tô SlashFill */
    setbkcolor (GREEN);
    setcolor (WHITE);
    setfillstyle (SLASH_FILL, RED);
    /* Vẽ: một cung tròn ở góc phần tư thứ nhất,
       một cung Ellipse ở góc phần tư thứ ba,
       một đường tròn, một quạt tròn */
    arc(160, 50, 0, 90, 45);
    ellipse(480, 50, 180, 270, 150, 45);
    circle(160, 150, 45);
    pieslice(480, 150, 90, 360, 45);
    getch();
    /* Kết thúc chế độ đồ họa */
    closegraph();
}

```

5.2. Đường gấp khúc và đa giác

1. Muốn vẽ một đường gấp khúc đi qua n điểm: (x1,y1), ..., (xn,yn) thì trước hết ta phải đưa các tọa độ vào một mảng a nào đó kiểu int. Nói một cách chính xác hơn, cần gán x1 cho a[0], y1 cho a[1], x2 cho a[2], y2 cho a[3]... Sau đó ta viết lời gọi hàm:

```
drawpoly(n, a);
```

Khi điểm cuối (xn, yn) trùng với điểm đầu (x1, y1) ta nhận được một đường gấp khúc khép kín.

2. Giả sử a là mảng đã nói trong điểm 1, khi đó lời gọi hàm

```
fillpoly(n, a);
```

sẽ có tác dụng vẽ và tô màu một đa giác có đỉnh là các điểm

(x1, y1), ..., (xn, yn).

Ví dụ 2: Chương trình dưới đây sẽ vẽ một đường gấp khúc và hai hình tam giác.

```

#include <graphics.h>
#include <conio.h>
    /* Xây dựng các mảng chứa tọa độ các đỉnh */

```

```

int poly1[]={5,200,190,5,100,300};
int poly2[]={205,200,390,5,300,300};
int poly3[]={405,200,590,5,500,300,405,200};
main()
{
    int mh=0, mode=0;
    initgraph(&mh, &mode, "");
    /* Màu nền CYAN, màu đường vẽ
    YELLOW, màu tô MAGENTA, mẫu tô SolidFill */
    setbkcolor (CYAN);
    Setcolor (YELLOW);
    setfillstyle (SOLID_FILL, MAGENTA);
    /* Đường gấp khúc */
    drawpoly (3, poly1);
    /* Hình đa giác */
    fillpoly (3, poly2);
    /* Hình đa giác */
    fillpoly(4, poly3);
    getch();
    closegraph();
}

```

5.3. Đường thẳng

1. Hàm line

```
void line(int x1,int y1,int x2,int y2);
```

vẽ đường thẳng nối hai điểm (x1, y1) và (x2, y2) nhưng không làm thay đổi vị trí con trỏ.

2. Hàm lineto

```
void lineto(int x,int y);
```

vẽ đường thẳng từ điểm hiện tại tới điểm (x, y) và chuyển con trỏ đến điểm (x, y).

3. Hàm linerel

```
void linerel(int dx,int dy);
```

vẽ một đường thẳng từ vị trí hiện tại (x, y) của con trỏ đến điểm (x + dx, y + dy). Con trỏ được di chuyển đến vị trí mới.

4. Hàm moveto

```
void moveto(int x,int y);
```

sẽ di chuyển con trỏ tới vị trí (x, y).

Ví dụ 3: Chương trình dưới đây tạo lên một đường gấp khúc bằng các đoạn thẳng. Đường gấp khúc đi qua các đỉnh sau: (20,20), (620, 20), (620, 180), (20, 180) và (320, 100).

```

#include <graphics.h>
#include <conio.h>
main()
{
    int mh=0, mode=0;
    initgraph(&mh, &mode, "");
    setbkcolor(GREEN);
    setcolor(YELLOW);
    moveto(320,100);
    line(20,20,620,20);
    linerel(-300,80);
    lineto(620,180);
    lineto(620,20);
    getch();
    closegraph();
}

```

5.4. Hình chữ nhật

1. Hàm rectangle

```
void rectangle(int x1,int y1,int x2,int y2);
```

sẽ vẽ một đường chữ nhật có các cạnh song song với các cạnh của màn hình. Tọa độ đỉnh trên bên trái của hình chữ nhật là (x1,y1) và điểm dưới bên phải là (x2,y2).

2. Hàm bar

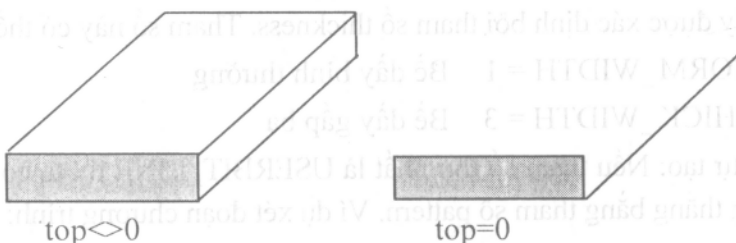
```
void bar(int x1,int y1,int x2,int y2);
```

sẽ vẽ và tô màu một hình chữ nhật. Các giá trị x1, y1, x2 và y2 có ý nghĩa như đã nói trong điểm 1.

3. Hàm bar3d

```
void bar3d(int x1,int y1,int x2,int y2,int depth,int top);
```

sẽ vẽ một khối hộp chữ nhật, mặt ngoài của nó là hình chữ nhật xác định bởi các tọa độ x1,y1,x2,y2 (như đã nói trong điểm 2). Hình chữ nhật này được tô màu. Tham số depth ấn định số điểm ảnh trên bề sâu của khối 3 chiều. Tham số top cho biết khối hộp có nắp hay không: Khi $top > 0$ là có nắp, $top = 0$ là không nắp (xem hình vẽ).



Ví dụ 4: Chương trình dưới đây sẽ vẽ một đường chữ nhật, một hình chữ nhật và một khối hộp chữ nhật có nắp.

```
#include <graphics.h>
#include <conio.h>
main()
{
    int mh=0, mode=0;
    initgraph(&mh, &mode, "");
    setbkcolor(GREEN);
    setcolor(RED);
    setfillstyle(CLOSE_DOT_FILL, YELLOW);
    rectangle(5, 5, 300, 160);
    bar(5, 175, 300, 340);
    bar3d(320, 100, 500, 340, 100, 1);
    getch();
    closegraph();
}
```

§6. CHỌN KIỂU ĐƯỜNG

Mục này giới thiệu 3 hàm

1. Hàm setlinestyle

`void setlinestyle(int linestyle, int pattern, int thickness);`
tác động đến nét vẽ của các thủ tục `line`, `lineto`, `rectangle`, `drawpoly`, `circle`,...
Hàm này cho phép ta ấn định 3 yếu tố của đường thẳng là dạng, bề dày và mẫu tự tạo.

+ Dạng đường được xác định bởi tham số `linestyle`. Sau đây là các giá trị khả dĩ của `linestyle` và dạng đường thẳng tương ứng.

<code>SOLID_LINE = 0</code>	Nét liền
<code>DOTTED_LINE = 1</code>	Nét chấm
<code>CENTER_LINE = 2</code>	Nét chấm gạch
<code>DASHED_LINE = 3</code>	Nét gạch
<code>USERBIT_LINE = 4</code>	Mẫu tự tạo

+ Bề dày được xác định bởi tham số `thickness`. Tham số này có thể là:

<code>NORM_WIDTH = 1</code>	Bề dày bình thường
<code>THICK_WIDTH = 3</code>	Bề dày gấp ba

+ Mẫu tự tạo: Nếu tham số thứ nhất là `USERBIT_LINE` thì ta có thể tạo ra mẫu đường thẳng bằng tham số `pattern`. Ví dụ xét đoạn chương trình:

```
int pattern= 0x1010;
setlinestyle(USERBIT_LINE, pattern, NORM_WIDTH);
line(0,0,100,200);
```

Giá trị của pattern trong hệ 16 là 1010 hay trong hệ 2 là

```
0001 0000 0001 0000
```

Chỗ nào có bit 1 điểm ảnh sẽ sáng, bit 0 làm tắt điểm ảnh.

2. Để nhận các giá trị hiện hành của 3 yếu tố trên ta dùng hàm:

```
void getlinesettings(struct linesettingstype *lineinfo);
```

với kiểu linesettingstype đã được định nghĩa trước như sau:

```
struct linesettingstype
{
    int linestyle;
    unsigned int upattern;
    int thickness;
};
```

Ví dụ 1: Chương trình dưới đây minh họa cách dùng các hàm setlinestyle và getlinesettings để vẽ đường thẳng.

```
/* Kiểu đường */
#include <graphics.h>
#include <conio.h>
main()
{
    struct linesettingstype kieu_cu;
    int mh=0, mode=0;
    initgraph(&mh, &mode, "");
    if (graphresult!= grOk)
        exit(1);
    setbkcolor(GREEN);
    setcolor(RED);
    line(0,0,100,0);
    /* Lưu lại kiểu hiện tại */
    getlinesettings(kieu_cu);
    /* Thiết lập kiểu mới */
    setlinestyle(DOTTED_LINE,0,THICK_WIDTH);
    line(0,0,100,10);
    /* Phục hồi kiểu cũ */
    setlinestyle(kieu_cu.linestyle,
    kieu_cu.upattern, kieu_cu.thickness);
    line(0,20,100,20);
    getch();
    closegraph();
}
```

3. Hàm setwritemode

```
void setwritemode( int writemode);
```

sẽ thiết lập kiểu thể hiện đường thẳng cho các hàm line, drawpoly, linerel, lineto, rectangle. Kiểu thể hiện được xác định bởi tham số writemode:

- Nếu writemode bằng COPY_PUT = 0, thì đường thẳng được viết đè lên dòng đang có trên màn hình.

- Nếu writemode bằng XOR_PUT = 1, thì màu của đường thẳng định vẽ sẽ kết hợp với màu của từng chấm điểm của đường hiện có trên màn hình theo phép toán XOR (Chương 2, §15) để tạo lên một đường thẳng mới.

Một ứng dụng của XOR_PUT là: Khi thiết lập kiểu writemode bằng XOR_PUT rồi vẽ lại đường thẳng cùng màu thì sẽ xóa đường thẳng cũ và trả về màn hình nguyên thủy.

Chương trình dưới đây minh họa cách dùng hàm setwritemode. Khi thực hiện ta sẽ thấy hình chữ nhật thu nhỏ dần vào tâm màn hình.

Ví dụ 2:

```
/* Thu hình; */
#include <graphics.h>
#include <conio.h>
main()
{
    struct linesettingstype kieu_cu;
    int mh=0, mode=0, x1, y1, x2, y2;
    initgraph(&mh, &mode, "");
    if (graphresult!= grOk)
        exit(1);
    setbkcolor(GREEN);
    setcolor(RED);
    setfillstyle(CLOSE_DOT_FILL, YELLOW);
    x1=0;
    y1=0;
    x2=getmaxx();
    y2=getmaxy();
    setwritemode(XOR_PUT);
    /* Vẽ hình chữ nhật */
    tt: rectangle(x1,y1,x2,y2);
    if ( (x1+1)<(x2-1) && (y1+1)<(y2-1) )
    {
        /* xóa hình chữ nhật */
        rectangle(x1,y1,x2,y2);
        x1=x1+1;
        y1=y1+1;
        /* co hình chữ nhật */
        x2=x2-1;
        y2=y2-1;
        goto tt;
    }
```

```

    }
    /* Trờ về overwrite mode */
    setwritemode(COPY_PUT);
    closegraph();
}

```

§7. CỬA SỐ (VIEWPORT)

1. Viewport

Là một vùng chữ nhật trên màn hình đồ họa tựa như window trong textmode. Để thiết lập viewport ta dùng hàm

```
void setviewport(int x1,int y1,int x2,int y2,int clip);
```

trong đó (x1,y1) là tọa độ góc trên bên trái và (x2,y2) là tọa độ góc dưới bên phải. Bốn giá trị này phải thỏa mãn:

$$0 \leq x1 \leq x2$$

$$0 \leq y1 \leq y2$$

Tham số clip có thể nhận một trong hai giá trị:

clip = 1 không cho phép vẽ ra ngoài viewport

clip = 0 Cho phép vẽ ra ngoài viewport.

Ví dụ câu lệnh

```
setviewport(100,50,200,150, 1);
```

sẽ thiết lập viewport. Sau khi lập viewport ta có hệ tọa độ mới mà góc trên bên trái của viewport sẽ có tọa độ (0,0).

2. Để nhận viewport hiện hành ta dùng hàm

```
void getviewsettings(struct viewporttype *vp);
```

ở đây kiểu viewporttype đã được định nghĩa như sau:

```

struct viewporttype
{
    int left, top, right, bottom;
    int clip;
};

```

3. Để xóa viewport ta dùng hàm

```
void clearviewport(void);
```

4. Để xóa màn hình và đưa con chạy về tọa độ (0,0) của màn hình ta dùng hàm

```
void cleardevice(void);
```

Chú ý: Câu lệnh này sẽ xóa mọi thứ trên màn hình.

5. Tọa độ âm dương:

Nhờ sử dụng Viewport có thể viết các chương trình đồ họa theo tọa độ âm dương. Muốn vậy ta thiết lập viewport sao cho tâm (điểm giữa) màn hình là góc trên bên trái của viewport và cho clip = 0 để có thể vẽ ra ngoài giới hạn của viewport. Sau đây là đoạn chương trình thực hiện công việc trên

```
int xc, yc;
xc= getmaxx()/2; yc= getmaxy()/2;
setviewport(xc, yc, getmaxx(), getmaxy(), 0);
```

Như thế màn hình sẽ được chia làm 4 phần với tọa độ âm dương như sau:

Phần tư trái trên x âm, y âm

Phần tư trái dưới x âm, y dương

Phần tư phải trên x dương, y âm

Phần tư phải dưới x dương, y dương

Chương trình dưới đây vẽ đồ thị hàm $\sin(x)$ trong hệ trục tọa độ âm dương. Hoành độ x lấy các giá trị từ $-4*PI$ đến $4*PI$. Trong chương trình có dùng hai hàm mới là: outtextxy và putpixel (xem các mục sau).

Ví dụ 1:

```
/* Đồ thị hàm sin; */
#include <graphics.h>
#include <conio.h>
#include <math.h>
#define SCALEX 20
#define SCALEY 60
main()
{
    int mh=0, mode=0, x, y, i;
    initgraph(&mh, &mode, "");
    if (graphresult!= grOk)
        exit(1);
    setviewport(getmaxx()/2, getmaxy()/2,
        getmaxx(), getmaxy(), 0);
    /* Kc hệ trục tọa độ */
    setcolor(BLUE);
    line(-(getmaxx()/2), 0, getmaxx()/2, 0);
    line(0, -(getmaxy()/2), 0, getmaxy()/2);
    settextjustify(1, 1);
    setcolor(RED);
    outtextxy(0, 0, "(0,0)");
    for (i = -400; i <= 400; ++i)
    {
```

```

        x=round(2*M_PI*i*SCALEX/200);
        y=round(sin(2*M_PI*i/200)*SCALEY);
        putpixel(x,y,YELLOW);
    }
    getch();
    closegraph();
}

```

Ví dụ 1 tạo lên một đồ thị từ các chấm điểm. Bây giờ ta sửa ví dụ 1 đôi chút: giữ nguyên từ đầu đến outtextxy, thay phần cuối bởi đoạn chương trình dưới đây. Ta sẽ được đồ thị liên tục gồm các đoạn thẳng ghép lại với nhau.

Ví dụ 2:

```

/* Phần đầu giống ví dụ 1 */
setcolor(YELLOW);
for (i=-400;i<=400;++i)
{
    x=round(2*M_PI*i*SCALEX/200);
    y=round(sin(2*M_PI*i/200)*SCALEY);
    if(i== -400)
        moveto(x,y);
    else
        lineto(x,y);
}
getch();
closegraph();

```

§8. TÔ ĐIỂM, TÔ MIỀN

1. Hàm putpixel

```
void putpixel(int x, int y, int color);
```

sẽ tô điểm (x,y) theo mẫu xác định bởi color.

2. Hàm getpixel

```
unsigned getpixel(int x, int y);
```

sẽ trả về số hiệu màu của điểm ảnh ở vị trí (x,y).

Chú ý: nếu điểm này chưa được tô mẫu bởi các hàm vẽ hoặc putpixel (mà chỉ mới được tạo mẫu nền bởi setbkcolor) thì hàm cho giá trị bằng 0. Vì vậy có thể dùng hàm này theo mẫu dưới đây để xác định các nét vẽ trên màn hình đồ hoạ và vẽ ra giấy.

```

if (getpixel(x,y)!=0)
{
    /* Điểm (x,y) được vẽ, đặt một chấm điểm ra giấy */
}

```

3. Tô miền. Để tô màu cho một miền nào đó trên màn hình ta dùng hàm:

```
void floodfill(int x, int y, int border);
```

ở đây:

(x,y) là tọa độ của một điểm nào đó gọi là điểm gieo.

tham số border chứa mã của một màu.

Sự hoạt động của hàm floodfill phụ thuộc vào giá trị của x, y, border và trạng thái màn hình.

a) Khi trên màn hình có một đường (cong hoặc gấp khúc) khép kín mà mã màu của nó bằng giá trị của border thì:

+ Miền giới hạn bởi đường kín sẽ được tô màu nếu điểm gieo (x,y) nằm bên trong miền này.

+ Nếu (x,y) nằm bên ngoài thì phần màn hình bên ngoài miền đóng nói trên được tô màu.

b) Khi trên màn hình không có một đường nào như vậy, thì cả màn hình được tô màu.

Ví dụ 1: Chương trình dưới đây sẽ vẽ một đường tròn đỏ trên màn hình xanh. Tọa độ (x,y) của điểm gieo được nạp vào từ bàn phím. Tùy thuộc vào giá trị cụ thể của x, y, chương trình sẽ tô màu vàng cho hình tròn hoặc phần màn hình bên ngoài hình tròn.

```
#include <graphics.h>
#include <stdio.h>
main()
{
    int mh=0, mode=0, x, y;
    initgraph(&mh, &mode, "");
    if (graphresult!= grOk)
        exit(1);
    setbkcolor(GREEN);
    setcolor(RED);
    setfillstyle(11, YELLOW);
    circle(320, 100, 50);
    moveto(1, 150);
    outtext(" Tọa độ điểm gieo x,y ");
    scanf("%d%d", &x, &y);
    flooddfill(x, y, RED);
    getch();
    closegraph();
}
```

Ví dụ 2: Minh họa cách dùng hàm putpixel và hàm getpixel để vẽ các điểm ảnh và sau đó xóa các điểm ảnh. Muốn kết thúc chương trình bấm ESC

```

#include <conio.h>
#include <graphics.h>
#include <stdio.h>
#include <stdlib.h>
int seed = 1962; /* Nhân cho bộ tạo
    số ngẫu nhiên */
int numpts = 2000; /* Vẽ 2000 chấm điểm */
int ESC = 27;
void putpixelplay(void);
main()
{
    int mh=0, mode=0;
    initgraph(&mh, &mode, "");
    if (graphresult() != grOk)
    {
        exit(1);
    }
    putpixelplay();
    closegraph();
}
void putpixelplay(void)
{
    int i,x,y,color,xmax,ymax,maxcolor,ch;
    struct viewporttype v;
    getviewsettings(&v);
    xmax=(v.right - v.left -1);
    ymax=(v.bottom - v.top -1);
    maxcolor=getmaxcolor();
    while (!kbhit())
    {
        /* Vẽ các chấm điểm một
        cách ngẫu nhiên */
        srand(seed);
        i=0;
        while(i<=numpts)
        {
            ++i;
            x=random(xmax)+1;
            y=random(ymax)+1;
            color=random(maxcolor);
            putpixel(x,y,color);
        }
        /* Xóa các điểm ảnh */
        srand(sced);
        i=0;
        while(i<=numpts)

```

```

        {
            ++i;
            x= random(xmax) + 1;
            y= random(ymax) + 1;
            color=random(maxcolor);
            putpixel(x,y,0);
        }
    if(kbhit())
    {
        ch=getch();
        if(ch==ESC)
            break;
    }
}
} /* Kết thúc hàm putpixelplay */

```

§9. XỬ LÝ VĂN BẢN TRÊN MÀN HÌNH ĐỒ HOẠ

9.1. Hiện thị văn bản trên màn hình đồ hoạ

Hàm outtext

```
void outtext (char *s);
```

sẽ cho hiện chuỗi ký tự (do s trỏ tới) tại vị trí hiện tại của con trỏ.

Hàm outtextxy

```
void outtextxy(int x,int y,char *s);
```

sẽ cho hiện chuỗi ký tự (do s trỏ tới) tại vị trí (x,y).

Ví dụ 1: Hai cách sau đây sẽ cho cùng kết quả

```
outtextxy (100,100,"Chao ban");
```

và

```
moveto (100,100);
outtext ("Chao ban");
```

Chú ý: Trong một đồ hoạ vẫn cho phép hàm vào dữ liệu scanf và các hàm bắt phím getch, kbhit.

9.2. Fonts

Như đã nêu các Fonts nằm trong các tệp tin .CHR trên đĩa. Các Font này cho các kích thước và kiểu chữ khác nhau sẽ hiện thị trên màn hình đồ hoạ bằng các hàm outtext, outtextxy. Để chọn và nạp Font chúng ta dùng hàm:

```
void settextstyle(int font,int direction,int charsize);
```

Chú ý: hàm chỉ có tác dụng nếu tồn tại các tệp .CHR

+ Tham số direction có thể nhận một trong hai giá trị:

HORIZ_DIR = 0

VERT_DIR = 1

Nếu direction = HORIZ_DIR, văn bản sẽ hiển thị theo chiều nằm ngang từ trái sang phải. Nếu direction = VERT_DIR, văn bản sẽ hiển thị theo chiều đứng từ dưới lên trên.

+ Tham số charsize là hệ số phóng to ký tự và có giá trị trong khoảng từ 1 đến 10.

- Nếu charsize = 1. Font được thể hiện trong hình chữ nhật 8*8 pixel.

- Nếu charsize = 2. Font được thể hiện trong hình chữ nhật 16*16 pixel.

...

- Nếu charsize = 10, Font được thể hiện trong hình chữ nhật 80*80 pixel.

+ Tham số font dùng để chọn kiểu chữ và nhận một trong các giá trị sau:

DEFAULT_FONT = 0

TRIPLEX_FONT = 1

SMALL_FONT = 2

SANS_SERIF_FONT = 3

GOTHIC_FONT = 4

Các giá trị do settextstyle thiết lập sẽ dữ nguyên cho đến khi gọi một settextstyle mới.

Ví dụ 2:

```
settextstyle (3, VERT_DIR, 2);
```

```
outtextxy (50, 50, " HELLO ");
```

9.3. Vị trí hiển thị

Hàm settextjustify quy định vị trí văn bản hiển thị (trong các hàm outtext và outtextxy) so với điểm mốc (x,y).

Hàm này có dạng

```
void settextjustify(int horiz, int vert);
```

Tham số horiz có thể là một trong các hằng số sau:

LEFT_TEXT = 0 (Điểm mốc bên trái văn bản)

CENTER_TEXT = 1 (Điểm mốc ở giữa văn bản theo chiều ngang)

RIGHT_TEXT = 2 (Điểm mốc bên phải văn bản)

Tham số Vert có thể là một trong các hằng số sau:

BOTTOM_TEXT = 0 (Điểm mốc ở đáy văn bản)

CENTER_TEXT = 1 (Điểm mốc ở giữa văn bản theo chiều dọc)

TOP_TEXT = 2 (Điểm mốc ở đỉnh văn bản)

Ví dụ 3:

```
settextjustify(1,1);  
outtextxy(100,100,"ABC");
```

Kết quả là điểm (100,100) sẽ nằm giữa chữ B.

9.4. Bề rộng và bề cao của văn bản

Hàm textheight

```
void textheight (char *s);
```

trả về chiều cao (theo pixel) của chuỗi do s trở tới. Với 8*8 bit map Font và hệ số khuyếch đại chữ là 1 thì

```
textheight ("H") = 8
```

Ví dụ 4: Đoạn chương trình dưới đây sẽ cho hiện 5 dòng chữ.

```
#include <graphics.h>  
#include <conio.h>  
main()  
{  
    int mh=0,mode=0,y,size;  
    initgraph(&mh,&mode,"");  
    y=10;  
    settextjustify(0,0);  
    for(size=1; size<=5; ++size)  
    {  
        settextstyle(0,0,size);  
        outtextxy(0,y,"GRAPHICS");  
        y += textheight("GRAPHICS") + 10;  
    }  
    getch();  
    closegraph();  
}
```

Hàm textwidth

```
void textwidth(char *s);
```

sẽ dựa vào chiều dài của chuỗi, kích thước Font chữ, hệ số khuyếch đại chữ để trả về bề rộng (theo pixel) của chuỗi do s trở tới.

Ví dụ 5: Trong chương trình dưới đây sẽ lập các hàm vào ra trên màn hình đồ hoạ.

```
#include <graphics.h>  
#include <conio.h>  
#define Enter 13  
#define Lmargin 10
```

```

void text_write(int *x,int *y,char *s);
void text_writeln(int *x,int *y,char *s);
void text_read(int *x,int *y,char *s);
void text_write(int *x,int *y,char *s)
{
    outtextxy(*x,*y,s);  *x += textwidth(s);
}
void text_writeln(int *x,int *y,char *s)
{
    outtextxy(*x,*y,s);
    *x+= lmargin;
    *y += textheight(s)+5;
}
void text_read(int *x,int *y,char *s)
{
    int i=0;
    char ch[2];
    ch[1]=0;
    while(1)
    {
        ch[0]=getch();
        if(ch[0]==Enter) break;
        text_write(x,y,ch);
        s[i]=ch[0]; ++i;
    }
    s[i]=0;
}
main()
{
    int mh=0,mode=0,x,y,xmax,ymax;
    char name[25];
    initgraph(&mh,&mode,"");
    settextstyle(TRIPLEX_FONT,HORIZ_DIR,3);
    x=lmargin;
    y=100;
    text_write (&x,&y,"cho ten cua ban: ");
    text_read (&x,&y,name);
    text_writeln (&x,&y,"");
    text_write(&x,&y,"chao ban ");
    text_write(&x,&y,name);
    getch();
    closegraph();
}

```

§10. CẮT HÌNH, DÁN HÌNH VÀ TẠO ẢNH CHUYỂN ĐỘNG

1. Hàm imagesize

unsigned imagesize(int x1,int y1,int x2,int y2)

trả về số byte cần thiết để lưu trữ ảnh trong phạm vi hình chữ nhật (x1,y1,x2,y2).

2. Hàm malloc

```
#include <alloc.h>
void *malloc(unsigned n);
```

trả về con trỏ trỏ tới một vùng nhớ n byte mới được cấp phát.

3. Hàm getimage

```
void getimage(int x1,int y1,int x2,int y2,void *bitmap);
```

sẽ chép các điểm ảnh của hình chữ nhật (x1,y1,x2,y2) và các thông tin về bề rộng, cao của hình chữ nhật vào vùng nhớ do bitmap trỏ tới. Vùng nhớ và biến bitmap cho bởi hàm malloc. Độ lớn của vùng nhớ được xác định bằng hàm imagesize.

4. Hàm putimage

```
void putimage(int x,int y,void *bitmap,int copymode);
```

dùng để sao ảnh lưu trong vùng nhớ bitmap ra màn hình tại vị trí (x,y). Tham số copymode xác định kiểu sao chép ảnh, nó có thể nhận các giá trị sau:

COPY_PUT = 0	Sao chép nguyên bản
XOR_PUT = 1	Các điểm ảnh trong bitmap kết hợp với các điểm ảnh trên màn hình bằng phép XOR
OR_PUT = 2	Các điểm ảnh trong bitmap kết hợp với các điểm ảnh trên màn hình bằng phép OR
AND_PUT = 3	Các điểm ảnh trong bitmap kết hợp với các điểm ảnh trên màn hình bằng phép AND
NOT_PUT = 4	ảnh xuất hiện trên màn hình theo dạng đảo ngược (phép NOT) với ảnh trong bitmap.

Nhận xét: Nếu dùng mode XOR_PUT để sao hình, rồi lặp lại đúng câu lệnh đó thì hình sẽ bị xóa và màn hình trở lại như cũ. Kỹ thuật này dùng để tạo lên các hình ảnh chuyển động.

Ví dụ 1: Chương trình dưới đây minh họa cách dùng imagesize, malloc, getimage và putimage.

```
#include <alloc.h>
#include <graphics.h>
main()
{
    int mh=0,mode=0;
    char *p;
    unsigned size;
    initgraph (&mh,&mode,"");
```

```

bar(0,0,getmaxx(),getmaxy());
size = imagesize(10,20,30,40);
p=(char*)malloc(size); /* p trỏ tới
    vùng nhớ size byte mới được cấp phát */
getimage (10,20,30,40,p); getch();
cleardevice();
putimage (100,100,p,COPY_PUT);
getch();
closegraph();
}

```

5. Tạo ảnh chuyển động

Nguyên tắc tạo ảnh chuyển động giống như phim hoạt hình:

- Vẽ một hình (trong chuỗi hình mô tả chuyển động)
- Delay
- Xoá hình đó
- Vẽ hình kế theo
- Delay

A) Vẽ hình

Cách 1: Vẽ lại một ảnh nhưng tại các vị trí khác nhau.

Cách 2: Lưu ảnh vào một vùng nhớ rồi đưa ảnh ra màn hình tại các vị trí khác nhau.

B) Xoá ảnh

Cách 1: Dùng hàm cleardevice

Cách 2: Dùng hàm putimage (mode XOR_PUT) để xếp chồng lên ảnh cần xoá.

Cách 3: Lưu trạng thái màn hình vào một chỗ nào đó. Vẽ một hình ảnh. Đưa trạng thái cũ màn hình ra xếp đè lên ảnh vừa vẽ.

Kỹ thuật tạo ảnh chuyển động được minh hoạ trong các chương trình của §11.

§11. MỘT SỐ CHƯƠNG TRÌNH ĐỒ HỌA

Chương trình 1: Đầu tiên vẽ bầu trời đầy sao. Sau đó từng chùm pháo hoa được bắn lên bầu trời. Khi bấm phím Enter thì việc bắn pháo hoa kết thúc, ta nhận lại bầu trời đầy sao. Bấm tiếp Enter thì kết thúc chương trình.

```

/* Bắn pháo hoa trên bầu trời đầy sao */
#include <graphics.h>
#include <conio.h>

```

```

#include <stdlib.h>
#include <alloc.h>
main()
{
    int x[101],y[101];
    int mh=0,mode=0,i,n;
    char *p[101];
    initgraph(&mh,&mode,"");
    if(graphresult()!=0)
        exit(1);
    setcolor(RED);
    /* Vẽ bầu trời đầy sao */
    for(i=1;i<=1000;++i)
    {
        putpixel(random(getmaxx()),
            random(getmaxy()),random(getmaxcolor()));
    }
    /*Lưu hiện trạng 100 hình chữ nhật
    trên màn hình để khôi phục*/
    for(i=1;i<=100;++i)
    {
        x[i]=random(getmaxx())-10;
        y[i]=random(getmaxy())-10;
        if(x[i]<0) x[i]=0;
        if(y[i]<0) y[i]=0;
        n=imagesize(x[i],y[i],x[i]+10,y[i]+10);
        p[i]=(char*)malloc(n);
        getimage(x[i],y[i],x[i]+10,y[i]+10,p[i]);
    }
    /* Chu trình bắn pháo hoa */
    do
    {
        /* Đưa 100 quả pháo lên màn hình
        tại các vị trí quy định */
        for(i=1;i<=100;++i)
        {
            setfillstyle(SOLID_FILL,i%15+1);
            pieslice(x[i]+5,y[i]+5,0,360,5);
        }
        delay(500);
        /*Xóa chùm pháo hoa vừa bắn
        bằng cách khôi phục màn hình*/
        for(i=100;i>=1;--i)
            putimage(x[i],y[i],p[i],COPY_PUT);
        delay(500);
    }
    while(!kbhit());
    getch();
    getch();
    closegraph();
}

```

Chương trình 2: Vẽ đồng hồ có 3 kim giờ, phút và giây. Đồng hồ chạy đúng theo giờ hệ thống. Muốn kết thúc chương trình bấm ↵.

```

/*
Đồng hồ
*/
#include <graphics.h>
#include <conio.h>
#include <math.h>
#include <dos.h>

/*
Hàm kẻ đoạn thẳng từ tâm đồng hồ theo độ,
chiều dài, độ dày và màu
*/
void ke(int ddo, unsigned dai,
        unsigned day, unsigned mau);
/* Kẻ kim giây khi biết số giây */
void ke_giay(unsigned giay);
/* Kẻ kim phút khi biết số phút */
void ke_phut(unsigned phut);
/* Kẻ kim giờ khi biết số giờ */
void ke_gio(unsigned gio, unsigned phut);
void chay_kim_giay(void);
void chay_kim_phut(void);
void chay_kim_gio(void);
int x0, y0, rgio, rphut, rgiay, mgio, mphut, mgiay;
unsigned phutgioht, gioht, phutht, giayht;
void ke(int ddo, unsigned dai,
        unsigned day, unsigned mau)
{
    unsigned x, y; float goc;
    while (ddo >= 360)
        ddo = ddo - 360;
    goc = (M_PI / 180) * ddo;
    x = x0 + (int) (dai * cos(goc) + 0.5);
    y = y0 - (int) (dai * sin(goc) + 0.5);
    setcolor(mau);
    setlinestyle(0, 0, day);
    line(x0, y0, x, y);
}

/* Hàm ke kim giây*/
void ke_giay(unsigned giay)
{
    int ddo;
    ddo = (90 - 6 * giay);
    ke(ddo, rgiay, 1, mgiay);
}

/* Hàm ke kim phút*/
void ke_phut(unsigned phut)
{
    int ddo;

```

```

        ddo= (90-6*phut);
        ke(ddo,rphut,3,mphut);
    }
/* Hàm ke kim gio*/
void ke_gio(unsigned gio, unsigned phut)
{
    int ddo;
    ddo = 360 + 90 - 30*(gio%12) - (phut+1)/2;
    ke(ddo,rgio,3,mgio);
}
/* Hàm chỉnh giây hiện tại và làm chuyển động kim giây */
void chay_kim_giay(void)
{
    unsigned giay; struct time t;
    gettimeofday(&t);
    giay=t.ti_sec;
    if (giay!=giayht)
    {
        ke_giay(giayht);
        giayht=giay;
        ke_giay(giayht);
    }
}
/* Hàm chỉnh phút hiện tại và làm chuyển động kim phút */
void chay_kim_phut(void)
{
    unsigned phut;
    struct time t;
    gettimeofday(&t);
    phut=t.ti_min;
    if(phut!=phutht)
    {
        ke_phut(phutht);
        phutht=phut;
        ke_phut(phutht);
    }
}
/*Hàm chỉnh giờ phút hiện tại và làm chuyển động kim giờ*/
void chay_kim_gio(void)
{
    unsigned h,gio,phut,sophut,sophutht;
    struct time t;
    gettimeofday(&t);
    gio=t.ti_hour;
    phut=t.ti_min;
    sophut = gio*60+phut;
    sophutht = gioht*60+phutgioht;
    if( sophut<sophutht)
        sophut=sophut+ 12*60;
    h=sophut-sophutht;
    if(h>=12)

```

```

        {
            ke_gio(gioht,phutgioht);
            phutgioht=phut;
            gioht=gio;
            ke_gio(gioht,phutgioht);
        }
    }

main()
{
    struct time t;
    char*dso[]={ "", "12", "1", "2", "3", "4", "5", "6",
        "7", "8", "9", "10", "11" };
    int i, mh=0, mode=0, r, x, y;
    float goc;
    initgraph(&mh, &mode, "");
    x0=(getmaxx()/2)-1;
    y0=(getmaxy()/2)-1;
    r=y0-2;
    rgiay = r-10;
    rphut=r-50;
    rgio=r-90;
    mgiay= BROWN;
    mphut=RED; /* mgio:=magenta;*/
    mgio=YELLOW;
    /* Vẽ chu vi đồng hồ */
    setcolor(BLUE);
    setlinestyle(0,0,3);
    circle(x0,y0,r);
    setfillstyle(1,YELLOW);
    floodfill(0,0,BLUE);
    setfillstyle(1,WHITE);
    floodfill(x0,y0,BLUE);
    setlinestyle(0,0,1);
    circle(x0,y0,10);
    setfillstyle(1,GREEN);
    floodfill(x0,y0,BLUE);
    setttextjustify(1,1);
    setcolor(MAGENTA);
    outtextxy(x0,y0+120,"IBM-JIMIKO");
    /* Ghi chữ số */
    setttextstyle(3,0,3);
    setttextjustify(1,1);
    setcolor(BLUE);
    for(i=1;i<=12;++i)
    {
        goc=(2*M_PI+M_PI/2) - (i-1)*(M_PI/6);
        x = x0+ (int)(rphut*cos(goc)+0.5);
        y = y0- (int)(rphut*sin(goc)+0.5);
        outtextxy(x,y,dso[i]);
    }
    /* Xác định thời điểm đầu */

```

```

    gettime(&t);
    gioht=t.ti_hour;
    phutht=t.ti_min;
    giayht=t.ti_sec;
    phutgioht=phutht;
    setwritemode(XOR_PUT);
    /* Vẽ kim giờ, phut, giây */
    ke_gio(gioht, phutgioht);
    ke_phut(phutht);
    ke_giay(giayht);
    /* Làm chuyển động các kim */
do
{
    chay_kim_giay();
    chay_kim_phut();
    chay_kim_gio();
}
while(!kbhit());
closegraph();
}

```

Chương trình 3: Vẽ một con tàu vũ trụ bay trên bầu trời đầy sao theo quỹ đạo ellipse. Trong khi tàu chuyển động thì các ngôi sao thay nhau nhấp nháy

```

/* Tàu vũ trụ chuyển động trên bầu trời
đầy sao nhấp nháy */
#include <graphics.h>
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
#include <stdlib.h>
#include <math.h>

/* Khai báo các hàm trong chương trình */
void tau_cd(void); /* tàu chuyển động */
void nhap_nhay_bt(void); /* sao nhấp nháy
trên bầu trời */
void main(void); /* hàm main */
/* Khai báo các biến mảng ngoài */
int a,b,x,y,x0,y0;
int mh=0,mode=0,n,i;
float goc,xt,yt;
char *p;
int xx[100],yy[100];
/* Hàm main */
void main(void)
{
    initgraph(&mh,&mode,"");
    if(graphresult()!=0)

```

```

        exit(1);
        /* Vẽ tàu vũ trụ */
setcolor(RED);
ellipse(100,50,0,360,20,8);
ellipse (100,46,190,357,20,6);
line(107,44,110,38);
circle(110,38,2);
line(93,44,90,38);
circle(90,38,2);
setfillstyle(SOLID_FILL,BLUE);
floodfill(101,54,RED);
setfillstyle(SOLID_FILL,MAGENTA);
floodfill(94,45,RED);
        /* Lưu ảnh của tàu vũ trụ vào bộ nhớ */
n=imagesize(79,36,121,59);
p=(char*)malloc(n);
getimage(79,36,121,59,p);
        /*
Vẽ bầu trời đầy sao và lưu vị trí của chúng
vào các mảng xx, yy để phục vụ hàm nhap_nhay_bt
*/
cleardevice();
for(i=1;i<=1000;++i)
{
    xx[i]=random(getmaxx());
    yy[i]=random(getmaxy());
    putpixel(xx[i],yy[i],
        random(getmaxcolor()));
}
        /* Xác định giá trị ban đầu cho các biến
dùng để điều khiển chuyển động tàu */
goc= 2*M_PI + M_PI/2;
x0= (getmaxx() - 42)/2;
y0= (getmaxy() - 25)/2;
a=x0;
b=y0;
        /* chu trình tàu vũ trụ chuyển động và
        các ngôi sao nhấp nháy*/
do
{
    tau_cd();
    nhap_nhay_bt();
}
while(!kbhit());
getch();
closegraph();
}
}
void tau_cd(void)
{
    xt=a*cos(goc)+x0;

```

```

yt=-b*sin(goc)+y0;
x=(int)(xt+0.5);
y=(int)(yt+0.5);
    /* Đặt tàu vũ trụ lên màn hình */
putimage(x,y,p,XOR_PUT);
delay(500);
    /* Xóa */
putimage(x,y,p,XOR_PUT);
    /* Thay đổi góc để làm cho tàu chuyển động */
goc += M_PI/30;
if (goc<M_PI/2)
    goc=2*M_PI+M_PI/2;
}
void nhap_nhay_bt(void)
{
    static i=1; /* Lệnh này thực hiện
        một lần khi dịch */
    int j;
        /* Cho nhấp nháy bằng cách đổi màu
            50 ngôi sao */
    for(j=1;j<=50;++j)
    {
        putpixel(xx[i],yy[i],
            random(getmaxcolor()));
        ++i;
        if(i>1000)
            i=1;
    }
}

```

§12. IN ẢNH TỪ MÀN HÌNH ĐỒ HỌA

Hàm `in_anh` dưới đây sẽ in ảnh trong miền chữ nhật (`xt`, `yt`, `xd`, `yd`) của màn hình đồ họa ra giấy trên các máy in LQ1070, LQ1170 và FX1050.

```
void in_anh(int dd,int xt,int yt,int xd,int yd);
```

Tham số `dd` là độ đậm của nét in. Thực chất `dd` là số lần in lại. Bình thường chọn `dd=1`. Nếu muốn in rõ hơn ta chọn `dd` bằng 2 hay 3.

Trong hàm `in_anh` có dùng hàm `tao_mau`, nó được mô tả như sau:

```
int tao_mau(int k,int x,int y);
```

Hàm này sẽ dò trên `k` chấm điểm theo chiều dọc bắt đầu từ tọa độ (`x`,`y`) trên màn hình để biết xem chấm điểm nào đã tô màu. Hàm sẽ trả về một giá trị nguyên tạo bởi các bit 1 (ứng với điểm đã tô màu) và 0 (ứng với điểm chưa tô màu).

Hàm `in_anh` sẽ dùng hàm `tao_mau` để duyệt trên miền chữ nhật (`xt`,`yt`,`xd`,`yd`). Mỗi lần duyệt sẽ nhận được một mẫu các chấm điểm (giá trị nguyên) và mẫu này được in ra giấy.

Dưới đây là nội dung của 2 hàm nói trên.

```
/* in ảnh c*/
```

```
#include "stdio.h"
#include "graphics.h"
int tao_mau(int k,int x,int y);
void in_anh(int dd,int xt,int yt,int xd,int yd);
int tao_mau(int k,int x,int y)
{
    int c=0,i;
    for(i=0;i<k;++i)
        if(getpixel(x,y+i))
            c = c | (128>>i);
    return c;
}
void in_anh(int dd,int xt,int yt,int xd,int yd)
{
    /*dd - so lan in lai mot dong */
    char c,ch1;
    int scot, m, mm, k, dong, cot, i, j, n1, n2;
    dong=(yd-yt+1)/6;
    mm=(yd-yt+1) % 6;
    cot=xd-xt+1;
    for(i=0;i<=dong;++i)
    {
        if(i<dong)
            m=6;
        else
            m=mm;
        if(m>0)
        {
            scot 0;
            for(j=0; j < cot;++j)
                if(tao_mau(m,xt+j,yt+i*6))
                    scot=j+1;
            if(scot)
            {
                n1=scot % 256;
                n2= scot/256;
                for(k=0;k<dd;++k)
                {
                    fprintf(stderr,"%c%c%c%c%c",13,27,'+',0,n1,n2);
                    /*LQ*/
                    for(j=0; j < scot;++j)
                    {
                        /*bat phim*/
                        if (kbhit())
```

```

    {
    if ({ch1=getch()}==0)
        getch();
    if(ch1==27)
        goto ket;
    }
    c=tao_mau(m,xt+j,yt+i*6);
    fprintf(stdprn,"%c",c);
    }
    }
    fprintf(stdprn,"%c%c%c",27,'A',m);
    fprintf(stdprn,"\n");
}
}
ket: fprintf(stdprn,"%c%c",27,'@');
}

```

TRUY NHẬP TRỰC TIẾP VÀO BỘ NHỚ

Trong chương này trình bày các vấn đề:

- Hai kiểu địa chỉ: Địa chỉ phân đoạn và địa chỉ thực
- Truy nhập tới địa chỉ phân đoạn
- Đổi từ địa chỉ phân đoạn sang địa chỉ thực
- Bộ nhớ màn hình, truy nhập trực tiếp vào bộ nhớ màn hình
- Dùng con trỏ để lấy dữ liệu từ bộ nhớ phân đoạn
- Dùng con trỏ hàm để thực hiện các thủ tục của DOS

§1. CÁC HÀM TRUY NHẬP THEO ĐỊA CHỈ PHÂN ĐOẠN

1. Hàm pokeb: Gửi một ký tự vào bộ nhớ.

+ Nguyên mẫu trong dos.h như sau:

```
void pokeb(unsigned seg, unsigned off, char value);
```

+ Công dụng: Gửi giá trị ký tự value vào bộ nhớ tại địa chỉ phân đoạn seg:off

2. Hàm peekb: Nhận một ký tự từ bộ nhớ.

+ Nguyên mẫu trong dos.h như sau:

```
char peekb(unsigned seg, unsigned off);
```

+ Công dụng: Nhận một byte tại địa chỉ phân đoạn seg:off

3. Hàm poke: Gửi một số nguyên vào bộ nhớ.

+ Nguyên mẫu trong dos.h như sau:

```
void poke(unsigned seg, unsigned off, int value);
```

+ Công dụng: Gửi giá trị nguyên value vào bộ nhớ tại địa chỉ phân đoạn seg:off

4. Hàm peek: Nhận một số nguyên từ bộ nhớ.

+ Nguyên mẫu trong dos.h như sau:

```
int peek(unsigned seg, unsigned off);
```

+ Công dụng: Nhận một word tại địa chỉ phân đoạn seg:off

5. Hàm movedata: Sao các byte.

+ Nguyên mẫu trong mem.h như sau:

```
void movedata(unsigned seg_gui, unsigned off_gui,  
              unsigned seg_nhan, unsigned off_nhan, int n);
```

+ Công dụng: Sao n byte từ seg_gui:off_gui đến seg_nhan:off_nhan

§2. BỘ NHỚ MÀN HÌNH VĂN BẢN

2.1. Cách biểu diễn ký tự trong bộ nhớ màn hình

Bộ nhớ màn hình văn bản bắt đầu từ địa chỉ :

(0xb800:0x0000)

Khi đưa một ký tự vào vùng nhớ màn hình, thì nó sẽ hiện lên màn hình. Mỗi ký tự trên màn hình chiếm 2 byte trong bộ nhớ màn hình: byte đầu chứa mã ASCII, byte thứ hai biểu diễn màu hiển thị gọi là byte thuộc tính. Các bit của byte thuộc tính:

B7B6B5B4B3B2B1B0

được chia làm 3 nhóm:

+ Nhóm 1 gồm bit B7 biểu thị sự nhấp nháy. Nếu B7=0 thì ký tự không nhấp nháy, nếu B7=1 thì ký tự sẽ nhấp nháy.

+ Nhóm 2 gồm các bit B6, B5 và B4. Các bit này chứa được một số nguyên từ 0 đến 7 và biểu thị 8 màu nền của ký tự.

+ Nhóm 3 gồm các bit B3, B2, B1 và B0. Các bit này chứa được một số nguyên từ 0 đến 15 và biểu thị 16 màu của ký tự.

2.2. Trang màn hình

Mỗi trang màn hình gồm 80x25 ký tự, do đó cần $80 \times 25 \times 2 = 4000$ byte bộ nhớ. Thực tế mỗi trang màn hình được phân bố $4096 = 0x1000$ byte. Như vậy 4 trang màn hình được phân bố như sau:

+ Trang màn hình thứ 0 bắt đầu từ địa chỉ 0xB800:0x0000

+ Trang màn hình thứ 1 bắt đầu từ địa chỉ 0xB800:0x1000

+ Trang màn hình thứ 2 bắt đầu từ địa chỉ 0xB800:0x2000

+ Trang màn hình thứ 3 bắt đầu từ địa chỉ 0xB800:0x3000

2.3. Chọn trang hiển thị

Tại mỗi thời điểm chỉ có thể hiển thị được một trong 4 trang màn hình. Để hiển thị trang màn hình thứ t (t=0,1,2,3) chúng ta sử dụng chức năng 5 của ngắt 0x10 theo mẫu sau:

```

union REGS v,r;
v.h.ah = 5 ; // Chức năng 5
v.h.al = t ; // Số hiệu trang màn hình cần hiển thị
int86(0x10, &v, &r); // Thực hiện ngắt 0x10

```

2.4. Ví dụ minh họa

Ví dụ sau dùng hàm pokeb để đưa các ký tự vào các trang của bộ nhớ màn hình, sau đó dùng chức năng 5 của ngắt 0x10 để chọn trang hiển thị.

```

//CT9_03.CPP
#include <dos.h>
#include <conio.h>
char
d1[]={'C',1*16+14,'H',1*16+14,'U',1*16+14,'C',1*16+14};
char
d2[]={'M',2*16+15,'U',2*16+15,'N',2*16+15,'G',2*16+15};
void main()
{
    union REGS v,r;
    clrscr();
        //Mặc định hiển thị trang 0
    for (int i=0;i<8;++i)
        pokeb(0xb800,i,d1[i]);
    getch();
        //Hiện thị trang 1
    v.h.ah = 5 ;
    v.h.al = 1 ;
    int86(0x10,&v,&r);
    for (i=0;i<8;++i)
        pokeb(0xb800,0x1000+i,d2[i]);
    getch();
        //Hiện thị trang 0
    v.h.ah = 5 ;
    v.h.al = 0 ;
    int86(0x10,&v,&r);
    getch();
        //Hiện thị trang 1
    v.h.ah = 5 ;
    v.h.al = 1 ;
    int86(0x10,&v,&r);
    getch();
}

```

§3. CHUYỂN ĐỔI ĐỊA CHỈ

3.1. Để chuyển từ địa chỉ thực sang địa chỉ phân đoạn ta dùng các macro:

```
unsigned FP_SEG(địa_chi_thực)
```

```
unsigned FP_OFF(địa_chi_thực)
```

3.2. Để chuyển từ địa chỉ phân đoạn sang địa chỉ thực ta dùng macro:

```
void far *MK_FP(seg, off)
```

Ví dụ 1. Sau khi thực hiện các câu lệnh:

```
char buf[100];
```

```
unsigned ds, dx;
```

```
ds = FP_SEG(buf); dx = FP_OFF(buf);
```

thì ds:dx chứa địa chỉ của mảng buf.

Ví dụ 2. Sau khi thực hiện các câu lệnh:

```
char far *pchar;
```

```
pchar = (char far*)MK_FP(0xb800:0);
```

thì pchar trỏ tới địa chỉ đầu của bộ nhớ màn hình. Khi đó ta có thể sử dụng các lệnh gán để truy nhập trực tiếp tới bộ nhớ màn hình.

§4. CÁC VÍ DỤ MINH HỌA

Chương trình 1. Chương trình minh họa cách truy nhập trực tiếp vào bộ nhớ màn hình có địa chỉ đầu là 0xB800:0. Chương trình gồm hàm main() và hai hàm sau:

1. Hàm cuaso

```
void cuaso(int dongt, int cott, int dongd,  
           int cotd, int maucs);
```

thiết lập một cửa sổ màu có tọa độ góc trên-trái là (dongt, cott) và góc dưới-phải là (dongd, cotd). Màu cho bởi tham số maucs. Ở đây sử dụng hàm pokeb và địa chỉ phân đoạn.

2. Hàm duarmh

```
void duarmh(char *day, int dong, int cotd,  
            int cotc, int m_nen, int m_chu);
```

sẽ đưa ra màn hình một dãy ký tự (chứa trong day) tại dòng dong, từ cột cotd đến cotc. Màu nền cho bởi m_nen, màu chữ cho bởi m_chữ. ở đây sử dụng toán tử gán trên địa chỉ thực.

Trong hàm main() sẽ sử dụng các hàm cuaso và duarmh để tạo hai cửa sổ và viết hai dòng chữ trên trang màn hình thứ hai (từ dòng 26 đến dòng 50).

```

/*
Chương trình minh họa cách truy nhập trực tiếp
vào bộ nhớ của màn hình */
#include "dos.h"
#include "conio.h"
void duarmh(char *day, int dong, int cotd,
            int cotc, int m_nen, int m_chu);
void cuaso(int dongt, int cott, int dongd,
            int cold, int maucs);
main()
{
    cuaso(26, 1, 50, 80, BLUE);
    duarmh("Chuc mung nam moi", 28, 30, 50, MAGENTA, WHITE);
    cuaso(30, 20, 46, 60, RED);
    duarmh("Chuc mung nam moi", 40, 30, 50,
          MAGENTA, YELLOW);
    getch();
}
void cuaso(int dongt, int cott, int dongd, int cotd, int maucs)
/* Dung dia phan doan */
{
    int i, j, p, t, dt, dd, mau;
    union REGS v, r;
    /* Xac dinh thuoc tinh mau */
    mau = (maucs << 4) | maucs;
    /* Xac dinh trang man hinh t va cac chi so
       dong tren dt, dong duoi dd trong trang t */
    t = (dongt - 1) / 25;
    dt = (dongt - 1) - t * 25;
    dd = (dongd - 1) - t * 25;
    /* Chon t la trang hien thi */
    v.h.ah = 5;
    v.h.al = t;
    int86(0x10, &v, &r);
    /* Dua cac khoang trong (ma 32) va thuoc tinh mau
       vao cac vi tri thich hop cua bo nho man hinh */
    for (i = dt; i <= dd; ++i)
    {
        p = t * 4096 + i * 160 + (cott - 1) * 2;
        for (j = 0; j <= cotd - cott; ++j)
        {
            pokeb(0xb800, p + 2 * j, 32);
            pokeb(0xb800, p + 2 * j + 1, mau);
        }
    }
}
void duarmh(char *day, int dong, int cotd,
            int cotc, int m_nen, int m_chu)
/* Dung dia chi thuc */
{
    int i, p, t, d, kt, mau;
    char far *buf;

```

```

union REGS v,r;
    /* Lay dia chi thuc cua bo nho man hinh */
buf=(char far*)MK_FP(0xb800,0);
    /* Xac dinh thuoc tinh mau */
mau = (m_nen << 4)+m_chu;
    /* Xac dinh trang man hinh t
    va cac chi so dong d trong trang t */
t=(dong-1)/25;
d=dong-1-t*25;
    /* Chon t la trang hien thi */
v.h.ah=5;v.h.al=t; int86(0x10,&v,&r);
p=t*4096+d*160+(cotd-1)*2;
    /* Dua cac ky tu va thuoc tinh mau
    vao cac vi tri thich hop cua bo nho man hinh */
for (i=0;i<=cotc-cotd;++i)
{
    if ((kt=day[i])==0) break;
    buf[p+2*i]=kt;
    buf[p+2*i+1]=mau;
}
}

```

Chương trình 2. Biết địa chỉ của các thủ tục xử lý ngắt được lưu trữ trong bộ nhớ từ địa chỉ 0000:0000 đến 0000:0x0400. Chương trình sẽ cho biết địa chỉ của thủ tục xử lý ngắt n (giá trị n nhập vào từ bàn phím). Số hiệu của ngắt được tính từ 0, nhưng n được đánh số từ 1.

```

/* Xac dinh dia chi cac thu tuc ngat */
#include "dos.h"
#include "conio.h" #include "stdio.h"
main()
{
    unsigned char far *p; /*p se tro toi bang vecto ngat*/
    int n; /* n - so hieu ngat, n=1,2,... */
    int k; /* vi tri cua ngat n trong bang vecto ngat */
    unsigned seg,off;
    /* p tro toi bang vecto ngat */
    p=(unsigned char far*)MK_FP(0,0);
    clrscr();
    while(1)
    {
        printf("\n So hieu ngat (Bam 0 - Ket thuc): ");
        scanf("%d",&n);
        if(n==0) break;
        k=(n-1)*4;
        off=p[k]+256*p[k+1];
        seg=p[k+2]+256*p[k+3];
        printf("\nDia chi %x:%x",seg,off);
    }
}

```

Chương trình 3. Chương trình minh họa cách dùng con trỏ hàm để thực hiện thủ tục khởi động lại máy của DOS, biết địa chỉ đầu của thủ tục này là 0xFFFF:0000. Chương trình yêu cầu nhập mật khẩu. Nếu chọn đúng (bấm ABCD và Enter) thì chương trình tiếp tục làm việc, nếu vào sai thì sẽ khởi động lại máy.

```
#include <dos.h>
#include <conio.h>
#include <iostream.h>
#include <ctype.h>
typedef void far (*HAM)(void);
void khoi_dong_may(void)
{
    HAM t;
    f = (HAM)MK_FP(0xFFFF, 0);
    f();
}
char mat_khau[] = {'A', 'B', 'C', 'D'};
int n = sizeof(mat_khau)/sizeof(char);
void main()
{
    char i, ch, sai_mat_khau;
    clrscr();
    i = 0;
    sai_mat_khau = 0;
    cout << "\nMat khau: ";
    while(1)
    {
        ch = getch();
        if (ch==13) break;
        cout << '*' ;
        if (i<n)
        {
            if (toupper(ch)!=mat_khau[i])
                sai_mat_khau=1;
        }
        else
            sai_mat_khau=1;
        ++i ;
    }
}
```



```

int giay    (int)(du/18);
            // Doi ra ky tu dua vao mang buf_time
buf_time[12]=gio/10 + 48;
buf_time[14]=gio%10 + 48;
buf_time[18]=phut/10 + 48;
buf_time[20]=phut%10 + 48;
buf_time[24]=giay/10 + 48;
buf_time[26]=giay%10 + 48;
            // Dua thong bao goi ra man hinh
for (i=0; i<28; ++i)
p_mh[i] = buf_time[i];
getch();
            //Khoi phuc man hinh
for (i=0; i<28; ++i)
p_mh[i] = buf_luu[i];
}
void main()
{
    int ch1, ch2;
    clrscr();
    while(1)
    {
        if (kbhit())
        {
            ch1=getch();
            if (ch1==0)
                ch2=getch();
            if (ch1==27)    //ESC
                break;
            if (ch1==0 && ch2==59)    // Bam F1
                thong_bao_thoi_gian();
        }
        //In cac chu cai mot cach ngau nhien
        gotoxy(random(80)+1,random(25)+1);
        putchar(random(26)+65);
        delay(400);
    }
}

```

KHUÔN HÌNH (TEMPLATE)

§1. KHUÔN HÌNH HÀM

1.1. Khái niệm

Xét hàm $\max(x,y)$ để tính $\max$ của 2 đối. Rõ ràng x và y có thể có bất kỳ kiểu nào (kiểu chuẩn của C^{++} , kiểu cấu trúc, kiểu hợp hoặc kiểu đối tượng) miễn sao chúng có một thứ tự nào đấy.

Nếu không dùng khuôn hình, thì cần xây dựng nhiều hàm (được định nghĩa chồng) ứng với các kiểu khác nhau của x và y .

Dùng khuôn hình, ta có thể định nghĩa một mẫu cho một họ các hàm tương ứng, bằng cách xem kiểu dữ liệu (data type) như một tham số (đối).

1.2. Khai báo

Các đối ứng với các kiểu dữ liệu được khai báo như sau:

```
template <class T1, class T2, ..., class Tm>
```

Chú ý:

a. Khuôn hình hàm trong đó có sử dụng các đối $T1, T2, \dots, Tm$ phải được viết ngay sau dòng khai báo nói trên.

b. Các đối $T1, T2, \dots, Tm$ là đối của chỉ khuôn hình hàm này.

1.3. Cách xây dựng khuôn hình hàm

Khuôn hình hàm được xây dựng theo mẫu sau:

```
template <class T>
T max(T x, T y)
{
    return (x>y) ? x : y ;
}
```

1.4. Cách sử dụng

+ Sử dụng thông qua lời gọi hàm.

+ Khi sử dụng khuôn hình này trong chương trình (thông qua lời gọi), thì Compiler sẽ sản sinh một hàm tương ứng với kiểu dữ liệu trong lời gọi, ví dụ:

```
int a, b, c;
Myclass u, v, w;
c = max(a,b);
w := max(u,v);
```

+ Có thể dùng khuôn hình hàm max đối với các kiểu dữ liệu chuẩn của C++ cũng như kiểu đối tượng bất kỳ, miễn sao có thể áp dụng phép so sánh >

1.5. Định nghĩa chồng khuôn hình hàm

Có thể dùng cùng một tên để định nghĩa các khuôn hình hàm với một trong các khác biệt sau:

+ Khác nhau về đối: kiểu đối, số đối (xem trường hợp 1 và 2 trong ví dụ sau)

+ Khác nhau về số kiểu dữ liệu (xem ví dụ, hai trường hợp đầu chỉ có một kiểu T, trường hợp 3 có 2 kiểu T1 và T2)

Ví dụ:

```
template <class T>
T operator*(T, T) // Nhân 2 đối tượng cùng kiểu
template <class T>
T operator*(T*, int) // Nhân dãy đối tượng cùng kiểu
template <class T1, class T2>
T2 operator*(T1, T2) // Nhân 2 đối tượng khác kiểu,
ví dụ ma trận và véc tơ
```

§2. CÁC VÍ DỤ VỀ KHUÔN HÌNH HÀM

Ví dụ 1

```
/* Function Templates */
#include <iostream.h>
#include <conio.h>
#include <math.h>
class DS
{
private:
    int n;
    int *a;
public:
    DS(int n1)
    {
        n=n1;
        a = new int[n+1];
    }
    void nhap();
    void xuat();
```

```

    friend int operator>(const DS &x,const DS &y);
    friend DS operator-(const DS &x, const DS &y);
    friend double abss(const DS &x);
};

double abss(const DS &x)
{
    double s=0.0; int i;
    for(i=1;i<=x.n;++i)
        s += x.a[i]*x.a[i];
    return sqrt(s);
}

DS operator-(const DS &x,const DS &y)
{
    int i; DS z(x.n);
    for(i=1;i<=x.n;++i)
        z.a[i] = x.a[i] - y.a[i];
    return z;
}

int operator>(const DS &x,const DS &y)
{
    int i;
    for(i=1;i<=x.n;++i)
        if(x.a[i] > y.a[i])
            return 1;
    return 0;
}

void DS::xuati()
{
    cout << "\n";
    for(int i=1;i<=n;++i) cout << a[i] << " ";
}

void DS::nhap()
{
    int i;
    for(i=1;i<=n;++i)
    {
        cout << "\n P" << i << " thu " << i << " = ";
        cin >> a[i];
    }
}

double abss(int x)
{
    return ((double)abs(x));
}

double abss(double x)
{
    return fabs(x);
}

template <class T>
T max(T x, T y)
{

```

```

    return (x>y)?x:y;
}
template <class M>
double KC(M x, M y)
{
    return abss(x-y);
}
void main()
{
    double fmax;
    int imax;
    fmax = max(4.5,10.9);
    imax = max(20,4);
    cout << fmax <<" "<<imax;
    DS x(3),y(3),z(3);
    x.nhap();
    y.nhap();
    z=max(x,y);
    x.xuat();
    y.xuat();
    z.xuat();
    double d1,d2,d3;
    d1=KC(1,11);
    d2=KC(1.1,11.1);
    d3=KC(x,y);
    cout<<'\\n' <<d1<<" "<<d2<<" "<<d3;
    getch();
}

```

Ví dụ 2

```

/*
Function Templates
Gom 3 ham co doi Template:
template <class T>
void sap_xep(T *x, int n)
template <class T>
void nhap_day_dt(T *dt, int &n)
template <class T>
void xuat_day_dt(T *dt, int &n)
Doi T bieu thi 3 kieu:
    + int
    + double
    + TS (lop)
*/
#include <iostream.h>
#include <conio.h>
#include <math.h>
class TS
{

```

```

private:
    char ht[25];
    double td;
public:
    friend istream& operator>>(istream &in, TS &t)
    {
        cout << "\nHọ ten: ";
        in.get(t.ht,25);
        cout << "Tong diem: ";
        in >> t.td;
        in.ignore();
        return in;
    }
    friend ostream& operator<<(ostream &out,const TS &t)
    {
        out << "\nHọ ten: " << t.ht << " Tong diem: " << t.td;
        return out;
    }
    friend int operator>(const TS &ts1,const TS &ts2)
    {
        return (ts1.td>ts2.td) ;
    }
};

template <class T>
void sap_xep(T *x, int n)
{
    int i,j;
    T tg;
    for(i=1; i<n; ++i)
        for(j=i+1; j<=n; ++j)
            if( !(x[i]>x[j]) )
            {
                tg=x[i]; x[i]=x[j]; x[j]=tg;
            }
}

template <class T>
void nhap_day_dt(T *dt, int &n)
{
    cout << "\nSo doi tuong: ";
    cin >> n; cin.ignore();
    for(int i=1; i<=n; ++i)
    {
        cout << "\nNhap doi tuong thu " << i << " : " ;
        cin >> dt[i];
    }
}

template <class T>
void xuat_day_dt(T *dt, int n)
{
    for(int i=1; i<=n; ++i)
        cout << "\nDoi tuong thu " << i << " : " << dt[i];
}

```

```

    }
void main()
{
    int a[10];
    double b[10];
    TS t[10];
    int n;
    nhap_day_dt(a,n);
    sap_xep(a,n);
    xuat_day_dt(a,n);
    nhap_day_dt(b,n);
    sap_xep(b,n);
    xuat_day_dt(b,n);
    nhap_day_dt(t,n);
    sap_xep(t,n);
    xuat_day_dt(t,n);
    getch();
}

```

Ví dụ 3

```

/*
templ_5.CPP
Function Templates
Hai doi template T1, T2 dung cho ham
Chu y cach khai bao 2 doi template:
template <class T1, class T2>
T2 nhan(T1 a, T2 x)
{
    ...
}
*/
#include <iostream.h>
#include <conio.h>
#include <math.h>
#include <iomanip.h>
class VT;
class MT;

class MT
{
private:
    int a[20][20];
    int n;
public:
    friend istream& operator>>(istream &in, MT &x);
    friend ostream& operator<<(ostream &out, const MT &x);
    VT operator*(const VT &x);
};

istream& operator>>(istream &in, MT &x)
{
    int i,j;

```

```

    cout << "\n Cap ma tran N= ";
    in >> x.n;
    for(i=1; i<=x.n; ++i)
        for(j=1; j<=x.n; ++j)
        {
            cout << "a{" <<i<<"}["<<j<<"]=" ";
            in >> x.a[i][j];
        }
    return in;
}
ostream& operator<<(ostream &out,const MT &x)
{
    int i,j;
    for(i=1; i<=x.n; ++i)
    {
        out << "\n";
        for(j=1; j<=x.n; ++j)
            out << setw(5) << x.a[i][j];
    }
    return out;
}
class VT
{
private:
    int a[20];
    int n;
public:
    friend istream& operator>>(istream &in, VT &x);
    friend ostream& operator<<(ostream &out,const VT &x);
    friend class MT;
} ;
VT MT::operator*(const VT &x)
{
    VT y; int i,j;
    y.n=n;
    for(i=1; i<=n; ++i)
    {
        y.a[i]=0;
        for(j=1; j<=n; ++j)
            y.a[i] += a[i][j]*x.a[j];
    }
    return y;
}
istream& operator>>(istream &in, VT &x)
{
    int i;
    cout << "\n Cap vec to N= ";
    in >> x.n;
    for(i=1; i<=x.n; ++i)
    {

```

```

        cout << "a[" <<i<<"]  " ;
        in >> x.a[i];
    }
    return in;
}
ostream& operator<<(ostream &out,const VT &x)
{
    int i;
    out << "\n";
    for(i=1; i<=x.n; ++i)
        out << setw(5) << x.a[i];
    return out;
}
template <class T1, class T2>
T2 nhan(T1 a, T2 x)
{
    return a*x;
}
void main()
{
    int k; double x,y;
    clrscr();
    k = nhan(4,5);
    x = nhan(3,5.5);
    y = nhan(5.0,6.5);
    cout << "\nk= " << k << " x= " << x << " y= " << y;
    MT a; VT u,v;
    cout << "\nNhap MT a:" ; cin>>a;
    cout << "\nNhap VT u:" ; cin>>u;
    v = a*u;
    cout << "\nMa tran A: " << a ;
    cout << "\n Vec to u: " << u;
    cout << "\n Vec to v=Au: " << v;
    getch();
}

```

§3. KHUÔN HÌNH LỚP

3.1. Khái niệm

Có thể dùng template để xây dựng một khuôn hình lớp trong đó kiểu thuộc tính được xem như các đối. Bằng cách đó ta có thể tạo ra một họ các lớp giống nhau về bản chất xử lý, nhưng khác nhau về kiểu dữ liệu.

3.2. Khai báo

Các đối ứng với các kiểu dữ liệu được khai báo như sau:

```
template <class T1, class T2, ....., class Tm>
```

Chú ý:

- Khuôn hình lớp trong đó có sử dụng các đối T1, T2,...,Tm phải được viết ngay sau dòng khai báo nói trên.
- Các đối T1, T2,...,Tm là đối của chỉ khuôn hình lớp này.

3.3. Cách xây dựng khuôn hình lớp

Khuôn hình lớp được xây dựng theo mẫu sau:

```
template <class T>
class List
{
    private:
        T *v ;
        int n;
    public:
        List(int);
        T& operator[] (int i)
        {
            return v[i] ;
        }
};
// Bên ngoài định nghĩa lớp
template <class T> List<T>::List(int n)
{
    v = new T[n];
    size = n;
}
```

3.4. Cách sử dụng

+ Dùng khuôn hình lớp để tạo một các đối tượng với các kiểu dữ liệu bất kỳ, bằng câu lệnh khai báo, new, hàm tạo.

+ Sử dụng các phương thức để xử lý các đối tượng tạo ra.

Ví dụ có thể tạo ra các đối tượng List với kiểu dữ liệu bất kỳ như sau:

```
List <int> x(20) ;
List <Myclass> u(30);
```

Sau đó dùng phương thức chỉ số và phép gán:

```
x[3] = 7 ;
u[10] = m ;    // m là một đối tượng kiểu List
```

§4. VÍ DỤ VỀ KHUÔN HÌNH LỚP

```
/* templ_4.CPP
class Templates
    Một doi template T dung cho lop List
```

```

    Doi T biey thi 3 kieu:
        + int
        + double
        + TS (lop)
    Chu y: Dong dau tien cua phuong thuc viet ben ngoai lop*/
    // template <class T> void List <T>::sap_xep()
#include <iostream.h>
#include <conio.h>
#include <math.h>
class TS
{
    private:
        char ht[25];
        double td;
    public:
        friend istream& operator>>(istream &in, TS &t)
        {
            cout << "\nHo ten: ";
            in.get(t.ht,25);
            cout << "Tong diem: ";
            in >> t.td;
            in.ignore();
            return in;
        }
        friend ostream& operator<<(ostream &out,const TS &t)
        {
            out << "\nHo ten: " << t.ht << " Tong diem: " << t.td;
            return out;
        }
        friend int operator>(const TS &ts1,const TS &ts2)
        {
            return (ts1.td>ts2.td) ;
        }
} ;
template <class T>
class List
{
    private:
        T *t;
        int n;
    public:
        List()
        {
            n=0; t=NULL;
        }
        List(int );
        void sap_xep();
        void nhap_day_dt();
        void xuat_day_dt();
};

```

```

template <class T> List<T>::List(int n1)
{
    n=n1;
    t = new T[n+1];
}
template <class T> void List <T>::sap_xep()
{
    int i,j;
    T tg;
    for(i=1; i<n; ++i)
        for(j=i+1; j<=n; ++j)
            if( !(t[i]>t[j]) )
            {
                tg=t[i]; t[i]=t[j]; t[j]=tg;
            }
}
template <class T> void List<T>::nhap_day_dt()
{
    if (t!=NULL) delete t;
    cout << "\nSo doi tuong: ";
    cin >> n; cin.ignore();
    t = new T[n+1];
    for(int i=1; i<=n; ++i)
    {
        cout << "\nNhap doi tuong thu " << i << " : ";
        cin >> t[i];
    }
}
template <class T> void List<T>::xuat_day_dt()
{
    for(int i=1; i<=n; ++i)
        cout << "\nDoi tuong thu " << i << " : " << t[i];
}
void main()
{
    List <int> a;
    List <double> b;
    List <TS> ts;
    a.nhap_day_dt();
    a.sap_xep();
    a.xuat_day_dt();
    b.nhap_day_dt();
    b.sap_xep();
    b.xuat_day_dt();
    ts.nhap_day_dt();
    ts.sap_xep();
    ts.xuat_day_dt();
    getch();
}

```

MỘT SỐ CHƯƠNG TRÌNH HƯỚNG ĐỐI TƯỢNG TRÊN C++

Chương này trình bày thêm một số chương trình hướng đối tượng trên C++. Đây là các chương trình tương đối phức tạp, hữu ích và sử dụng các công cụ mạnh của C++ như: Cách truy nhập trực tiếp bộ nhớ màn hình, kỹ thuật đồ họa, con trỏ void, tính kế thừa, lớp cơ sở trừu tượng, tương ứng bội, phương thức ảo.

§1. LỚP CỬA SỔ

Chương trình gồm lớp cửa sổ và lớp stack

+ Lớp cửa sổ

Thuộc tính gồm:

```
char *noidung; // Trỏ đến vùng nhớ chứa nội dung
                // soạn thảo trên cửa sổ
int cao,rong ; // Chiều cao và chiều rộng của sổ
int mau;       // mau = 16*mau_nen + mau_chu
int ra_mh;     // Cho biết cửa sổ đã được đưa ra
                // màn hình chưa?
int posx,posy; // Vị trí trên trái của cửa sổ
                // trên màn hình
word *pluu;    // Trỏ đến vùng nhớ chứa nội dung
                // phần màn hình bị cửa sổ đè lên
```

Phương thức gồm:

```
cua_so();
cua_so(int c,int r,byte mau_nen, byte mau_chu);
int push(int x,int y); // Đưa cửa sổ ra
                        // màn hình tại (x,y)
                        // cho phép soạn thảo trên cửa sổ
                        // Bấm F6 chuyển sang cửa sổ khác
                        // Bấm ESC kết thúc
void pop();           // Tháo gỡ cửa sổ và
                        // khôi phục màn hình
int get_ra_mh();
```

+ Lớp stack (dùng để quản lý một dãy cửa sổ)

Thuộc tính gồm:

```
int max; //Số của số cực đại có thể quản lý
int num; //Số của số hiện có trong stack
cua_so **pcs; //Con trỏ trỏ đến vùng nhớ chứa
               //địa chỉ của các đối tượng cua_so
```

Phương thức gồm:

```
stack();
stack(int max_cs);
int accept(cua_so *cs,int x,int y); //Đưa một cửa sổ
                                   //vào stack, nó sẽ hiện lên màn hình
void del(); // Loại cửa sổ khỏi stack,
            // nó sẽ bị xóa khỏi màn hình
```

Nội dung chương trình:

+ Đầu tiên hiện cửa sổ thứ nhất nền GREEN chữ WHITE. Có thể soạn thảo trên đó.

+ Nếu bấm ESC kết thúc chương trình, nếu bấm F6 thì hiện thêm cửa sổ thứ hai nền CYAN chữ MAGENTA. Có thể soạn thảo trên đó.

+ Nếu bấm ESC kết thúc chương trình, nếu bấm F6 thì hiện thêm cửa sổ thứ ba nền RED chữ YELLOW. Có thể soạn thảo trên đó.

+ Đang ở một cửa sổ, nếu bấm ESC thì kết thúc chương trình, nếu bấm F6 thì hiện cửa sổ tiếp theo (theo thứ tự vòng quanh: 1 -> 2 -> 3 -> 1).

Chương trình sử dụng phương pháp truy nhập trực tiếp bộ nhớ màn hình trình bày trong chương 9.

```
// CT10_01.CPP
// lop_cua_so
#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include <alloc.h>
typedef unsigned int word;
typedef unsigned char byte;
struct kt_word
{
    word kt;
};
struct kt_byte
{
    byte ma, mau;
};
union ky_tu
```

```

{
    struct kt_byte h;
    struct kt_word x;
};
typedef union ky tu far *VP;
VP vp_ptr = (VP)MK_FP(0xb800,0);
// Vi tri x,y tren man hinh
#define VPOS(x,y) (VP)(vp_ptr + ((y)-1)*80+(x)-1)
class cua_so
{
private:
    char *noidung;
    int cao, rong;
    int mau;          // mau = 16*mau_nen + mau_chu
    int ra_mh, int posx,posy;
    word *pluu;
public:
    cua_so();
    cua_so(int c,int r,byte mau_nen, byte mau_chu);
    int push(int x,int y);
    void pop();
    int get_ra_mh();
};
cua_so::cua_so()
{
    cao=rong=mau=ra_mh-posx=posy=0;
    noidung=NULL; pluu=NULL;
}
cua_so::cua_so(int c,int r,byte mau_nen, byte mau_chu)
{
    cao=c; rong=r;
    mau = 16*mau_nen+mau_chu;
    ra_mh=posx=posy=0;
    noidung = (char*)malloc(cao*rong);
    for (int i=0;i<cao*rong;++i)
        noidung[i]=32;
    pluu= (word*)malloc(2*cao*rong);
}
int cua_so::push(int x,int y)
{
    word *p= pluu; char *pnd=noidung;
    VP ptr;
    int i,j;
    // Luu man hinh
    if (ra_mh==0)
    {
        ra_mh=1; posx=x;posy=y;
        for (i=posx;i<=posx+rong-1;++i)
            for (j=posy;j<=posy+cao-1;++j)
            {
                ptr=VPOS(i,j); *p=ptr->x.kt; ++p;
            }
    }
}

```

```

    }
}
// Hien noi dung dang soan thao tren cua so
for (i=posx;i<=posx+rong-1;++i)
    for (j=posy;j<=posy+cao-1;++j)
    {
        ptr=VPOS(i,j);
        ptr->h.mau=mau;
        ptr->h.ma=*pnd; ++pnd;
    }
// Soan thao
int xx=posx,yy=posy,ch1,ch2;
while (1)
{
    gotoxy(xx,yy);
    if ((ch1=getch())==0) ch2=getch();
    if (ch1==27)break; // ESC Ket Thuc Soan Thao
    else if (ch1==0&&ch2==64)break; //F6
    else if (ch1==13)
    {
        ++yy; xx=posx; if(yy>=posy+cao) break;
    }
    else if (ch1!=0)
    {
        ptr=VPOS(xx,yy);
        ptr->h.ma=ch1;
        ++xx;
        if (xx>=posx+rong) {++yy; xx=posx;}
        if (yy>=posy+cao) break;
    }
    else if (ch2==72||ch2==80||ch2==75||ch2==77)
    {
        if (ch2==72) yy--;
        else if (ch2==80) ++yy;
        else if (ch2==75) --xx;
        else ++xx;
        if (xx<posx) xx=posx;
        if (xx>=posx+rong) {++yy; xx=posx;}
        if (yy<posy) yy=posy;
        if (yy>=posy+cao) break;
    }
}
// Luu ket qua soan thao
pnd=noidung;
for (i=posx;i<=posx+rong-1;++i)
    for (j=posy;j<=posy+cao-1;++j)
    {
        ptr=VPOS(i,j);
        *pnd=ptr->h.ma; ++pnd;
    }
if (ch1==0&&ch2==64) return 0; //F6

```

```

        else return 1;
    }
}

void cua_so::pop() // Khoi phuc vung nho bi cua so chiem
{
    if (ra_mh==0) return;
    ra_mh=0;
    word *p=pluu;
    VP ptr;
    int i, j;
    for (i=posx; i<=posx+rong-1; ++i)
        for (j=posy; j<=posy+cao-1; ++j)
        {
            ptr=VPOS(i, j);
            ptr->x.kt=*p;
            ++p;
        }
}

int cua_so::get_ra_mh()
{
    return ra_mh;
}

//class stack
class stack
{
private:
    int max, num;
    cua_so **pcs;
public:
    stack();
    stack(int max_cs);
    int accept(cua_so *cs, int x, int y);
    void del();
};

stack::stack()
{
    max=num=0;
    pcs=NULL;
}

stack::stack(int max_cs)
{
    max=max_cs;
    num=0;
    pcs=(cua_so**)malloc(max*sizeof(cua_so*));
    for (int i=0; i<max; ++i) pcs[i]=NULL;
}

int stack::accept(cua_so *cs, int x, int y)
{
    int gt;
    if (num==max) return 0;
    if (!cs->get_ra_mh())
    {

```

```

        pcs[num] cs;
        ++num;
    }
    gt=cs->push(x,y);
    return gt;
}
void stack::del()
{
    if (num==0) return;
    --num;
    pcs[num]->pop();
    pcs[num]=NULL;
}
main()
{
    int ch;
    cua_so w1(10,40, GREEN,WHITE),
           w2(12,42, CYAN,MAGENTA),
           w3(14,44, RED,YELLOW);
    stack s(4);
    clrscr();
    while(1)
    {
        ch=s.accept(&w1,5,5);
        if(ch==-1)break;
        ch=s.accept(&w2,8,8);
        if(ch==1)break;
        ch=s.accept(&w3,11,11);
        if(ch==1)break;
    }
    s.del(); s.del(); s.del();
}

```

§2. LỚP MENU

Lớp cmenu có 2 phương thức để tạo lập và sử dụng menu:

1. Hàm tạo

```
cmenu(int so_cn_menu, char **nd_menu);
```

dùng để tạo một menu (đối tượng kiểu cmenu). Hàm tạo chứa 2 đối là:

- + Biến so_cn_menu chứa số chức năng của menu
- + Con trỏ nd_menu trỏ tới một vùng nhớ chứa địa chỉ các chuỗi ký tự dùng làm tiêu đề menu và tiêu đề các chức năng menu.

Ví dụ các câu lệnh:

```

char *nd[]={ "Quản lý vật tư", "Nhập số liệu",
             "Tìm kiếm", "Kết thúc" };
cmenu mc(3,nd);

```

sẽ tạo một menu mc gồm 3 chức năng: Nhập số liệu, Tìm kiếm và Kết thúc. Menu có tiêu đề là: Quản lý vật tư

2. Phương thức

```
int menu(int x,int y,int mau_nen,int mau_chon);
```

thực hiện các việc sau:

- + Hiển thị menu tại vị trí (x,y) trên màn hình. Menu có màu nền xác định bởi đối mau_nen và màu chức năng định chọn (hộp sáng) xác định bởi đối mau_chon.

- + Cho phép sử dụng các phím mũi tên lên, xuống để di chuyển hộp sáng và dùng phím Enter để thoát khỏi phương thức.

- + Sau khi thoát khỏi, phương thức trả về giá trị bằng số thứ tự (tính từ 1) của chức năng được chọn.

Chương trình dưới đây xây dựng lớp cmenu và minh họa cách sử dụng lớp này.

```
/* CT10_02.CPP
menu.cpp
lop cmenu */
#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include <alloc.h>
#include <string.h>
typedef unsigned int word;
typedef unsigned char byte;
struct kt_word
{
    word kt;
};
struct kt_byte
{
    byte ma, mau;
};
union ky_tu
{
    struct kt_byte h;
    struct kt_word x;
};
typedef union ky_tu far *VP;
VP vptr (VP)MK_FP(0xb800,0);
// Vị trí x,y trên màn hình
#define VPOS(x,y) (VP)(vptr + ((y)-1)*80+(x)-1)
class cmenu
{
private:
```

```

    int so_cn,cao,rong,posx,posy;
    int chon;
    char **nd;
private:
    void hiendc(char *dc,int x,int y, int mau);
    void hien_menu(int x,int y,int mau_nen,
                    int mau_chon);
public:
    cmenu(int so_cn_menu,char **nd_menu);
    int menu(int x,int y,int mau_nen,int mau_chon);
};

cmenu::cmenu(int so_cn_menu,char **nd_menu)
{
    cao=so_cn=so_cn_menu; nd=nd_menu;
    rong=0;
    chon=1;
    int d;
    for(int i=0;i<=so_cn;++i)
        if( (d=strlen(nd[i])) > rong) rong=d;
}

void cmenu::hiendc(char *dc,int x,int y, int mau)
{
    VP ptr; int i;
    byte m=16*mau+15; //chu trang
    for(i=0;i<rong;++i)
    {
        ptr=VPOS(x+i,y);
        ptr->h.mau=m;
        ptr->h.ma=32;
    }
    for(i=0;i<rong;++i)
    {
        ptr=VPOS(x+i,y);
        if(dc[i]!='\0')break;
        ptr->h.ma=dc[i];
    }
}

void cmenu::hien_menu(int x,int y,
                      int mau_nen,int mau_chon)
{
    for(int i=0;i<=so_cn;++i)
        hiendc(nd[i],x,y+i,mau_nen);
    hiendc(nd[chon],x,y+chon,mau_chon);
}

int cmenu::menu(int x,int y,int mau_nen,int mau_chon)
{
    int ch1,ch2,chonluu;
    //Trinh bay
    hien_menu(x,y,mau_nen,mau_chon);
    //Bat phim
    while(1)

```

```

{
    if( (ch1=getch())==0 ) ch2=getch();
    if(ch1==13) //chon chuc nang
        return (chon);
    else if( (ch1==0)&&(ch2==80||ch2==72))
    {
        //Di chuyen hop sang
        chonluu=chon;
        if(ch2==80) ++chon;
        else --chon;
        if(chon<1) chon=cao;
        else if(chon>cao) chon=1;
        if(chon!=chonluu)
        {
            hiendc(nd[chonluu],x,y+chonluu,mau_nen);
            hiendc(nd[chon],x,y+chon,mau_chon);
        }
    }
}

char *nd[]={ "TINH DIEN TICH", "Tam giac","Hinh tron",
             "Chu nhât", "Hinh vuong",
             "Ket thuc chuong trinh"};

void main()
{
    cmenu mc(5,nd); int chon;
    clrscr();
    while(1)
    {
        chon=mc.menu(5,5,BLUE,MAGENTA);
        if(chon==1)
        {
            clrscr();
            puts("TAM GIAC");
            getch();
            clrscr();
        }
        else if(chon==2)
        {
            clrscr();
            puts("HINH TRON");
            getch();
            clrscr();
        }
        else if(chon==3)
        {
            clrscr();
            puts("CHU NHAT");
            getch();
            clrscr();
        }
    }
}

```

```

        else if (chon==4)
        {
            clrscr();
            puts("HINH VUONG");
            getch();
            clrscr();
        }
        else break;
    }
}

```

§3. LỚP HÌNH HỌC

Chương trình dưới đây gồm:

- + Lớp “hinh” là lớp cơ sở trừu tượng
- + Và 3 lớp dẫn suất từ lớp “hinh” là:
 - Lớp “khoihop” biểu thị các khối hộp lập phương
 - Lớp “duong” biểu thị các đoạn thẳng qua 2 điểm
 - Lớp “tron” biểu thị các đường tròn

Chương trình minh họa cách dùng tượng ứng bội và phương thức ảo. Nội dung chương trình như sau:

- + Khi chạy chương trình sẽ thấy xuất hiện một khối hộp lập phương.
- + Có thể di chuyển khối hộp bằng các phím mũi tên.
- + Bấm phím Q sẽ xuất hiện một đoạn thẳng.
- + Có thể di chuyển đoạn thẳng bằng các phím mũi tên.
- + Bấm phím Q sẽ xuất hiện một đường tròn.
- + Có thể di chuyển đường tròn bằng các phím mũi tên.
- + Bấm phím Q sẽ kết thúc chương trình.

```

/* CT10_03.CPP
   LOP hinh hoc
   Minh hoa cach dung:
   + lop co so tru tru tuong
   + Tuong ung boi va phuong thuc ao
*/
#include <graphics.h>
#include <process.h>
#include <stdio.h>
#include <conio.h>
char getkey(int &dx,int &dy);
class hinh
{
protected:

```

```

    int mau;
public:
    hinh(void)
    {
        mau = 0;
    }
    hinh(int m)
    {
        mau = m;
    }
    virtual void dchuyen(int b)=0;
};

class khoihop : public hinh
{
private:
    int x,y;
    int a ;
public:
    khoihop(void):hinh()
    {
        x=y=a=0;
    }
    khoihop(int m,int x1,int y1, int a1):hinh(m)
    {
        x=x1;
        y=y1;
        a=a1;
    }
    virtual void dchuyen(int b);
    void hien(void)
    {
        setfillstyle(1,mau);
        bar3d(x,y,x+a,y+a,a/2,1);
    }
    void an(void)
    {
        setfillstyle(1,getbkcolor());
        bar(x,y-a/2,x+a/2,y+a/2);
    }
};

class duong:public hinh
{
private:
    int x1,y1,x2,y2;
public:
    duong(void):hinh()
    {
        x1=x2=y1=y2=0;
    }
    duong(int m,int a,int b,int c,int d):hinh(m)
    {

```

```

        x1=a;
        y1=b;
        x2=c;
        y2=d;
    }
    virtual void dchuyen(int b);
    void hien(void)
    {
        setcolor(mau);
        line(x1,y1,x2,y2);
    }
    void an(void)
    {
        setcolor(getbkcolor());
        line(x1,y1,x2,y2);
    }
};

class tron:public hinh
{
    private:
        int x,y,r;
    public:
        tron(void):hinh()
        {
            x=y=r=0;
        }
        tron(int m,int a,int b,int d):hinh(m)
        {
            x=a;
            y=b;
            r=d;
        }
        virtual void dchuyen(int b);
        void hien(void)
        {
            setcolor(mau);
            circle(x,y,r);
        }
        void an(void)
        {
            setcolor(getbkcolor());
            circle(x,y,r);
        }
};

char getkey(int &dx,int &dy)
{
    int ch1,ch2;
    dx dy=0;
    while (1)
    {
        ch1=getch();

```

```

        if (ch1==0)
            ch2=getch();
        if (ch1=='q' || ch1=='Q') return('q');
        if ((ch1==0 && (ch2==80 || ch2==72 || ch2==75 || ch2==77)))
        {
            if (ch2==80) dy=1;
            else if (ch2==72) dy=-1;
            else if (ch2==77) dx=1;
            else dx=-1;
            return(0);
        }
    }
}

void khoihop::dchuyen(int b)
{
    int dx,dy;
    while (1)
    {
        hien();
        if (getkey(dx,dy)=='q') break;
        an();
        x+=b*dx;
        y+=b*dy;
    }
}

void duongq::dchuyen(int b)
{
    int dx,dy;
    while (1)
    {
        hien();
        if (getkey(dx,dy)=='q') break;
        an();
        x1+=b*dx;
        x2+= b*dx;
        y1+=b*dy;
        y2+=b*dy;
    }
}

void tron::dchuyen(int b)
{
    int dx,dy;
    while (1)
    {
        hien();
        if (getkey(dx,dy)=='q') break;
        an();
        x+=b*dx;
        y+=b*dy;
    }
}

```

```

void main()
{
    int mh=0,mode=0;
    initgraph(&mh,&mode,"");
    if (graphresult())
    {
        printf("\n LOI");
        getch();
        exit(0);
    }
    setbkcolor(0);
    // setwritemode(0);
    hình *h[3];
    khoi hop M(4,300,200,15);
    duong D(10,10,10,60,60);
    tron T(14,200,200,50);
    h[0]=&M; h[1]=&D; h[2]=&T;
    for(int i=0;i<3;++i)
        h[i]->dechuyen(10);
    closegraph();
}

```

§4. CÁC LỚP NGĂN XẾP VÀ HÀNG ĐỢI

Chương trình tổ chức thành 4 lớp chính:

1. Lớp container (thùng chứa) gồm 2 thuộc tính:

```

unsigned long count; //Số phần tử trong thùng chứa
void (*errhandler)(); //Con trỏ tới hàm xử lý lỗi

```

2. Lớp s_list thừa kế từ lớp container, có thêm 2 thuộc tính các con trỏ kiểu cấu trúc listnode:

```

struct listnode
{
    void *dataptr;
    listnode *next;
};

listnode *head; // Trỏ tới đầu danh sách
listnode *tail; // Trỏ tới cuối danh sách

```

Các phần tử được chứa trong lớp s_list dưới dạng một danh sách móc nối đơn. Mỗi nút chứa địa chỉ của một phần tử. Do ở đây dùng kiểu con trỏ void nên có thể đưa vào lớp s_list các phần tử có kiểu bất kỳ.

3. Lớp stack thừa kế từ lớp s_list

4. Lớp queue thừa kế từ lớp stack

Các lớp stack và queue không có các thuộc tính riêng. Hai phương thức quan trọng của các lớp này là:

```
virtual int store(void *item) ; // Cất vào một phần tử  
virtual void *retrieve () ; // Lấy ra một phần tử
```

Chú ý: Lớp stack hoạt động theo nguyên tắc LIFO (vào sau ra trước) còn lớp queue hoạt động theo nguyên tắc FIFO (vào trước ra trước).

Chương trình sau minh họa cách dùng liên kết bội, phương thức ảo và con trỏ kiểu void để quản lý các kiểu dữ liệu khác nhau.

Hoạt động của chương trình như sau:

+ Trước tiên lần lượt đưa địa chỉ của biến đối tượng ts1, chuỗi “HA NOI”, biến nguyên a, biến đối tượng ts2 và biến thực x vào ngăn xếp s1 và hàng đợi q1.

+ Thực hiện phép gán các biến đối tượng:

```
s2 = s1 ;
```

```
q2 = q1 ;
```

+ Lấy các phần tử trong ngăn xếp s2 theo trình tự ngược với lúc đưa vào.

+ Lấy các phần tử trong hàng đợi q2 theo trình tự như lúc đưa vào.

```
/* CT10_05.CPP  
Lop vat chua (container)  
Lop danh sach moc noi  
Lop ngăn xếp  
Lop hàng đợi  
Chu y:  
1. constructor sao chép của lớp danh sách  
2. toán tử gán của lớp danh sách  
3. có thể dùng các phương thức khác  
để viết constructor và destructor  
4. Dùng con trỏ this  
*/  
  
#include <stdio.h>  
#include <iostream.h>  
#include <iomanip.h>  
#include <conio.h>  
#include <alloc.h>  
#include <dos.h>  
//Lop container  
class container  
{
```

```

protected:
    unsigned long count; //so pt trong thung chua
    void (*errhandler)();
public:
    container();
    container(const container &c); // Ham tao sao chep
    void operator=(const container &c); // Gan
    unsigned long getcount(); // Cho biet so phan tu
        // Dinh ham xl loi
    void seterrorhandler(void (*userhandler)());
        // 4 phuong thuc thuan ao
    virtual int store(void *item)=0; //Cat mot phan tu
        // vao thung
    virtual void *examine()=0; // Xem gia tri
        // mot phan tu
    virtual void *retrieve ()=0; // Lay mot pt ra
    virtual void empty()=0; // Lam cho thung
        // tro nen rong
};
// Cai dat
// Ham xl loi mac dinh
void defaulthandler();
void defaulthandler()
{
    puts("\nContainer error: memory allocation failure");
}
container::container ()
{
    count=0;
    errhandler= defaulthandler;
}
container::container(const container &c)
{
    count=c.count;
    errhandler=c.errhandler;
}
// Gan
void container::operator=(const container &c)
{
    count=c.count;    errhandler=c.errhandler;
}
// Cho biet so pt
unsigned long container::getcount()
{
    return count;
}

```

```

    // Dinh ham xl loi
void container::seterrorhandler(void (*userhandler)())
{
    errhandler=userhandler;
}

    // lop danh sach moc noi don
class s_list:public container
{
protected:
    //Cau truc mot nut trong ds
    struct listnode
    {
        void *dataptr;
        listnode *next;
    };
    listnode *head;
    listnode *tail;
private:
    // phuong thuc sao chep
    void copy(const s_list &s1);
public:
    s_list();
    s_list(const s_list &s1);
    ~s_list();
    void operator=(const s_list &s1);
        // 4 phuong thuc ao
    virtual int store(void *item)=0; // Cat mot
                                   //phan tu vao thung
    virtual void *examine()=0; // Xem gia tri
                                   // mot phan tu
    virtual void *retrieve ()=0; // Lay mot pt ra
    virtual void empty(); // Lam cho thung tro
                           // nen rong
};

    //Cai dat
void s_list::copy(const s_list &s1)
{
    head=NULL; tail=NULL;
    listnode *temp = s1.head;
    while(temp!=NULL)
    {
        if(head==NULL)
        {
            head= new listnode;
            if(head==NULL) errhandler();
            tail=head;

```

```

    }
    else
    {
        tail->next = new listnode;
        if(tail->next == NULL) errhandler();
        tail = tail->next;
    }
    tail->dataptr= temp->dataptr;
    tail->next= NULL;
    temp = temp->next;
}
}
// constructor
s_list::s_list() : container()
{
    head=NULL;
    tail=NULL;
}
s_list::s_list(const s_list &sl):container(sl)
{
    copy(sl);
}
s_list::~s_list()
{
    this->empty();
}
void s_list::operator=(const s_list &sl)
{
    this->empty();
    count=sl.count;
    copy(sl);
}
void s_list::empty()
{
    listnode *q,*p;
    p = head;
    head=NULL;
    tail=NULL;
    while (p!=NULL)
    {
        q=p;
        p=p->next;
        delete q;
    }
}
// Pop stack

```

```

class stack:public s_list
{
    public:
...    stack();
        stack(const stack &st);
        void operator=(const stack &st);
        virtual int store(void *item); // Cat mot phan tu
                                         // vao thung
        virtual void *examine(); // Xem gia tri mot
                                   // phan tu
        virtual void *retrieve(); // Lay mot pt ra
};
stack::stack():s_list()
{
}
stack::stack(const stack &st):s_list(st)
{
}
void stack::operator=(const stack &st)
{
    this->s_list::operator=(st); //Dung toan tu gan
                                // cua s_list
}
int stack::store(void *item) // Cat mot phan tu
                             // vao thung
{
    //Dua vao dau danh sach
    listnode *p;
    p= new listnode ;
    if(p==NULL) return 1;
    count++;
    p->dataptr=item;
    p->next=head;
    head=p;
    return 0;
}
void *stack::examine() // Xem gia tri mot phan tu
{
    if(count==0) return NULL;
    else
        return head->dataptr;
}
void *stack::retrieve() // Lay mot pt ra
{
    if(count==NULL) return NULL;
    else

```

```

    {
        listnode *p; void *value;
        value = head->dataptr;
        p=head;
        head = p->next;
        delete p;
        count--;
        return value;
    }
}
// Lop queue
class queue:public stack
{
    public:
        queue();
        queue(const queue &q);
        void operator=(const queue &q);
        virtual int store(void *item); // Cat mot
                                         // phan tu vao thung
};
queue::queue(): stack()
{
}
queue::queue(const queue &q):stack(q)
{
}
void queue::operator=(const queue &q)
{
    this->stack::operator=(q); //Dung toan tu
                              // gan cua stack
}
int queue::store(void *item)
{
    // Dat vao cuoi
    listnode *q;
    q=new listnode;
    if(q==NULL)return 1;
    // Bo sung
    q->next=NULL;
    q->dataptr=item;
    if(count==0)
    {
        head=q;
        tail=q;
    }
    else

```

```

    {
        tail->next=q;
        tail=q;
    }
    count++;
    return 0;
}
class TS
{
    private:
        char ht[25];
        int sobd;
        float td;
    public:
        void nhap()
        {
            cout << "\nHo ten: " ;
            fflush(stdin);
            gets(ht);
            cout << "So bao danh: " ;
            cin >> sobd;
            cout << "Tong diem: " ;
            cin >> td;
        }
        void xuat()
        {
            cout << "\nHo ten: " << ht;
            cout << "\nSo bao danh: " << sobd;
            cout << "\nTong diem: "
                << setiosflags(ios::showpoint)
                << setprecision(1)<<setw(5)<< td;
        }
};

// Ham main
void main()
{
    stack s1,s2; queue q1,q2;
    TS ts1,ts2,ts;
    int a=123,b;
    float x=3.14,y;
    char *str;
    clrscr();
    ts1.nhap();
    ts2.nhap();
    //Gui vao
    s1.store(&ts1);

```

```

q1.store(&ts1);
s1.store("HA NOI");
q1.store("HA NOI");
s1.store(&a);
q1.store(&a);
s1.store(&ts2);
q1.store(&ts2);
s1.store(&x);
q1.store(&x);

    //Lay ra tu ngan xep theo nguyen tac LIFO
cout <<"\n\nLay ra tu ngan xep:" ;
s2=s1;
y    *((float*)s2.retrieve());
cout << "\nSo thuc - " <<setiosflags(ios::showpoint)
    << setprecision(2)<< y;
ts = *((TS*)s2.retrieve());
ts.xuat();
b = *((int*)s2.retrieve());
cout << "\nSo nguyen = " << b;
str  (char*)s2.retrieve();
cout << "\nChuoi ky tu: " << str;
ts = *((TS*)s2.retrieve());
ts.xuat();

    //Lay ra tu hang doi theo nguyen tac FIFO
cout <<"\n\nLay ra tu hang doi:" ;
q2=q1;
ts = *((TS*)q2.retrieve());
ts.xuat();
str  (char*)q2.retrieve();
cout << "\nChuoi ky tu: " << str;
b = *((int*)q2.retrieve());
cout << "\nSo nguyen = " << b;
ts = *((TS*)q2.retrieve());
ts.xuat();
y = *((float*)q2.retrieve());
cout << "\nSo thuc = " << setiosflags(ios::showpoint)
    << setprecision(2)<< y;
getch();
}

```

§5. CÁC LỚP SẮP XẾP

Trong tệp C_SORT.H dưới đây sẽ chứa 4 lớp sắp xếp: sort, select_sort, quick_sort và heap_sort. tổng quát hơn. So với các lớp sắp xếp trong mục §7 chương 6 thì các lớp ở đây tổng quát hơn ở chỗ:

+ Các lớp trong mục §7 chương 6 chỉ cho phép sắp xếp một dãy số nguyên theo thứ tự tăng dần.

+ Các lớp dưới đây cho phép sắp xếp một dãy phần tử có kiểu bất kỳ (nguyên, thực, cấu trúc, lớp, ...) và theo một tiêu chuẩn sắp xếp bất kỳ.

1. Lớp `sort` là lớp cơ sở trừu tượng

+ Các thuộc tính:

```
protected:
    void *a ; // Trỏ tới vùng nhớ chứa dãy
                // phần tử cần sắp xếp
    int size; // Độ lớn tính theo byte của phần tử
    int (*nhỏ_hơn)(void* pt1, void* pt2);
                // Con trỏ hàm định nghĩa pt1 nhỏ hơn pt2
```

+ Các phương thức:

```
protected:
    void hoán_vị(int i, int j) ; // Hoán vị các
                                // phần tử thứ i và j
    void * địa_chi (int m); // Cho địa chỉ của
                                // phần tử thứ m

public:
    virtual void sạpxep(void *a1,int n,int itemsize,
        int (*ss_nhỏ_hơn)(void*,void* )) ;
        // Sắp xếp dãy n phần tử chứa trong vùng nhớ a1,
        // mỗi phần tử có độ dài itemsize, thứ tự
        // tăng được quy định bởi hàm ss_nhỏ_hơn
```

2. Lớp `select_sort` dẫn xuất từ lớp `sort`. Lớp này sẽ thực hiện việc sắp xếp theo phương pháp chọn (xem mục §7 chương 6).

+ Các phương thức:

```
public:
    virtual void sạpxep(void *a1,int n,int itemsize,
        int (*ss_nhỏ_hơn)(void*,void* )) ;
        // thực hiện sắp xếp theo phương pháp chọn
```

3. Lớp `quick_sort` dẫn xuất từ lớp `sort`. Lớp này sẽ thực hiện việc sắp xếp theo phương pháp quick sort (xem mục §7 chương 6)

+ Các phương thức:

```
private:
    void q_sort(int l, int r);

public:
    virtual void sạpxep(void *a1,int n,int itemsize,
        int (*ss_nhỏ_hơn)(void*,void* )) ;
        //thực hiện sắp xếp theo phương pháp quick sort
```

4. Lớp heap_sort dẫn xuất từ lớp sort. Lớp này sẽ thực hiện việc sắp xếp theo phương pháp heap sort (xem mục §7 chương 6).

+ Các phương thức:

```
private:
    void shift(int i, int n);
public:
    virtual void sapxep(void *a1,int n,int itemsize,
        int (*ss_nho_hon)(void*,void* ));
        // thực hiện sắp xếp theo phương pháp heap sort
```

Dưới đây là nội dung tệp C_SORT.H

```
//C_SORT.H
// Lop co so truu tuong
// Lop sort
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>
#include <iomanip.h>
#include <mem.h>
class sort
{
protected:
    void *a;
    int size;
    int (*nhohon)(void*,void*);
    void* dia_chi(int m)
    {
        return (void*) ((char*)a + size*(m-1));
    }
    void hoan_vi(int i, int j)
    {
        void *tg, *di, *dj;
        di= dia_chi(i);
        dj= dia_chi(j);
        tg = new char[size];
        memcpy(tg,di,size);
        memcpy(di,dj,size);
        memcpy(dj,tg,size);
    }
public:
    virtual void sapxep(void *a1,int n,int itemsize,
        int (*ss_nho_hon)(void*,void*))
    {
        a= a1;
```

```

        size=n; // Cho C'' hai long
        size=itemsize;
        nho_hon= ss_nho_hon;
    }
} ;

class select_sort : public sort
{
public:
    virtual void sapxep(void *a1,int n,int itemsize,
        int (*ss_nho_hon)(void*,void*)) ;
} ;

void select_sort::sapxep(void *a1,int n,int itemsize,
    int (*ss_nho_hon)(void*,void*))
{
    int i,j,r;
    sort::sapxep(a1,n,itemsize,ss_nho_hon);
    for(i=1; i<n; ++i)
    {
        r=i;
        for(j=i+1; j<=n; ++j)
            if(nho_hon(dia_chi(j),dia_chi(r))) r = j;
        if(r!=i) hoan_vi(i,r);
    }
}

class quick_sort : public sort
{
private:
    void q_sort(int l, int r);
public:
    virtual void sapxep(void *a1,int n,int itemsize,
        int (*ss_nho_hon)(void*,void*));
} ;

void quick_sort::q_sort(int l, int r)
{
    void *x;
    int i,j;
    x = new char[size];
    if(l < r)
    {
        memcpy(x, dia_chi(l), size);
        i = l; j = r+1;
        do
        {
            ++i; --j;
            while(i<r && nho_hon(dia_chi(i),x)) ++i;
            while(nho_hon(x,dia_chi(j)) ) --j ;

```

```

        if(i<j) hoan_vi(i,j);
    } while (i<j);
    hoan_vi(l,j);
    q_sort(l,j-1);
    q_sort(j+1,r);
}
}

void quick_sort::sapxep(void *al,int n,int itemsize,
                        int (*ss_nho_hon)(void*,void*))
{
    sort::sapxep(al,n,itemsize,ss_nho_hon);
    q_sort(l,n);
}

class heap_sort : public sort
{
private:
    void shift(int i, int n);
public:
    virtual void sapxep(void *al,int n,int itemsize,
                        int (*ss_nho_hon)(void*,void*));
};

void heap_sort::shift(int i, int n)
{
    int l,r,k;
    l = 2*i;
    r = l+1;
    if(l>n) return;
    if(l==n)
    {
        if (nho_hon(dia_chi(l), dia_chi(l)))
        {
            hoan_vi(l,l);
            return;
        }
    }
    if(nho_hon(dia_chi(r), dia_chi(l)))
        k = l;
    else
        k = r;
    if (!nho_hon(dia_chi(i), dia_chi(k)))
        return;
    else
    {
        hoan_vi(i,k);
        shift(k,n);
    }
}

void heap_sort::sapxep(void *al,int n,int itemsize,

```

```

int (*ss_nho_hon)(void*,void*)
{
    long i;
    sort::sapxep(a1,n,itemsiz,ss_nho_hon);
    // Tao dong
    for(i=n/2 ; i>=1; --i) shift(i,n);
    // lap
    for(i=n ; i>=2; --i)
    {
        hoan_vi(1,i);
        shift(1,i-1);
    }
}

```

§6. VÍ DỤ VỀ CÁC LỚP SẮP XẾP

Trong mục này trình bày 2 chương trình minh họa cách dùng các lớp nói trên. Chương trình thứ nhất minh họa cách sử dụng các lớp trong tệp C_SORT.H để sắp xếp một dãy thí sinh theo thứ tự giảm và thứ tự tăng của tổng điểm. Chương trình thứ hai minh họa cách dùng các lớp trong C_SORT.H để sắp xếp một dãy số nguyên theo chiều tăng và chiều giảm.

Chương trình 1

```

//Crl0-08
// Lop co so truu tuong
// Lop sort
#include "c_sort.h"
class TS
{
private:
    char ht[25];
    int sobd;
    float td;
public:
    float get td()
    {
        return td;
    }
    void nhap()
    {
        cout << "\nHo ten: " ;
        fflush(stdin);
        gets(ht);
        cout << "So bao danh: " ;
        cin >> sobd;
    }
}

```

```

        cout << "Tong diem: " ;
        cin >> td;
    }
    void xuat()
    {
        cout << "\nHo ten: " << ht;
        cout << "\nSo bao danh: " << sobd;
        cout << "\nTong diem: " <<
            setiosflags(ios::showpoint)
            << setprecision(1)<<setw(5)<< td;
    }
};

int ss_tong_diem_giam(void *ts1, void *ts2)
{
    return (((TS*)ts1)->get_td())>(((TS*)ts2)->get_td());
}

int ss_tong_diem_tang(void *ts1, void *ts2)
{
    return (((TS*)ts1)->get_td())<(((TS*)ts2)->get_td());
}

void main()
{
    TS t[100];
    sort *sa;
    int n,i;
    clrscr();
    cout << "\nSo thi sinh: ";
    cin >> n;
    for(i=1; i<=n; ++i) t[i].nhap();
    for(i=1; i<=n; ++i) t[i].xuat();
    getch();
    cout << "\n\nSap xep giam theo tong diem
        - PP Select Sort" ;
    sa= new select_sort;
    sa->sapxep( t+1,n,sizeof(TS),ss_tong_diem_giam);
    for(i=1; i<=n; ++i) t[i].xuat();
    delete sa;
    getch();
    cout << "\n\nSap xep tang theo tong diem
        - PP Select Sort";
    sa= new select_sort;
    sa->sapxep( t+1,n,sizeof(TS),ss_tong_diem_tang);
    for(i=1; i<=n; ++i) t[i].xuat();
    delete sa;
    getch();
    cout << "\n\nSap xep giam theo tong diem

```

```

        - PP Quick Sort" ;
sa= new quick_sort;
sa->sapxep( t+1,n,sizeof(TS),ss_tong_diem_giam);
for(i=1; i<=n; ++i) t[i].xuat();
delete sa;
getch();
cout << "\n\nSap xep tang theo tong diem
        - PP Quick Sort" ;
sa= new quick_sort;
sa->sapxep( t+1,n,sizeof(TS),ss_tong_diem_tang);
for(i=1; i<=n; ++i) t[i].xuat();
delete sa;
getch();
cout << "\n\nSap xep giam theo tong diem
        - PP Heap Sort" ;
sa= new heap_sort;
sa->sapxep( t+1,n,sizeof(TS),ss_tong_diem_giam);
for(i=1; i<=n; ++i) t[i].xuat();
delete sa;
getch();
cout << "\n\nSap xep tang theo tong diem
        - PP Heap Sort" ;
sa= new heap_sort;
sa->sapxep( t+1,n,sizeof(TS),ss_tong_diem_tang);
for(i=1; i<=n; ++i) t[i].xuat();
delete sa;
getch();
}

```

Chương trình 2

```

//CT10-09
// Lop co so tru tuong
// Lop sort
#include "c_sort.h"
int ss_tang(void *i1,void *i2)
{
    return *((int*)i1) < *((int*)i2);
}
int ss_giam(void *i1,void *i2)
{
    return *((int*)i1) > *((int*)i2);
}
void main()
{
    int i,n;
    struct time t1,t2;

```

```

int b[20],a[20], k, tg, sec, hund;
n=10;
sort *s[3];
select_sort ss;
quick_sort qs;
heap_sort hs;
s[0]=&ss;
s[1]=&qs;
s[2]=&hs;
clrscr();
srand(5000);
for(i=1;i<=n;++i)
b[i]=rand();
cout<<"\nDay ban dau\n ";
for(i=1;i<=n;++i) cout <<b[i]<<" ";
cout<<"\n\nCac day tang sap xep theo ";
cout << "select_sort, quick_sort, heap_sort\n";
for(k=0; k<3; ++k)
{
    for(i=1;i<=n;++i)
        a[i]=b[i];
    s[k]->sapxep (a+1,n,sizeof(int),ss_tang);
    // In
    for(i=1;i<=n;++i) cout <<a[i]<<" ";
    cout<<"\n";
}
cout<<"\n\nCac day giam sap xep theo ";
cout << "select_sort, quick_sort, heap_sort\n";
for(k=0; k<3; ++k)
{
    for(i=1;i<=n;++i)
        a[i]=b[i];
    s[k]->sapxep (a+1,n,sizeof(int),ss_giam);
    //In
    for(i=1;i<=n;++i) cout <<a[i]<<" ";
    cout << "\n";
}
getch();
}

```

THỨ TỰ ƯU TIÊN CỦA CÁC PHÉP TOÁN

Các phép toán được chia thành 16 nhóm. Các phép toán trong cùng nhóm có mức độ ưu tiên như nhau.

Về trình tự kết hợp thì:

+ Các phép tính của nhóm 2, nhóm 14 và toán tử gán (nhóm 15) kết hợp từ phải sang trái.

+ Các phép toán còn lại kết hợp từ trái qua phải.

1. Nhóm một

- () Gọi hàm (Function call)
- [] Chỉ số mảng (Array subscript)
- > Chọn gián tiếp một thành phần (indirect component selector)
- :: Xác định phạm vi truy nhập (scope access/resolution)
- . Chọn trực tiếp một thành phần (direct component selector)

2. Nhóm hai

- () Gọi hàm (Function call)
- ! Phủ định logic (Logical negation -NOT)
- ~ Lấy phần bù theo bit (Bitwise (1's) complement)
- + Dấu cộng (Unary plus)
- Dấu trừ (Unary minus)
- ++ Phép tăng một (Preincrement or postincrement)
- Phép giảm một (Predecrement or postdecrement)
- & Phép lấy địa chỉ (Address)
- * Truy nhập gián tiếp (Indirection)
- sizeof Cho kích thước của toán hạng (returns size of operand, in bytes)
- new Cấp phát bộ nhớ động (dynamically allocates C⁺⁺ storage)
- delete Giải phóng bộ nhớ (dynamically deallocates C⁺⁺ storage)

3. Nhóm ba

- * Nhân (Multiply)
- / Chia (Divide)
- % Lấy phần dư (Remainder - modulus)

4. Nhóm bốn

- * Gọi gián tiếp tới thành phần từ một biến đối tượng
- >* Gọi gián tiếp tới thành phần từ một con trỏ đối tượng

5. Nhóm năm

- + Cộng (Binary plus)
- Trừ (Binary minus)

6. Nhóm sáu

- << Dịch trái (Shift left)
- >> Dịch phải (Shift right)

7. Nhóm bảy

- < Nhỏ hơn (Less than)
- <= Nhỏ hơn hoặc bằng (Less than or equal to)
- > Lớn hơn (Greater than)
- >= Lớn hơn hoặc bằng (Greater than or equal to)

8. Nhóm tám

- = = Bằng (Equal to)
- != Không bằng (Not equal to)

9. Nhóm chín

- & Phép và theo bit (Bitwise AND)

10. Nhóm mười

- ^ Phép hoặc loại trừ theo bit (Bitwise XOR)

11. Nhóm mười một

- | Phép hoặc theo bit (Bitwise OR)

12. Nhóm mười hai

- && Phép và logic (Logical AND)

13. Nhóm mười ba

- || Phép hoặc logic (Logical OR)

14. Nhóm mười bốn

- ? : Toán tử điều kiện (a ? x : y means "if a then x, else y")

15. Nhóm mười năm

- = Phép gán đơn giản (Simple assignment)
- *= Phép gán sau khi nhân (Assign product)
- /= Phép gán sau khi chia (Assign quotient)

- %= Phép gán sau khi lấy phần dư (Assign remainder)
- += Phép gán sau khi cộng (Assign sum)
- = Phép gán sau khi trừ (Assign difference)
- &= Phép gán sau khi AND theo bit (Assign bitwise AND)
- ^= Phép gán sau khi XOR theo bit (Assign bitwise XOR)
- |= Phép gán sau khi OR theo bit (Assign bitwise OR)
- <<= Phép gán sau khi dịch trái (Assign left shift)
- >>= Phép gán sau khi dịch phải (Assign right shift)

16. Nhóm mười sáu

, Toán tử phẩy dùng để phân cách các phần tử

Tất cả các toán tử nói trên đều có thể định nghĩa chồng trừ các toán tử sau:

- . Chọn trực tiếp một thành phần
- * Gọi gián tiếp tới thành phần từ một biến đối tượng
- :: Toán tử xác định phạm vi truy nhập
- ? Toán tử điều kiện

Phụ lục 2

CÁC TỪ KHOÁ CỦA C++

asm	double	new	switch
auto	else	operator	template
break	enum	private	this
case	extern	protected	throw
catch	float	public	try
char	for	register	typedef
class	friend	return	union
const	goto	short	unsigned
continue	if	signed	virtual
default	inline	sizeof	void
delete	int	static	volatile
do	long	struct	while

BẢNG MÃ ASCH VÀ MÃ QUÉT

1. Bảng mã ASCII

Bộ ký tự ASCII gồm 256 ký tự được phân bố như sau:

+ 32 ký tự đầu tiên là các ký tự điều khiển không in được như ký tự Enter (mã 13), ký tự ESC (mã 27).

+ Các mã ASCII 32-47, 58-64, 91-96 và 123-127 là các ký tự đặc biệt như dấu chấm, dấu phẩy, dấu cách, dấu ngoặc, dấu móc, dấu hỏi,...

+ Các mã ASCII 48-57 là 10 chữ số

+ Các mã ASCII 65-90 là các chữ cái hoa từ A đến Z

+ Các mã ASCII 97-122 là các chữ cái thường từ a đến z

Lưu ý: Chữ thường có mã ASCII lớn hơn 32 so với chữ hoa tương ứng. Ví dụ mã ASCII của a là 97 còn mã ASCII của A là 65.

+ Các mã ASCII 128-255 là các ký tự đồ họa.

Bảng sau cho mã ASCII của 128 ký tự đầu tiên. Để nhận được các ký tự đồ họa (có mã từ 128 đến 255) có thể dùng chương trình sau:

```
// In các ký tự đồ họa lên màn hình
#include <stdio.h>
#include <conio.h>
main()
{
    int i;
    clrscr();
    for (i=128; i<=255; ++i)
        printf("%6d%2c", i, i);
}
```

BẢNG MÃ ASCII

MÃ (Số TT)	KÝ TỰ	MÃ (Số TT)	KÝ TỰ	MÃ (Số TT)	KÝ TỰ
0	NUL	26	SUB	52	4
1	SOH	27	ESC	53	5
2	STX	28	FS	54	6
3	ETX	29	GS	55	7

MÃ (Số TT)	KÝ TỰ	MÃ (Số TT)	KÝ TỰ	MÃ (Số TT)	KÝ TỰ
4	EOT	30	RS	56	8
5	ENQ	31	US	57	9
6	ACK	32	Space	58	:
7	BEL	33	!	59	;
8	BS	34	“	60	<
9	HT	35	#	61	=
10	LF	36	\$	62	>
11	VT	37	%	63	?
12	FF	38	&	64	@
13	CR	39	'	65	A
14	SO	40	(	66	B
15	SI	41	)	67	C
16	DLE	42	*	68	D
17	DC1	43	+	69	E
18	DC2	44	,	70	F
19	DC3	45	-	71	G
20	DC4	46	.	72	H
21	NAK	47	/	73	I
22	SYN	48	0	74	J
23	ETB	49	1	75	K
24	CAN	50	2	76	L
25	EM	51	3	77	M
78	N	95	_	112	p
79	O	96	*	113	q
80	P	97	a	114	r
81	Q	98	b	115	s
82	R	99	c	116	t
83	S	100	d	117	u
84	t	101	e	118	v
85	U	102	f	119	w
86	V	103	g	120	x

MÃ (Số TT)	KÝ TỰ	MÃ (Số TT)	KÝ TỰ	MÃ (Số TT)	KÝ TỰ
87	W	104	h	121	y
88	X	105	i	122	z
89	Y	106	J	123	{
90	Z	107	k	124	
91	[	108	l	125	}
92	\	109	m	126	~
93	]	110	n	127	DEL
94	^	111	o		

2. Bảng mã scan từ bàn phím

Mỗi phím trên bàn phím của IBM PC đều được gán một con số, gọi là mã scan, từ 1 đến 83. IBM PC AT dùng một nhóm mã khác, từ 1 đến 108 các mã này bắt đầu bằng các phím số, các phím chữ, rồi đến các phím chức năng và cuối cùng là các phím cho con trỏ, khi một phím được nhấn thì bộ xử lý của bàn phím gửi cho CPU mã scan tương ứng, khi nó được nhà thì mã scan cộng thêm 80 hex sẽ được gửi tiếp cho CPU.

Hex	Thập phân	Phím của PC	Phím của PC-AT
1	1	ESC	Tilde
2-B	2-11	1-9,0	1-9,0
C	12	trừ, gạch dưới	trừ, gạch dưới
D	13	=, +	=, +
E	14	Backspace	\, thanh đứng
F	15	Tab	Backspace
10	16	Q	Tab
11	17	W	Q
12	18	E	W
13	19	R	E
14	20	T	R
15	21	Y	T
16	22	U	Y
17	23	I	U

Hex	Thập phân	Phím của PC	Phím của PC-AT
18	24	O	I
19	25	P	O
1A	26	[	P
1B	27	]	[
1C	28	Enter	]
1D	29	Ctrl	
1E	30	A	Ctrl
1F	31	S	A
20	32	D	S
21	33	F	D
22	34	G	F
23	35	H	G
24	36	J	H
25	37	K	J
26	38	L	K
27	39	Chấm phẩy, :	L
28	40	Nháy	Chấm phẩy,:
29	41	Title	Nháy
2A	42	Shift	trái
2B	43	\, thanh đứng	Enter
2C	44	Z	Shift trái
2D	45	X	
2E	46	C	Z
2F	47	V	X
30	48	B	C
31	49	N	V
32	50	M	B
33	51	Phẩy	N
34	52	Chấm	M
35	53	/,?	Phẩy
36	54	Shift phải	Chấm

Hex	Thập phân	Phím của PC	Phím của PC-AT
37	55	*, PrtScr	/, ?
38	56	Alt	
39	57	Space bar	Shift phải
3A	58	Caps Lock	Alt
3B	59	F1	
3C	60	F2	
3D	61	F3	Space bar
3E	62	F4	
3F	63	F5	
40	64	F6	Caps Lock
41	65	F7	F2
42	66	F8	F4
43	67	F9	F6
44	68	F10	F8
45	69	Num Lock	F10
46	70	Scroll Lock,Break	F1
47	71	Home	F3
48	72	mũi tên lên	F5
49	73	PgUp	F7
4A	74	Đấu trừ bàn tính	F9
4B	75	Mũi tên trái	
4C	76	5	của bàn tính
4D	77	Mũi tên phải	
4F	79	End	
50	80	Mũi tên xuống	
51	81	PgDn	
52	82	Ins	
53	83	Del	
5A	90		ESC
5B	91		Home
5C	92		Mũi tên trái
5D	93		End

Hex	Thập phân	Phím của PC	Phím của PC-AT
5F	95		Num Lock
60	96		Mũi tên lên
61	97		5 của bàn tính
62	98		Mũi tên xuống
63	99		Ins
64	100		Scroll Lock
65	101		PgUp
66	102		Mũi tên phải
67	103		PgDn
68	104		Del
69	105		Sys
6A	106		*, PrtScr
6B	107		-
6C	108		+

HÀM VỚI ĐỐI SỐ BẤT ĐỊNH TRONG C

Trong các giáo trình C thường chỉ hướng dẫn cách xây dựng hàm với các đối cố định. Mỗi đối cần có một tham số (cùng kiểu với nó) trong lời gọi hàm. Tuy nhiên một vài hàm chuẩn của C lại không như vậy, mà linh hoạt hơn, chẳng khi dùng hàm printf hay scanf thì số tham số mà ta cung cấp cho hàm là không cố định cả về số lượng lẫn kiểu cách. Ví dụ trong câu lệnh:

```
printf("\n Tổng = %d ", 3+4+5);
```

có 2 tham số, nhưng trong câu lệnh:

```
printf("\n Hà Nội" );
```

chỉ có một tham số.

Như vậy cần phân biệt các khái niệm sau:

- Đối số cố định được khai báo trong dòng đầu của hàm, nó có tên và kiểu
- Tham số ứng với đối số cố định gọi là tham số cố định
- Đối bất định được khai báo bởi ba dấu chấm; bất định cả về số lượng và kiểu
- Tham số bất định (ứng với đối bất định) là một danh sách giá trị với số lượng và kiểu tùy ý (không xác định)

Trong phụ lục này sẽ trình bày cách xây dựng các hàm với đối số bất định. Công cụ chủ yếu được dùng là con trỏ và danh sách.

1. Biến con trỏ

Biến con trỏ (hay con trỏ) dùng để chứa địa chỉ của biến, mảng, hàm, ... Có nhiều kiểu địa chỉ. vì vậy cũng có nhiều kiểu con trỏ. Biến con trỏ được khai báo theo mẫu:

```
Kiểu     *Tên_biến_con_trỏ ;
```

Ví dụ:

```
float px ; // px là con trỏ thực
```

Các phép toán quan trọng trên con trỏ gồm:

- + Gán địa chỉ một vùng nhớ cho con trỏ (dùng toán tử gán, phép lấy địa chỉ, các hàm cấp phát bộ nhớ)

+ Truy nhập vào vùng nhớ thông qua con trỏ, dùng phép toán:

`*Tên_con_trỏ`

(Đề ý ở đây có 2 vùng nhớ: vùng nhớ của biến con trỏ và vùng nhớ mà địa chỉ đầu của nó chứa trong biến con trỏ)

+ Cộng địa chỉ để con trỏ chứa địa chỉ của phần tử tiếp theo, dùng phép toán:

`++ Tên_con_trỏ` hoặc `Tên_con_trỏ ++`

Chú ý rằng các phép toán trên chỉ có thể thực hiện đối với con trỏ có kiểu.

2. Danh sách không cùng kiểu

Dùng con trỏ có kiểu chỉ quản lý được một danh sách giá trị cùng kiểu, ví dụ dãy số thực, dãy số nguyên, dãy các cấu trúc,....

Khi cần quản lý một danh sách các giá trị không cùng kiểu ta phải dùng con trỏ không kiểu (void) khai báo như sau:

`void * Tên_con_trỏ;`

Con trỏ void có thể chứa các địa chỉ có kiểu bất kỳ, và dùng để trỏ đến vùng nhớ chứa danh sách cần quản lý. Một chú ý quan trọng là mỗi khi gửi vào hay lấy ra một giá trị từ vùng nhớ, thì tùy theo kiểu giá trị mà ta phải dùng phép chuyển kiểu thích hợp đối với con trỏ. Ví dụ sau minh họa cách lập một danh sách gồm một số nguyên, một số thực và một chuỗi ký tự. Chúng ta cần một bộ nhớ để chứa số nguyên, số thực và địa chỉ chuỗi và dùng các con trỏ void để quản lý vùng nhớ này.

```
void *list, *p ; // Con trỏ list trỏ tới đầu danh sách
// p dùng để duyệt qua các phần tử của danh sách
list=malloc(sizeof(int) + sizeof(float)+ sizeof(char*));
p=list;
*((int*)p) = 12; // Đưa số nguyên 12 vào danh sách
((int*)p)++; // Chuyển sang phần tử tiếp theo
*((float*)p) = 3.14; // Đưa số thực 3.14 vào danh sách
((float*)p)++; // Chuyển sang phần tử tiếp theo
*((char**)p) = "HA NOI"; // Đưa địa chỉ chuỗi "HA NOI"
// vào danh sách
// Nhận các phần tử trong danh sách
p=list; // về đầu danh sách
int a = *((int*)p); // Nhận phần tử thứ nhất
((int*)p)++; // Chuyển sang phần tử tiếp theo
float x= *((float*)p); // Nhận phần tử thứ hai
```

```
((float*)p)++; // Chuyển sang phần tử tiếp theo
char *str = *((char**)p); // Nhận phần tử thứ ba
```

3. Hàm với đối số bất định

+ Đối số bất định bao giờ cũng đặt sau cùng và được khai báo bằng dấu ba chấm. Ví dụ ví dụ hàm

```
void f(int n, char *s, ...);
```

có 2 đối số cố định là n, s và đối số bất định.

+ Để nhận được các tham số bất định trong lời gọi hàm ta cần lưu ý các điểm sau:

- Các tham số bất định chứa trong một danh sách. Để nhận được địa chỉ đầu danh sách ta dùng một con trỏ void và phép gán sau:

```
void *list;
list = ...;
```

- Dùng một tham số cố định kiểu chuỗi để quy định số lượng và kiểu của mỗi tham số trong danh sách, ví dụ:

“3i” hiểu là : tham số bất định gồm 3 giá trị int

“3f” hiểu là : tham số bất định gồm 3 giá trị float

“fiss” hiểu là có 4 tham số bất định có kiểu lần lượt là float, int, char*, char*

Một khi đã biết được địa chỉ đầu danh sách, biết được số lượng và kiểu của mỗi tham số, thì dễ dàng nhận được giá trị các tham số để sử dụng trong thân hàm.

Ví dụ sau đây minh họa cách xây dựng các hàm với tham số bất định. Hàm dùng để in các giá trị kiểu int, float và char. Hàm có một tham số cố định để cho biết có bao nhiêu giá trị và kiểu các giá trị cần in. Kiểu quy định như sau: i là int, f là float, s là char*. Tham số có 2 cách viết: lặp (gồm một hằng số nguyên và một chữ cái định kiểu) và liệt kê (một dãy các chữ cái định kiểu).

Ví dụ:

“4s” có nghĩa in 4 chuỗi

“siif” có nghĩa in một chuỗi, 2 giá trị nguyên và một giá trị thực:

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <conio.h>
#include <stdlib.h>
#include <stdarg.h>
```

```

void InDanhSachGiaTri(char *st,...)
{
    void *list ;
    int gt_int ;
    float gt_float;
    char *gt_str;
    int n,i ;
    char kieu;
    int lap;
    list = ... ; // list tro toi vung nho chua
                  // danh sach dia chi cac tham so
    lap = isdigit(st[0]) ;
    if (lap)
        n=st[0] - '0' ;
    else
        n=strlen(st);
    printf("\n n= %d lap = %d",n,lap); getch();
    for(i=0;i<n;i++)
    {
        if(lap)
            kieu=st[i];
        else
            kieu = st[i];
        printf("\nKieu= %c",kieu); getch();
        switch(kieu)
        {
            case 'i' :
                gt_int = *((int*)list);
                if(!lap)
                    ((int*)list)++ ;
                printf("\nGia tri %d = %d",i,gt_int);
                break;
            case 'f' :
                gt_float = (float) *((double*)list);
                if(!lap)
                    ((double*)list)++ ;
                printf("\nGia tri %d = %0.2f",i,gt_float);
                break;
            case 's' :
                gt_str = *((char**)list) ;
                if(!lap)
                    ((char**)list)++ ;
                printf("\nGia tri %d = %s",i,gt_str);
            }
        }
    }
}

```

```

void main()
{
    float x=3.14;
    int a=123;
    char *tp="HAI PHONG";
    InDanhSachGiaTri("4i",a);
    InDanhSachGiaTri("4s","HA NOI");
    InDanhSachGiaTri("ifsssfii", a, x, tp, tp,"QUY NHON",
                    x, 6.28, a, 246);
    InDanhSachGiaTri("4f",6.28);
    getch();
}

```

4. Hàm không đối và hàm với đối bất định

Nhiều người nghĩ hàm khai báo như sau

```
void f();
```

là hàm không đối trong C. Trong C++ thì hiểu như thế là đúng, còn trong C thì đó là hàm có đối bất định (hàm không đối trong C khai báo như sau: `f(void)`). Do không có đối cố định nào cho biết về số lượng và kiểu của các tham số bất định, nên giải pháp ở đây là dùng các biến toàn bộ. Rõ ràng giải pháp này không thuận tiện cho người dùng vì phải khai báo đúng tên biến toàn bộ và phải khởi gán giá trị cho nó trước khi gọi hàm. Ví dụ trình bày một hàm chỉ có đối bất định dùng để tính max và min của các giá trị thực. Các tham số bất định được đưa vào theo trình tự sau: Địa chỉ chứa max, địa chỉ chứa min, các giá trị nguyên cần tính max, min. Chương trình dùng biến toàn bộ N để cho biết số giá trị nguyên cần tính max, min.

```

int N;
void maxmin()
{
    void *lt = ... ;
    float *max, *min, tg;
    int i;
    max = *((float**)lt)++;
    min = *((float**)lt)++;
    *max = *min - (float) *((double*)lt)++;
    for(i=1; i<N; ++i)
    {
        tg= (float) *((double*)lt)++;
        if(tg > *max) *max = tg;
        if(tg < *min) *min = tg;
    }
}

```

TÓM TẮT CÁC HÀM CỦA TURBO C THEO THỨ TỰ ABC

1. _chmod	<io.h>
2. _close	<io.h>
3. _creat	<io.h>
4. _open	<io.h>
5. abort	<process.h>
6. abs	<stdlib.h>
7. acos	<math.h>
8. arc	<graphics.h>
9. asin	<math.h>
10. atan	<math.h>
11. atan2	<math.h>
12. atof	<ctype.h>
13. atoi	<ctype.h>
14. atol	<ctype.h>
15. bar	<graphics.h>
16. bar3d	<graphics.h>
17. cabs	<math.h>
18. calloc	<alloc.h>
19. ceil	<math.h>
20. chdir	<dir.h>
21. chmod	<io.h>
22. circle	<graphics.h>
23. cleardevive	<graphics.h>
24. clearviewport	<graphics.h>
25. close	<io.h>
26. circol	<conio.h>
27. clrscr	<conio.h>
28. coreleft	<alloc.h>
29. cos	<math.h>
30. cosh	<math.h>
31. cprintf	<conio.h>
32. creat	<io.h>
33. cscanf	<conio.h>
34. delay	<dos.h>
35. delline	<conio.h>
36. disable	<dos.h>
37. drawpoly	<graphics.h>
38. ecvt	<ctype.h>
39. ellipse	<graphics.h>
40. enable	<dos.h>
41. exit	<process.h>

42. exp	<math.h>
43. fabs	<math.h>
44. fclose	<stdio.h>
45. fcloseall	<stdio.h>
46. fcvrt	<ctype.h>
47. fcof	<stdio.h>
48. ferror	<stdio.h>
49. fflush	<stdio.h>
50. fflushall	<stdio.h>
51. fgetc	<stdio.h>
52. fgets	<stdio.h>
53. fillpopy	<graphics.h>
54. findfirst	<dir.h>
55. findnext	<dir.h>
56. floodfill	<graphics.h>
57. floor	<math.h>
58. fmode	<math.h>
59. fopen	<stdio.h>
60. FP_OFF	<dos.h>
61. FP_SEG	<dos.h>
62. fprintf	<stdio.h>
63. fprintf	<stdio.h>
64. fputc	<stdio.h>
65. fputs	<stdio.h>
66. fread	<stdio.h>
67. free	<alloc.h>
68. fscanf	<stdio.h>
69. fseek	<stdio.h>
70. fteel	<stdio.h>
71. fwrite	<stdio.h>
72. gcvt	<ctype.h>
73. geninterrupt	<dos.h>
74. getbkcolor	<graphics.h>
75. getc	<stdio.h>
76. getch	<conio.h>
77. getchar	<stdio.h>
78. getche	<conio.h>
79. getcolor	<graphics.h>
80. getcwd	<dir.h>
81. getdate	<time.h>
82. getimage	<graphics.h>
83. getlinesettings	<graphics.h>
84. getmaxcolor	<graphics.h>
85. getmaxx	<graphics.h>
86. getmaxy	<graphics.h>
87. getpalette	<graphics.h>
88. getpixel	<graphics.h>

89. gets	<stdio.h>
90. gettextinfo	<conio.h>
91. gettime	<dos.h>
92. gettime	<time.h>
93. getvect	<dos.h>
94. getviewport	<graphics.h>
95. getw	<stdio.h>
96. gotoxy	<conio.h>
97. gotoxy	<conio.h>
98. grapherrormsg	<graphics.h>
99. graphresult	<graphics.h>
100. imagesize	<graphics.h>
101. initgraph	<graphics.h>
102. int86	<dos.h>
103. int86x	<dos.h>
104. intdos	<dos.h>
105. intdosx	<dos.h>
106. intr	<dos.h>
107. inxdigit	<ctype.h>
108. isalnum	<ctype.h>
109. isalpha	<ctype.h>
110. iscntrl	<ctype.h>
111. isdigit	<ctype.h>
112. isgraph	<ctype.h>
113. islower	<ctype.h>
114. isprint	<ctype.h>
115. ispunct	<ctype.h>
116. isspace	<ctype.h>
117. isupper	<ctype.h>
118. itoa	<ctype.h>
119. kbhit	<conio.h>
120. keep	<dos.h>
121. labs	<stdlib.h>
122. line	<graphics.h>
123. linerel	<graphics.h>
124. lineto	<graphics.h>
125. log	<math.h>
126. log10	<math.h>
127. lseek	<io.h>
128. ltoa	<ctype.h>
129. malloc	<alloc.h>
130. memccpy	<memory.h> hoặc <string.h>
131. memchr	<memory.h> hoặc <string.h>
132. memcmp	<memory.h> hoặc <string.h>
133. memcpy	<memory.h> hoặc <string.h>
134. memicmp	<memory.h> hoặc <string.h>
135. memset	<memory.h> hoặc <string.h>

136. MK_FP	<dos.h>
137. mkdir	<dir.h>
138. movedata	<mem.h>
139. movedata	<memory.h> hoặc <string.h>
140. moveto	<graphics.h>
141. nosound	<dos.h>
142. open	<io.h>
143. outtext	<graphics.h>
144. outtextxy	<graphics.h>
145. peek	<dos.h>
146. peekb	<dos.h>
147. perror	<stdio.h>
148. pieslice	<graphics.h>
149. poke	<dos.h>
150. pokeb	<dos.h>
151. pow	<math.h>
152. printf	<stdio.h>
153. putc	<stdio.h>
154. putch	<conio.h>
155. putchar	<stdio.h>
156. putimage	<graphics.h>
157. putpixel	<graphics.h>
158. puts	<stdio.h>
159. putw	<stdio.h>
160. rand	<stdlib.h>
161. random	<stdlib.h>
162. randomize	<stdlib.h> và <time.h>
163. read	<io.h>
164. realloc	<alloc.h>
165. rectangle	<graphics.h>
166. remove	<stdio.h>
167. rewind	<stdio.h>
168. rmdir	<dir.h>
169. scanf	<stdio.h>
170. seqread	<dos.h>
171. setbkcolor	<graphics.h>
172. setcolor	<graphics.h>
173. setdate	<time.h>
174. setfillstyle	<graphics.h>
175. setlinestyle	<graphics.h>
176. setpalette	<graphics.h>
177. settextjustify	<graphics.h>
178. settextstyle	<graphics.h>
179. settime	<time.h>
180. setvect	<dos.h>
181. setviewport	<graphics.h>
182. setwritemode	<graphics.h>

183. sin	<math.h>
184. sinh	<math.h>
185. sleep	<dos.h>
186. sound	<dos.h>
187. sprintf	<stdio.h>
188. sqrt	<math.h>
189. srand	<stdlib.h>
190. strcat	<string.h>
191. strchr	<string.h>
192. strcmp	<string.h>
193. strcmpi	<string.h>
194. strcpy	<string.h>
195. strcpyn	<string.h>
196. strdup	<string.h>
197. strcmp	<string.h>
198. strlen	<string.h>
199. strlwr	<string.h>
200. strncat	<string.h>
201. strncmp	<string.h>
202. strncpy	<string.h>
203. strncpy	<string.h>
204. strnset	<string.h>
205. strpbrk	<string.h>
206. strrchr	<string.h>
207. strrev	<string.h>
208. strset	<string.h>
209. strspn	<string.h>
210. strstr	<string.h>
211. strupr	<string.h>
212. system	<process.h>
213. tan	<math.h>
214. tanh	<math.h>
215. textbackground	<conio.h>
216. textcolor	<conio.h>
217. textheight	<graphics.h>
218. textmode	<conio.h>
219. textwidth	<graphics.h>
220. time	<time.h>
221. tolower	<ctype.h>
222. toupper	<ctype.h>
223. ultoa	<ctype.h>
224. unlink	<stdio.h>
225. wherex	<conio.h>
226. wherey	<conio.h>
227. window	<conio.h>
228. write	<io.h>

PHÂN TÍCH, THIẾT KẾ VÀ LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

§1. PHÂN TÍCH HƯỚNG ĐỐI TƯỢNG

1.1. Giới thiệu

Phân tích hệ thống không chỉ có liên quan chặt chẽ với sự xuất hiện của máy tính, mà thực tế nhu cầu phân tích đã có trước khi máy tính xuất hiện từ nhiều thế kỷ. Khi các Vua Pharaon của Ai Cập cổ đại xây dựng các Kim Tự Tháp, thì những người thiết kế Kim Tự Tháp có thể được coi như những nhà thiết kế hệ thống, những kiến trúc sư đại tài, còn những người tổ chức vận chuyển nguyên vật liệu, huy động nhân công xây dựng Kim Tự Tháp, theo một nghĩa nào đó, chính là những người phân tích hệ thống. Từ giữa thế kỷ trước, các nhà tư bản, các doanh nghiệp muốn có lợi nhuận cao đã phải tiến hành nghiên cứu phương pháp, cách tổ chức, phân công lao động hợp lý để cho các hệ thống sản xuất, kinh doanh hoạt động đạt hiệu quả cao hơn. Chính họ đã thực hiện phân tích những hệ thống đó để đề ra những phương pháp quản lý, cách tổ chức mới, tốt hơn.

Cùng với sự phát triển của công nghiệp điện tử, giá thành phần cứng giảm nhiều, nhưng giá phần mềm lại tăng. Nhất là phí tổn cho bảo trì để hệ thống đáp ứng được nhu cầu của người sử dụng lại chiếm một tỷ trọng rất lớn trong tổng chi phí cho một dự án phát triển phần mềm. Điều này cho thấy vai trò của công việc phân tích hệ thống là rất quan trọng và cần thiết phải tìm ra phương pháp tốt hơn cho việc phát triển hệ thống.

Phân tích làm nhiệm vụ phân tách bài toán thành các thành phần nhỏ hơn. Trong công nghệ phần mềm thì nó còn có nghĩa là phải hiểu rõ quá trình xây dựng đặc tả yêu cầu của người sử dụng, nắm được các chức năng và cách phân rã hệ thống vật lý thành các đơn thể (module). Theo phương pháp truyền thống thì điều đó thường được thực hiện

theo cách tiếp cận trên-xuống (top-down), sử dụng phương pháp phân tích có cấu trúc. Phân tích hướng đối tượng cho phép mô tả hệ thống gần với thế giới thực hơn, xác định rõ các đối tượng, trừu tượng hoá các yêu cầu đề trên cơ sở đó xây dựng được cấu trúc của hệ thống. Phương pháp hướng đối tượng giải quyết được hố ngăn cách giữa phân tích và thiết kế hệ thống.

Trong mục này chúng ta đề cập đến các bước cần thực hiện trong phân tích hướng đối tượng (PTHĐT). Thông qua ví dụ về phân tích hệ thống thư viện, chúng ta hình dung rõ hơn công việc xây dựng các đặc tả yêu cầu, mô tả đối tượng và cách xác định mối quan hệ giữa các lớp đối tượng trong hệ thống.

1.2. Các bước thực hiện trong phân tích hướng đối tượng

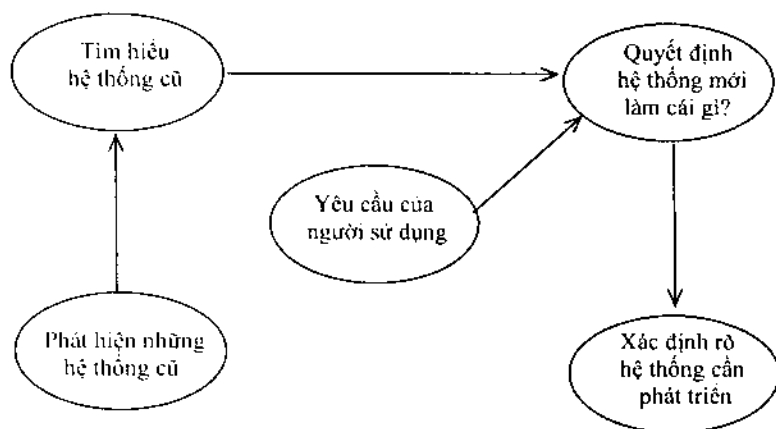
Để xây dựng một hệ thống phần mềm thì phải giải quyết ba vấn đề chính sau:

- + Dữ liệu, lớp các đối tượng và cấu trúc của chúng.
- + Những hành vi thể hiện các chức năng cục bộ, những quá trình trong hệ thống.
- + Điều khiển hành vi tổng thể của hệ thống.

Trong thực tế, cấu trúc dữ liệu và yêu cầu về hành vi của hệ thống thường xuyên thay đổi. Do vậy phải phân tích kỹ bài toán, lựa chọn phương pháp phát triển hệ thống thích hợp để cho hệ thống có tính chất mở, dễ thích nghi giúp cho công việc bảo trì hệ thống đỡ tốn kém.

Người phân tích hệ thống là người có kiến thức bao quát, có kinh nghiệm trong quá trình phân tích nhiều hệ thống ứng dụng khác nhau, đồng thời phải có khả năng giao tiếp, trao đổi và hiểu được những người đầu tư, thiết kế và những người sử dụng hệ thống.

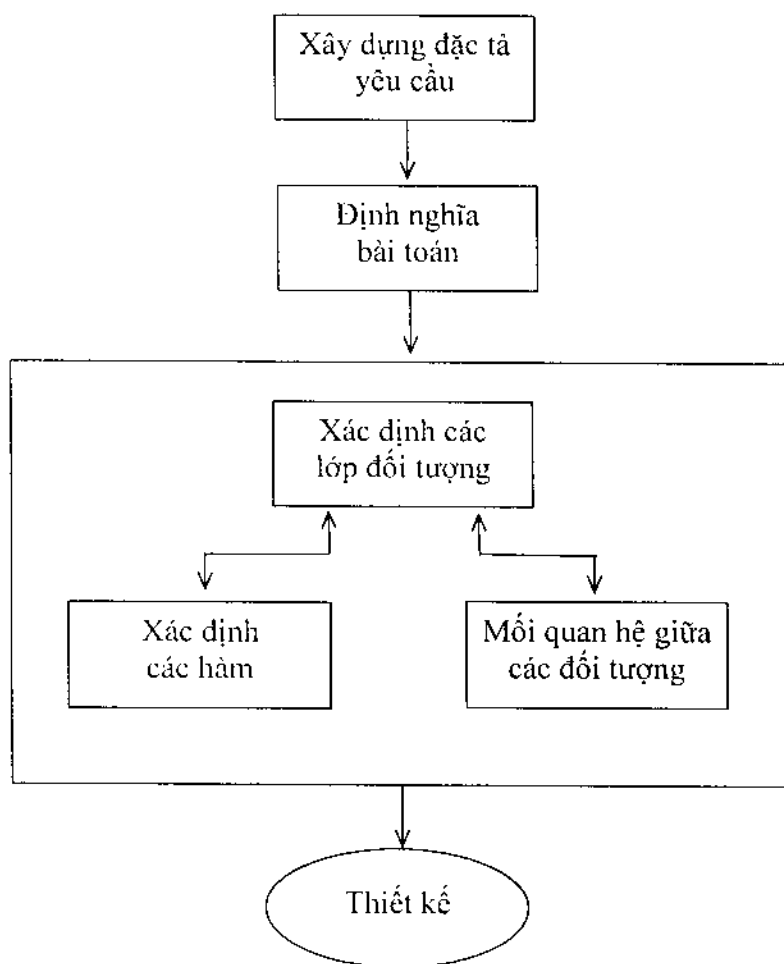
Nhiệm vụ của phân tích hệ thống là phải trả lời cho được câu hỏi "Hệ thống làm cái gì?" và "Tại sao?". Để xác định được bài toán và trả lời được những câu hỏi nêu trên thì người phân tích cũng cần phải phát hiện, tìm hiểu kỹ những hệ thống đã có hoặc đang hoạt động trong thực tế. Có thể đó chưa phải là hệ thống tin học hoá. Trên cơ sở nghiên cứu những hệ thống cũ, xác định rõ yêu cầu của người sử dụng để quyết định xem hệ thống cần xây dựng sẽ làm cái gì và hoạt động như thế nào. Quá trình đó được mô tả như ở hình 1-1.



Hình 1-1. Mức độ bao quát thể giới thực

Trong các phương pháp truyền thống thì mô hình dòng dữ liệu được mô tả thông qua sơ đồ dòng dữ liệu. Các quá trình trong hệ thống được xác định thông qua việc phân rã chức năng top-down. Sơ đồ biến đổi trạng thái được sử dụng để mô tả sự biến đổi thông tin và dòng điều khiển trong hệ thống. Phương pháp hướng đối tượng kết hợp hai phương diện dữ liệu với quá trình, gộp chung hành vi cục bộ với dữ liệu trong một đơn vị cấu trúc. Phương pháp phân tích hướng đối tượng cung cấp cho chúng ta công cụ đơn giản nhưng đủ mạnh để xác định các đối tượng và xây dựng các đơn nguyên của hệ thống cần phát triển. Phân tích hướng đối tượng bao gồm các bước sau:

- + Tìm hiểu bài toán.
- + Xác định rõ các đặc tả yêu cầu của người sử dụng, của hệ thống phần mềm.
- + Xác định các đối tượng và các thuộc tính của chúng.
- + Xác định các hàm mà các đối tượng sẽ phải thực hiện (hành vi của các đối tượng).
- + Xác định mối quan hệ tương tác giữa các đối tượng, các thông báo và sự truyền thông báo giữa các đối tượng.



Hình 1-2. Phân tích hướng đối tượng

1.2.1. Tìm hiểu kỹ bài toán

Nhiệm vụ đầu tiên của quá trình phân tích là phải tìm hiểu kỹ bài toán ứng dụng. Người phân tích phải gặp gỡ, trao đổi với những người đầu tư, những người sử dụng để biết rõ về chức năng, nhiệm vụ của hệ thống cần phát triển. Đồng thời người phân tích phải tìm hiểu, phát hiện những hệ thống cũ đã hoặc đang giải quyết những vấn đề tương tự như những vấn đề mà hệ thống cần xử lý. Dựa vào những kinh nghiệm, kết quả phân tích những hệ thống cũ, những công việc mà hàng ngày phải thực hiện để xác định chính xác bài toán. Trên cơ sở đó làm rõ hơn những yêu cầu của bài toán và định nghĩa lại theo quan điểm của các kỹ sư phần mềm để đảm bảo đưa ra được lời giải tin học (hệ thống thực hiện được trên máy tính).

Các khẳng định về bài toán phải đơn giản và rõ ràng, mạch lạc về văn phạm. Điều này giúp cho các kỹ sư phần mềm có điều kiện tập chung nhiều hơn vào việc xây dựng lời giải cho bài toán. Dựa trên những khẳng định của bài toán để xây dựng các đặc tả yêu cầu của người sử dụng lẫn của cả hệ thống phần mềm.

1.2.2. Xây dựng các đặc tả yêu cầu

Khi đã định nghĩa rõ bài toán thì bước tiếp theo là phải tìm hiểu xem hệ thống dự kiến sẽ yêu cầu làm cái gì? Điều quan trọng ở đây là phải xây dựng được danh sách các yêu cầu của người sử dụng. Rõ ràng là ở đây cần có sự trao đổi, hiểu biết giữa người sử dụng và người phát triển hệ thống về những điều mà họ mong muốn. Dựa trên những yêu cầu của người sử dụng, người phát triển đưa ra các đặc tả cho hệ thống. Người xây dựng hệ thống phải trả lời được các câu hỏi:

- + Đầu ra (output) của hệ thống là cái gì?
- + Hệ thống sẽ phải làm cái gì để có kết quả mong muốn, nghĩa là phải xử lý cái gì?
- + Đầu vào (input) của hệ thống là cái gì?
- + Những tài nguyên mà hệ thống yêu cầu là cái gì?

Phải hiểu rõ nguồn gốc, các dạng thông tin cần cung cấp cho hệ thống hoạt động. Hệ thống sẽ giải quyết vấn đề gì, những kết quả cần phải có là gì. Xác định được mối quan hệ giữa đầu vào/ra (input/output), nghĩa là xác định được những khẳng định về mối quan hệ giữa tiền điều kiện và hậu điều kiện cho các quá trình trong hệ thống.

Các đặc tả chi tiết phục vụ cho việc xây dựng và thử nghiệm hệ thống để kiểm tra xem những nhiệm vụ đặt ra có được hoàn thành hay không.

1.2.3. Xác định các đối tượng

Thông thường các đối tượng sẽ được xác định thông qua các thực thể trong thế giới thực và được trừu tượng hoá thành các đối tượng trừu tượng. Để xác định các đối tượng chúng ta có thể sử dụng một trong những công cụ sau:

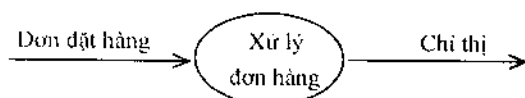
1. Sơ đồ dòng dữ liệu
2. Phân tích văn bản.

Sơ đồ dòng dữ liệu:

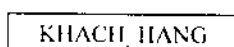
Sơ đồ dòng dữ liệu là mô hình hệ thống cho cả dữ liệu lẫn quá trình. Thông tin lấy từ các nguồn dữ liệu, được chuyển đến cho một hay nhiều quá

trình xử lý và ngược lại, một quá trình khi nhận đủ thông tin vào (input) thì bắt đầu thực hiện, xử lý thông tin và cho các kết quả (output) và chúng được gửi tới các kho dữ liệu. Trong sơ đồ dòng dữ liệu, một quá trình sẽ được thực hiện khi có đủ các thông tin đầu vào (theo các đường có mũi tên dẫn đến quá trình đó).

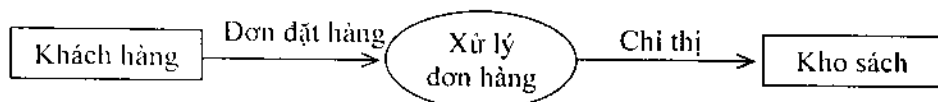
Trong sơ đồ, hình tròn hoặc ellipse được sử dụng để biểu diễn cho một quá trình, trong hình có tên gọi của quá trình. Tên gọi cho một quá trình phải là duy nhất và bao giờ cũng phải bắt đầu bằng động từ kết hợp với bổ ngữ như: "Xử lý đơn hàng", "Ghi nhận nguồn hàng" v.v... Ví dụ:



Chức năng quan trọng của quá trình là xử lý dữ liệu, biến đổi thông tin. Dòng dữ liệu được biểu diễn bằng đường thẳng có mũi tên làm nhiệm vụ chuyển tải thông tin vào hoặc ra khỏi một quá trình. Mũi tên chỉ hướng của dòng thông tin. Lưu ý là ở đây chỉ nói tới sự vận chuyển thông tin logic chứ không phải thông tin ở dạng vật lý. Dòng dữ liệu được gắn với một tên nhưng không nhất thiết phải là duy nhất. Các dòng dữ liệu, và tên được gắn cho nó phải chỉ ra được thông tin logic tương ứng cho một quá trình. Trong sơ đồ dòng dữ liệu, các dữ liệu được biểu diễn bằng hình chữ nhật có chứa tên của thông tin được cất giữ. Tên gắn với dữ liệu phải là danh từ. Ví dụ:



biểu diễn cho những thông tin về khách hàng được có tên là KHACH_HANG. Giữa dữ liệu và quá trình luôn có ít nhất một dòng dữ liệu liên kết.

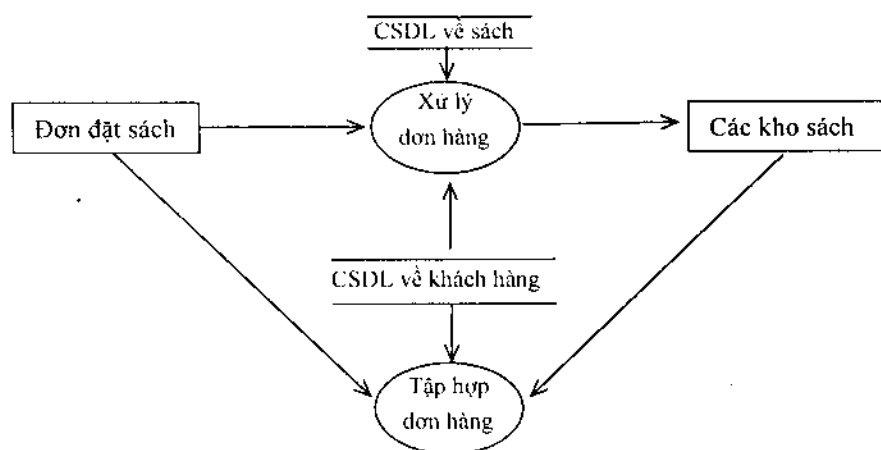


Hình 1-3. Sơ đồ dòng dữ liệu

Các quá trình được biểu diễn trong các ô hình tròn hoặc ellipse là các thủ tục, các hàm. Hình 1-3 mô tả sơ đồ dòng dữ liệu của hệ thống xử lý đơn hàng và vận chuyển thông tin cho công ty phát hành sách.

Trong sơ đồ dòng dữ liệu của hệ thống thì các thực thể được biểu diễn

trong các hình chữ nhật và các kho dữ liệu được biểu diễn với tên gọi đặt trong hai đường thẳng song song. Kho dữ liệu biểu diễn cho một lượng lớn thông tin cần phải lưu trữ trong một thời gian dài, thường là trong các tệp dữ liệu để cho nhiều người có thể truy nhập vào. Sơ đồ dòng dữ liệu có thể sử dụng để biểu diễn quá trình xử lý thông tin trong hệ thống ở nhiều mức độ trừu tượng khác nhau. Quá trình "Xử lý đơn hàng", "Tập hợp đơn hàng" ở hình 1-4 được làm mịn từ quá trình "Xử lý đơn hàng" ở hình 1-3 và có thể tiếp tục được làm mịn thêm, mô tả những quá trình như thanh toán, giao hàng v.v..., ở mức độ chi tiết hơn.



Hình 1-4. Sơ đồ dòng dữ liệu trong hệ xử lý đơn đặt hàng

Phương pháp tạo ra sơ đồ dòng dữ liệu

Chúng ta có thể tạo ra sơ đồ dòng dữ liệu theo một trong hai cách sau:

1. *Dùng sơ đồ chức năng:* Sơ đồ chức năng chỉ cho chúng ta biết về chức năng và cấu trúc phân cấp công việc cần thực hiện. Một trong những nhiệm vụ đầu tiên của người phân tích là phân tích bài toán để xây dựng sơ đồ chức năng của hệ thống. Theo phương pháp có cấu trúc, việc phân rã chức năng của hệ thống thành những chức năng con lại bao hàm nhiều chức năng con khác nữa sẽ cho kết quả là một sơ đồ phân cấp các chức năng của hệ thống (phân tích chức năng và cách xây dựng sơ đồ chức năng được đề cập kỹ trong cuốn "Phân tích, thiết kế và cài đặt hệ thống tin quản lý, Viện Tin học"). Các chức năng trong sơ đồ chức năng sẽ được chuyển tương ứng sang quá trình trong sơ đồ dòng dữ liệu. Dựa vào kết quả tìm hiểu, phân tích bài toán để xác định các nguồn dữ liệu, kho dữ liệu vào/ra cho các quá trình trong sơ đồ dòng dữ liệu.

2. *Sử dụng sơ đồ ngữ cảnh*: Sơ đồ ngữ cảnh thường được sử dụng ở giai đoạn đầu của quá trình phân tích và được dùng để vạch phạm vi hoạt động của hệ thống. Thông thường sơ đồ ngữ cảnh được xây dựng dưới dạng tựa như sơ đồ chức năng, bao gồm một nút chính biểu diễn cho nhiệm vụ trung tâm của hệ thống, và tỏa ra là các tác nhân ngoài hoặc nhóm công việc có liên quan.

Phân tích sơ đồ chức năng, sơ đồ ngữ cảnh và cách xây dựng sơ đồ dòng dữ liệu có thể tham khảo trong cuốn "Phân tích, thiết kế và cài đặt hệ thống tin quản lý, Viện Tin học".

Chúng ta có thể dựa vào định nghĩa của sơ đồ dòng dữ liệu để xác định các đối tượng. Trong sơ đồ dòng dữ liệu, những ô hình chữ nhật, ô có hai đường thẳng song song biểu diễn cho dữ liệu, kho dữ liệu có thể được xem như là các đối tượng. Lưu ý rằng không có sự tương ứng 1-1 giữa những nút biểu diễn cho dữ liệu, kho dữ liệu trong sơ đồ dòng dữ liệu với các đối tượng. Một đối tượng có thể là đại diện của một hay nhiều nút dữ liệu, kho dữ liệu trong sơ đồ dòng dữ liệu tùy thuộc vào ngữ cảnh của vấn đề mà nó mô tả. Ví dụ trong hình 1-4, chúng ta sẽ có ba đối tượng: SACH, DON_HANG và KHACH_HANG. Hai nút: kho dữ liệu "CSDL về sách" với nút dữ liệu "Các kho sách" cùng đại diện cho đối tượng SACH vì cùng quản lý những thông tin về sách; đối tượng DON_HANG được xác định từ nút "Đơn đặt sách" còn KHACH_HANG được xác định từ nút "CSDL về khách hàng".

Phân tích văn bản:

Cách thực hiện thứ hai là dựa trên mô tả bằng văn bản của bài toán hoặc lời giải để phân tích. Văn bản mô tả có thể gồm có một hay nhiều câu, một hay nhiều đoạn, chương, phần, tùy thuộc vào mức độ phức tạp của bài toán. Trong đó các đối tượng thường được mô tả bằng các danh từ. Danh từ thường được phân loại thành danh từ riêng, danh từ chung, và các danh từ trừu tượng hoặc danh từ chỉ đại lượng.

Điều quan trọng cần lưu ý khi phân tích là phải dựa vào ngữ nghĩa và ngữ cảnh để phân loại danh từ. Một từ có thể là danh từ chung trong ngữ cảnh này song nó cũng có thể là danh từ trừu tượng hoặc danh từ chỉ đại lượng trong ngữ cảnh khác. Cũng cần lưu ý là không phải tất cả các danh từ đều được dùng để biểu diễn cho những đối tượng cần thiết cho hệ thống của chúng ta.

Bảng 1-1. Bảng phân loại danh từ

Kiểu của danh từ	Ý nghĩa	Ví dụ
Danh từ chung	Xác định một lớp các thực thể	Ô tô, khách hàng, học sinh
Danh từ riêng	Tên của một đối tượng xác định	Nguyễn An, IBM, BBC
Danh từ trừu tượng hoặc đại lượng	Xác định chất lượng, đại lượng hoặc hoạt động ứng với danh từ	Thu nhập, lương, giao thông

Tóm lại, chúng ta có thể sử dụng một trong hai công cụ trên để xác định danh sách các đối tượng của bài toán ứng dụng và sau đó tiếp tục:

1. Xác định những đối tượng chỉ nằm trong không gian bài toán, không gian lời giải, và những đối tượng nằm trong không gian bài toán nhưng nằm ngoài giới hạn của hệ thống phần mềm.

2. Xây dựng các thuộc tính cho các đối tượng của không gian lời giải.

Sau khi đã xác định được các đối tượng thì nhiệm vụ tiếp theo là xác định những thuộc tính mô tả các tính chất của từng lớp đối tượng. Người phân tích có thể dựa vào ba nguồn cung cấp thông tin cơ bản sau để tập hợp, xây dựng những thuộc tính cho từng lớp đối tượng:

1. Từ những kinh nghiệm, tri thức của người phân tích hệ thống về thực tế công việc trong lĩnh vực tập trung nghiên cứu để dự đoán, xác định danh sách các thuộc tính.

2. Từ những người sử dụng, thông qua các cuộc phỏng vấn, trao đổi và tìm hiểu bài toán cụ thể để lập danh sách các thuộc tính.

3. Từ những hệ thống cũ, những bảng biểu, báo cáo và các tài liệu khoa học được sử dụng thường xuyên trong lĩnh vực đang nghiên cứu để chọn lọc ra những thuộc tính cho lớp các đối tượng đã xác định.

Theo cách thức đó chúng ta có thể đề xuất danh sách những thuộc tính cho các lớp SACH, DON_HANG và KHACH_HANG trong hệ quản lý kinh doanh sách đã nêu ở trên như sau:

Đối với lớp SACH

Tac_gia : Tên tác giả của cuốn sách

Ten_sach : Tên gọi, tiêu đề của cuốn sách

Nha_XB : Nhà xuất bản

Nam_XB : Năm xuất bản

Đối với lớp DON_HANG

So_hieu : Số hiệu đơn đặt hàng

SH_KH : Số hiệu hoặc tên khách hàng

Ngay_DH : Ngày đặt hàng

Ngay_GH : Ngày giao hàng

Đối với lớp KHACH_HANG

SH_KH : Số hiệu khách hàng

Ten_KH : Tên khách hàng

Dia_chi : Địa chỉ, nơi giao hàng

TK_KH : Số tài khoản của khách hàng trong ngân hàng

Danh sách các thuộc tính của các lớp sẽ được tiếp tục xem xét, bổ sung cho đầy đủ trong giai đoạn thiết kế. Cần lưu ý là phải cân nhắc để đưa ra được những thuộc tính chung nhất, với những tên gọi đặc trưng cho từng lớp đối tượng.

1.2.4. Xác định các hàm

Để mô tả đầy đủ, chính xác các đối tượng chúng ta cần tiếp tục xác định các hàm mô tả hành vi của chúng. Chúng ta có thể dựa vào văn bản mô tả bài toán để xác định các hàm. Thông thường, trong các câu mô tả thì động từ được dùng để chỉ một hành động, sự xuất hiện, phân loại hay cấu thành của các đối tượng.

Bảng 1-2. Bảng phân loại động từ

Các kiểu động từ	Ý nghĩa	Ví dụ
Động từ chỉ hành động	Nêu các hành động	Đọc, viết, mua, bán
Động từ chỉ sự xuất hiện	Phân loại	Là, nằm trong v.v...
Động từ chỉ sở hữu	Cấu thành	Có, là một phần của
Động từ chỉ sự so sánh	Các phép so sánh	Nhỏ hơn, bằng v.v...
Động từ chỉ trạng thái	Điều kiện - bất biến	Cần, phải có mặt

Các động từ chỉ hành động và so sánh giúp cho chúng ta xác định được các hàm, còn động từ chỉ sự xuất hiện, so sánh giúp chúng ta xây dựng được cấu trúc phân loại. Động từ sở hữu giúp cho việc xác định những cấu trúc cấu thành của các đối tượng. Cách thứ hai là dựa vào sơ đồ dòng dữ liệu để

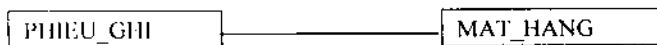
xác định các hàm, các chức năng được biểu diễn bằng các hình tròn hoặc ellipse. Ví dụ, để mô tả cho hành vi của đối tượng trong lớp KHACH_HANG chúng ta phải xây dựng các hàm xử lý những thuộc tính đã xác định ở trên như các hàm xác định những thông tin về khách hàng: số hiệu, họ và tên, địa chỉ, tài khoản v.v...

1.2.5. Xác định mối quan hệ giữa các đối tượng

Bước tiếp theo là xác định mối quan hệ giữa các đối tượng, nghĩa là sự trao đổi thông tin giữa chúng. Như chúng ta thấy, trong một hệ thống mỗi thực thể phải có quan hệ ít nhất với một thực thể khác. Chẳng hạn, trong hệ thống quản lý kinh doanh của công ty phát hành sách với sơ đồ dòng dữ liệu đã xây dựng ở hình 1-4, khách hàng muốn mua sách thì phải ghi vào đơn đặt hàng, nghĩa là đối tượng KHACH_HANG sẽ phải gửi một thông báo (đơn đặt hàng) cho đối tượng DON_HANG. Tương tự, DON_HANG lại có quan hệ với SACH vì những cuốn sách sẽ được bán cho khách hàng khi nhận được các đơn đặt hàng. Quan hệ giữa các lớp đối tượng có thể có những kiểu khác nhau và được phân thành ba kiểu sau:

1. Quan hệ một - một
2. Quan hệ một - nhiều
3. Quan hệ nhiều - nhiều

Quan hệ một - một: Hai lớp có quan hệ 1-1 nếu với mỗi đối tượng của lớp này có liên quan tương ứng một đối tượng ở lớp kia và ngược lại. Ví dụ: Hai lớp PHIEU_GHI và MAT_HANG có quan hệ 1-1. Mỗi phiếu ghi trong lớp PHIEU_GHI sẽ mô tả đúng một mặt hàng được quản lý trong lớp MAT_HANG. Quan hệ này được biểu diễn như sau:



Hình 1-5. Quan hệ một - một

Quan hệ một - nhiều: Hai lớp A và B có quan hệ một - nhiều nếu:

- Với mỗi đối tượng trong lớp A có quan hệ với một hay nhiều đối tượng trong lớp B.
- Mỗi đối tượng trong lớp B có quan hệ với một đối tượng của lớp A.

Quan hệ một - nhiều được biểu diễn như sau:



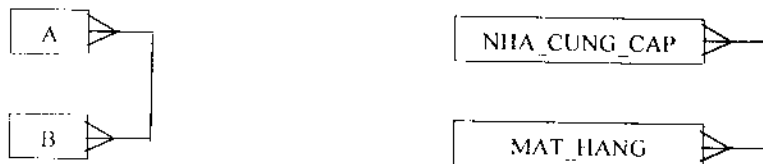
Hình 1-6. Quan hệ một - nhiều

Lớp KIIACH_HANG có quan hệ một - nhiều với lớp DON_HANG vì một khách hàng có thể đặt nhiều đơn hàng khác nhau.

Quan hệ nhiều - nhiều: Hai lớp A và B có quan hệ nhiều - nhiều nếu:

- Mỗi đối tượng A có sự tương ứng với nhiều đối tượng trong B.
- Ngược lại, mỗi đối tượng trong B cũng có tương ứng với nhiều đối tượng trong A.

Quan hệ nhiều - nhiều được biểu diễn như sau:



Hình 1-7. Quan hệ nhiều - nhiều

Hai lớp NHA_CUNG_CAP và MAT_HANG có quan hệ nhiều - nhiều vì mỗi xí nghiệp có thể sản xuất và bán ra nhiều mặt hàng và ngược lại, một mặt hàng cũng có thể được sản xuất ở nhiều nơi.

Mô hình dữ liệu và những quan hệ giữa các đối tượng được sử dụng không chỉ như một công cụ để phân tích, thiết kế mà còn như một phương pháp kiểm chứng các yêu cầu của hệ thống. Những vấn đề về xây dựng mô hình dữ liệu, mô hình quan hệ và các bước chuẩn hoá dữ liệu có thể tham khảo trong cuốn “Phân tích, thiết kế và cài đặt hệ thống tin quản lý, Viện Tin học”.

1.3. Ví dụ

1.3.1. Phát biểu bài toán

Tại khoa Tin học của một trường đại học có khoảng vài trăm cuốn sách để cho các cán bộ nhân viên trong khoa mượn. Hãy xây dựng một hệ thống để quản lý trên máy tính những cuốn sách mà khoa có, những cuốn nào đang trong phòng làm việc, những cuốn nào đang có người mượn và ai mượn.

Đây là ví dụ đơn giản, nhưng cũng khá điển hình trong quá trình phân tích, thiết kế hướng đối tượng. Chúng ta cũng sẽ phân tích khả năng mở rộng, khả năng bảo trì hệ thống hướng đối tượng theo những yêu cầu mới cần mở rộng và phát triển hệ thống nhằm đáp ứng nhu cầu của người sử dụng.

Để giải quyết bài toán đã nêu ở trên, chúng ta có thể sử dụng một trong những hệ quản trị dữ liệu phổ dụng như FoxPro, Access v.v... Ở đây chúng

ta muốn thông qua ví dụ này làm rõ hơn những công việc, các bước cần thực hiện trong quá trình phân tích hướng đối tượng.

1.3.2. Phân tích hệ thống

Bài toán nêu ở trên tương đối rõ ràng. Yêu cầu xây dựng hệ thống phần mềm để quản lý các cuốn sách. Phân tích hướng đối tượng là việc lặp lại nhiều lần việc phân tích bài toán để xác định các đối tượng và xây dựng các đặc tả bài toán. Nhưng phân tích hướng đối tượng còn có yếu tố tổng hợp. Việc thực hiện trừu tượng hoá những yêu cầu của người sử dụng và xác định rõ được các đối tượng chính cho phép tập hợp chúng để tạo ra cấu trúc hệ thống logic hỗ trợ cho giai đoạn thiết kế tiếp theo.

Nhiệm vụ chính của giai đoạn phân tích là xây dựng mô hình khái niệm cho thế giới thực. Thế giới thực của chúng ta ở đây gồm những cuốn sách và bạn đọc. Những cuốn sách sẽ được để ở đâu đó, trong phòng làm việc hoặc đã cho ai mượn.

Để hiểu rõ hơn về các thực thể và mối quan hệ của chúng trong thế giới thực mà bài toán đặt ra ở đây là sách và bạn đọc, chúng ta cần tìm hiểu kỹ về hệ thống có liên quan như hệ thống thư viện. Trên cơ sở đó, xây dựng các đặc tả yêu cầu cho bài toán.

Phân tích kỹ bài toán, dựa vào văn bản mô tả bài toán chúng ta thấy có hai lớp đối tượng là: SACH và BAN_DOC. Trong các hệ thống thư viện, một cuốn sách có thể được xác định thông các thuộc tính như: mã số thư viện, tên tác giả, tên gọi cuốn sách, nhà xuất bản, năm xuất bản v.v... Để đơn giản chúng ta có thể dùng tên tác giả để xác định cuốn sách, hoặc tên gọi cùng tên tác giả nếu như có hai cuốn cùng tác giả, còn bạn đọc thì sẽ được xác định thông qua họ và tên của từng người. Ví dụ: Peter Norton là cuốn sách “Cẩm nang lập trình” do Peter Norton viết, là một đối tượng trong lớp SACH. Lan Anh là tên một độc giả, là một đối tượng trong lớp BAN_DOC v.v...

Hệ thống phần mềm mà chúng ta xây dựng sẽ phải giải quyết các vấn đề sau:

- + Lan Anh đã mượn cuốn Peter Norton.
- + Hoang Trung đã mượn những cuốn sách nào?
- + Ai mượn cuốn sách Peter Henderson?
- + Lan Anh trả cuốn Peter Norton.
- + Một cuốn sách mới được bổ sung.

Đây chính là danh sách các yêu cầu của hệ thống.

Sau khi đã xác định được các yêu cầu của bài toán và lớp các đối tượng, chúng ta thực hiện bước tiếp theo là xác định các thuộc tính, hàm và mối quan hệ giữa các lớp đối tượng.

Sách là đối tượng đã được xác định và được biểu diễn như sau:

SACH
Peter Norton
...

Trong đó, SACH là tên gọi lớp tất cả các cuốn sách có trong thư viện. Peter Norton là tên một cuốn sách, một đối tượng cụ thể. Tương tự đối tượng bạn đọc sẽ được mô tả như sau:

BAN_DOC
Lan Anh
...

Trong đó, BAN_DOC là lớp các độc giả và Lan Anh là một bạn đọc, một đối tượng trong lớp đó.

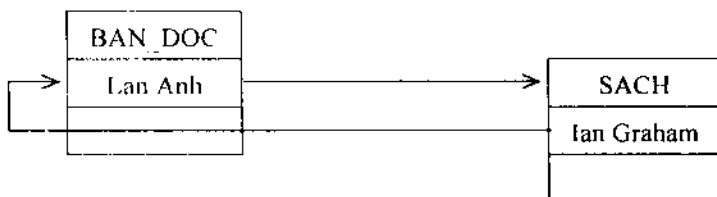
Trong hệ thống thư viện, những cuốn sách và độc giả sẽ được mô tả tương ứng là các đối tượng SACH và BAN_DOC. Bằng nhiều cách khác nhau, chúng ta phân tích và xác định được các thuộc tính, các hàm cho hai lớp SACH, BAN_DOC.

```
class SACH
{
    Attribute //Thuộc tính
        Tac_gia   : Tác giả cuốn sách,
        Ten_sach  : Tên gọi hoặc tiêu đề của cuốn sách
        Xuat_ban  : Nhà, năm xuất bản
        Noi_giu   : Sách đã cho ai mượn hay có tại thư viện
    Function //Hàm
        Nhap_sach()      : Nhập các thông tin về cuốn sách vào
        thư viện
        Cho_muon()       : Xác định là sách đã cho mượn
        Hoan_tra()        : Sách đã được trả lại thư viện
        Display()        : Hiện các thông tin về cuốn sách
}
class BAN_DOC
{
    Attribute //Thuộc tính
        Ho_ten      : Họ và tên người mượn sách,
        Dia_chi     : Địa chỉ, điện thoại của bạn đọc
        Ten_sach     : Tên những cuốn sách đã mượn
    Function //Hàm
```

Nhan HT() : Nhập họ tên, địa chỉ của một bạn đọc
 Muon() : Nhập thêm những cuốn sách mới mượn
 Tra() : Trả sách cho thư viện
 Display() : Cho biết những thông tin về bạn đọc

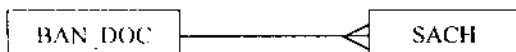
}

Bây giờ chúng ta cần xác định mối quan hệ giữa hai lớp SACH và BAN_DOC. Ví dụ, Lan Anh mượn cuốn Peter Norton có thể được mô tả như đồ thị sau:



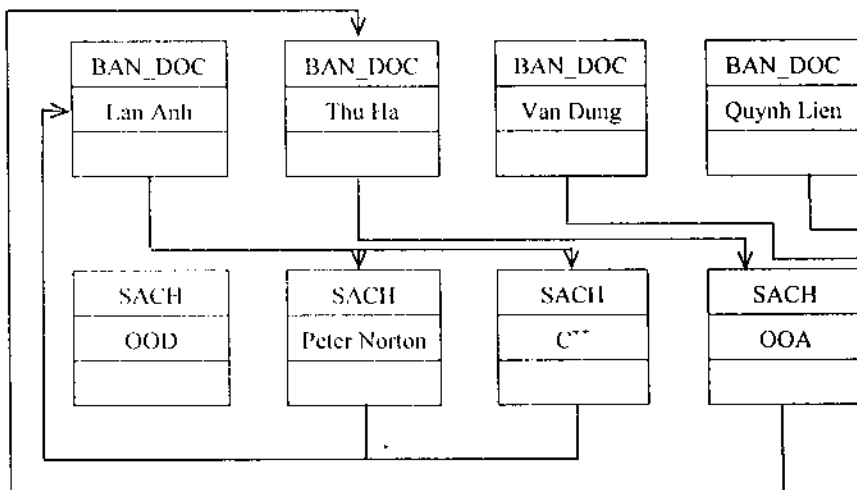
Hình 1-8. Lan Anh mượn cuốn sách Peter Norton

Một cuốn sách có thể cho nhiều nhất một người mượn, ngược lại một người có thể mượn nhiều cuốn sách. Do vậy BAN_DOC và SACH có quan hệ một - nhiều.



Hình 1-9. Mối quan hệ giữa độc giả và sách

Trong mô hình đồ thị của chúng ta, mỗi quan hệ giữa hai lớp đối tượng được thể hiện chi tiết hơn như hình 1-10.



Hình 1-10. Thể hiện mối quan hệ giữa hai lớp đối tượng SACH và BAN_DOC

Sơ đồ trên mô tả sự trao đổi thông tin giữa các lớp đối tượng. Lan Anh mượn hai cuốn Peter Norton và C⁺⁺; Hoàng Trung mượn cuốn OOA, cuốn sách OOD vẫn chưa có ai mượn v.v...

Hệ thống của chúng ta luôn hoạt động bởi vì:

- + Thường xuyên có người mượn sách.
- + Một cuốn sách được trả lại hoặc được mua bổ sung.

Dựa vào kết quả phân tích ở trên chúng ta dễ dàng xây dựng thiết kế và cài đặt hệ thống quản lý sách đơn giản nhưng đáp ứng được yêu cầu đặt ra là quản lý được sách và dễ dàng sửa đổi, bổ sung khi cần thiết.

§2. THIẾT KẾ HƯỚNG ĐỐI TƯỢNG

2.1. Giới thiệu chung

Mục này sẽ mô tả phương pháp thiết kế phần mềm dựa trên các đối tượng. Phương pháp hướng đối tượng (HDT) nhằm che giấu thông tin ở mức tối đa và vì vậy hỗ trợ cho việc thiết kế những hệ thống với những cấp bộ giữa các thành phần là cực tiểu nhưng mức độ cô kết hệ thống lại cao hơn cách tiếp cận chức năng. Chúng ta sẽ tập trung nghiên cứu các bước cần thực hiện trong thiết kế hướng đối tượng và ví dụ mô tả cách thiết kế các lớp, xây dựng cấu trúc hệ thống trong quá trình phát triển phần mềm.

Che giấu thông tin là chiến thuật thiết kế sao cho có thể giấu được nhiều nhất lượng thông tin ở bên trong các thành phần cơ sở của một thiết kế. Điều này có nghĩa là sự trao đổi giữa các thực thể của thiết kế là cực tiểu và vì vậy thiết kế dễ dàng thay đổi hơn. Thiết kế hướng đối tượng (TKHDT) là phương pháp thiết kế được thực hiện theo nguyên lý che giấu thông tin. Khác với cách tiếp cận truyền thống (hướng chức năng) là nó xem hệ thống phần mềm (HTPM) là tập hợp các đối tượng tương tác với nhau. Mỗi đối tượng làm việc với trạng thái (dữ liệu) riêng của mình. Đối tượng, khái niệm cơ sở đã được đề cập nhiều ở các phần trước là một thực thể có tập các thuộc tính và tập các hàm tác động trên các thuộc tính đó.

Tập giá trị các thuộc tính xác định trạng thái của một đối tượng. Một đối tượng không được quyền truy nhập trực tiếp hoặc làm thay đổi trạng thái của đối tượng khác. Điều này dẫn đến là các đối tượng chỉ có thể trao đổi với nhau bằng các thông báo. Thông báo sẽ kích hoạt các hàm của đối tượng nhận thông tin tương ứng. Hoạt động của cơ chế truyền thông báo giữa các đối tượng là dị bộ (không đồng bộ) vì vậy chương trình được thiết kế theo hướng đối tượng có thể được thực hiện song song hoặc tuần tự tùy theo

phương pháp lập trình và những công cụ mà chúng ta thực hiện cài đặt có cho phép thực hiện song song hay không.

Thiết kế hướng đối tượng là phương pháp thiết kế hệ thống phần mềm không phụ thuộc vào ngôn ngữ lập trình. Nhiều đặc tính như “che dấu”, “kế thừa” làm cho việc thực hiện thiết kế trở nên dễ dàng hơn, đơn giản hơn. Những thiết kế này cũng có thể được cài đặt bằng ngôn ngữ chưa có đặc tính đó như Turbo C, hoặc Pascal, nhưng tốt nhất là nên sử dụng những ngôn ngữ hướng đối tượng để cài đặt những thiết kế được thực hiện theo cách tiếp cận hướng đối. Nhiều ngôn ngữ lập trình hướng đối tượng như Eiffel, Object Pascal, Smalltalk, C++ có những đặc tính hướng đối tượng hỗ trợ cho việc mô tả và thực hiện cài đặt trực tiếp những thiết kế hướng đối tượng hiệu quả hơn.

Tóm lại thiết kế hướng đối tượng có những ưu điểm chính sau:

- Loại bỏ được những miền dữ liệu dùng chung thông qua cơ chế trao đổi thông tin giữa các đối tượng bằng các thông báo.

- Các đối tượng được thiết kế là các thực thể độc lập (theo nghĩa không sử dụng dữ liệu chung), mọi thay đổi về trạng thái, bổ sung, sửa đổi các hoạt động chức năng của một đối tượng chỉ xảy ra bên trong của đối tượng đó, không ảnh hưởng đến các đối tượng khác. Mọi sự thay đổi trong thiết kế, trong hệ thống phần mềm chỉ xảy ra cục bộ đối với một số đối tượng liên quan. Điều này đảm bảo hệ thống có tính dễ mở rộng và dễ thích nghi, đáp ứng được nhiều tính chất quan trọng của sản phẩm phần mềm.

Các đối tượng có thể được tổ chức phân tán hoặc song song hay tuần tự theo yêu cầu của bài toán ứng dụng và khả năng kỹ thuật thực tế của dự án phát triển tin học ứng dụng.

2.2. Các bước thực hiện trong thiết kế hướng đối tượng

Nhiệm vụ của thiết kế hướng đối tượng là xác định các đối tượng trong không gian bài toán, chuyển chúng sang không gian lời giải, xây dựng mô hình kiến trúc và mô hình tính toán cho hệ thống phần mềm. Để xây dựng kiến trúc tổng thể cho hệ thống chúng ta sử dụng cách tiếp cận dưới - lên (bottom - up). Điều quan trọng là phải tạo ra được cấu trúc phân cấp, xác định được các lớp đối tượng trừu tượng và giảm thiểu được sự trao đổi giữa các đối tượng. Ở đây chúng ta cũng đề cập đến khả năng sử dụng lại trong thiết kế, phân loại các đối tượng thành những hệ thống con trong cấu trúc phân cấp.

Cách tiếp cận TKHĐT gồm các bước sau:

1. Xác định các lớp và các đối tượng, các thành phần cơ bản của lời giải.
2. Xây dựng các đặc tả cho các đối tượng, các lớp và mối quan hệ giữa chúng.
3. Xây dựng cấu trúc phân cấp cho các lớp.
4. Thiết kế các lớp.
5. Thiết kế các hàm thành phần của lớp.
6. Thiết kế chương trình chính.

Xác định các đối tượng trong không gian lời giải

Khi phân tích văn bản mô tả bài toán và các yêu cầu của người sử dụng, chúng ta xác định được các thực thể, những đối tượng trong không gian bài toán. Bước tiếp theo là phân tích kỹ các đối tượng, xác định các thuộc tính và các hàm đặc tả cho từng đối tượng. Đồng thời xác định thêm những đối tượng mới xuất hiện trong không gian lời giải. Khi xây dựng các đặc tả cho đối tượng, chúng ta phải xác định được các thuộc tính, dữ liệu mô tả trạng thái của đối tượng và các hàm mô tả hành vi của đối tượng. Thuộc tính là miền dữ liệu riêng của lớp đối tượng, là dữ liệu cục bộ trong một lớp. Thực hiện nguyên lý che giấu thông tin, trong một lớp dữ liệu có thể tổ chức thành hai vùng: vùng sở hữu riêng, chỉ dành riêng cho những đối tượng trong cùng lớp và vùng dùng chung, cho phép những đối tượng trong các lớp có quan hệ với nhau được quyền sử dụng. Các hàm (nhiều sách còn gọi là thủ tục, dịch vụ, phương thức) có thể dùng chung cho một số đối tượng. Quá trình xác định các hàm mô tả đối tượng (còn được gọi là hàm thành phần của lớp) được thực hiện như sau:

1. Nếu một hàm chỉ cần thiết cho một đối tượng thì hàm này chỉ hoạt động trong đối tượng yêu cầu.
2. Nếu có hai hoặc nhiều hơn đối tượng cần yêu cầu về một hàm thì cần phải xác định vùng hoạt động riêng của hàm trong các đối tượng đó.
3. Nếu có một hàm cần nhiều hơn một kiểu đối tượng (liên quan đến hai hoặc nhiều hơn các lớp đối tượng) thì hàm đó không phải là một hàm cố kết, do vậy cần phải phân tách dịch vụ đó ra thành các hàm mịn hơn.

Bằng cách đó chúng ta xây dựng được danh sách các hàm mô tả hành vi của các đối tượng. Đồng thời chúng ta cũng loại bỏ được những dư thừa, những thành phần phụ không cần thiết trong cấu trúc và trong các đối tượng.

Sự phụ thuộc giữa các lớp

Mục tiêu của thiết kế là xây dựng cấu trúc phân cấp cho hệ thống. Do vậy,

nhiệm vụ tiếp theo của chúng ta là xác định mối quan hệ giữa các lớp đối tượng cấu thành hệ thống. Lớp là tập hợp các đối tượng có chung một số thuộc tính, một số hàm vừa đủ để phân biệt với những lớp khác. Đối tượng là thể hiện của lớp. Trong thiết kế, khái niệm lớp đối tượng và đối tượng là hầu như không phân biệt, các lớp biểu diễn cho các đối tượng trong không gian lời giải. Để xây dựng được mô hình kiến trúc cho hệ thống phần mềm, chúng ta cần phân biệt ba loại quan hệ quan trọng giữa các lớp:

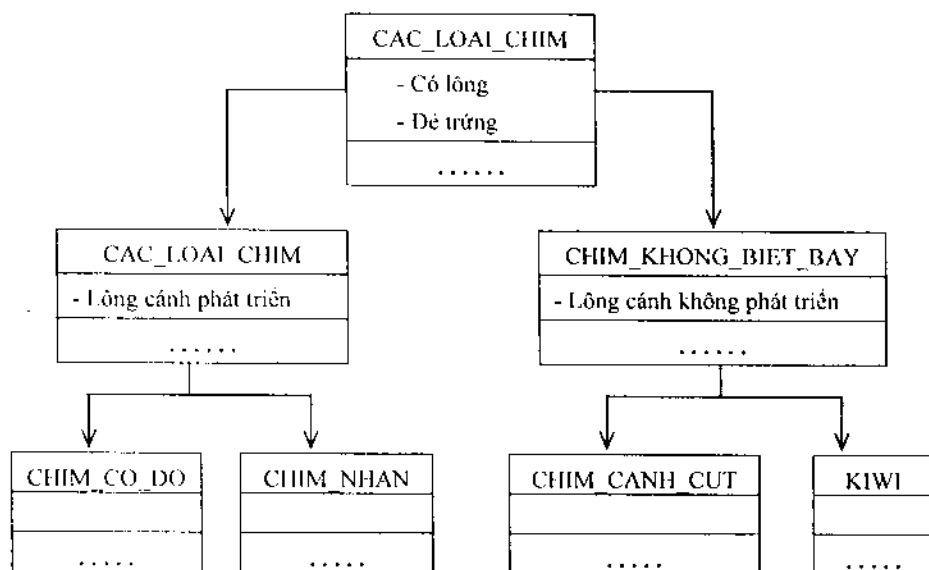
- Quan hệ kế thừa
- Quan hệ thành phần
- Quan hệ về sử dụng

Quan hệ kế thừa: Trong thực tế, có nhiều lớp có những thuộc tính, hàm giống nhau không những chỉ trong cùng một hệ thống mà có thể ở nhiều hệ thống khác nhau. Một trong những mục tiêu quan trọng của phương pháp hướng đối tượng là xây dựng các lớp đối tượng có khả năng sử dụng cho nhiều ứng dụng khác nhau trên cơ sở khai thác triệt để nguyên lý kế thừa. Quan hệ kế thừa giữa các lớp là sự giống nhau trong các lớp đối tượng và khả năng sử dụng một số đặc tính kế thừa từ những lớp trước. Một lớp có thể sử dụng lại một số thuộc tính, hàm của một hay nhiều lớp đã được định nghĩa trước. Lớp được định nghĩa trước có những tính chất chung để cho những lớp khác có thể kế thừa được gọi là lớp cơ sở và lớp kế thừa lớp cơ sở được gọi là lớp dẫn xuất (hoặc là lớp con).

Lớp dẫn xuất kế thừa một số hoặc tất cả các đặc tính của một hay nhiều lớp cơ sở. Một lớp có thể kế thừa các tính chất của nhiều lớp ở nhiều mức khác nhau và được bổ sung thêm một số đặc tính riêng. Có năm loại kế thừa: kế thừa đơn, kế thừa bội, kế thừa đa mức, kế thừa phân cấp, kế thừa phức hợp. Trong quan hệ kế thừa, chỉ những thuộc tính, hàm được khai báo sử dụng chung mới được quyền kế thừa.

Ví dụ: Trong hệ thống quản lý các loài chim, lớp cơ sở đầu tiên chúng ta có thể xây dựng là lớp CAC_LOAI_CHIM có thuộc tính, chức năng chung nhất như có lông, đẻ trứng. Trong số các loài chim thì chúng ta có thể phân làm hai loại: loại chim không bay được và loại chim biết bay. Hai lớp CHIM_BIET_BAY và CHIM_KHONG_BIET_BAY kế thừa từ lớp CAC_LOAI_CHIM nghĩa là các đặc tính: có lông và đẻ trứng không cần phải mô tả trong các lớp đó nữa mà chỉ cần bổ sung những đặc tính mô tả thêm về khả năng biết bay hoặc không biết bay của các loài chim. Tiếp tục phân tích lớp CHIM_KHONG_BIET_BAY, giả sử gồm hai lớp

CANH_CUT và KIWI còn lớp CHIM_BIET_BAY gồm các lớp CHIM_CO_DO, CHIM_NHAN v.v... Trên cơ sở xác định quan hệ kế thừa các loài chim chúng ta có cấu trúc như trong hình 2-1.



Hình 2-1. Quan hệ kế thừa

Quan hệ thành phần: Đối tượng của lớp này cũng là phần tử của lớp khác.

Ví dụ: Trong hệ thống quản lý cán bộ khoa học của một cơ quan thì một cán bộ nữ trẻ sẽ là thể hiện của LOP_CAN_BO_TRE và cũng là thành phần của lớp CAN_BO_NU.

Quan hệ về sử dụng: Khả năng sử dụng của một lớp để đọc, xử lý các đối tượng của những lớp khác.

Ví dụ: Một lớp A có thể sử dụng các lớp B và C theo các cách như sau:

- A đọc các phần tử của B
- A gọi tới các phần tử của C
- A tạo ra B bằng các sử dụng toán tử **new**

Mỗi quan hệ của các lớp đóng vai trò quan trọng trong thiết kế chương trình sau này.

Tổ chức phân cấp các lớp (theo nguyên lý tổng quát hoá)

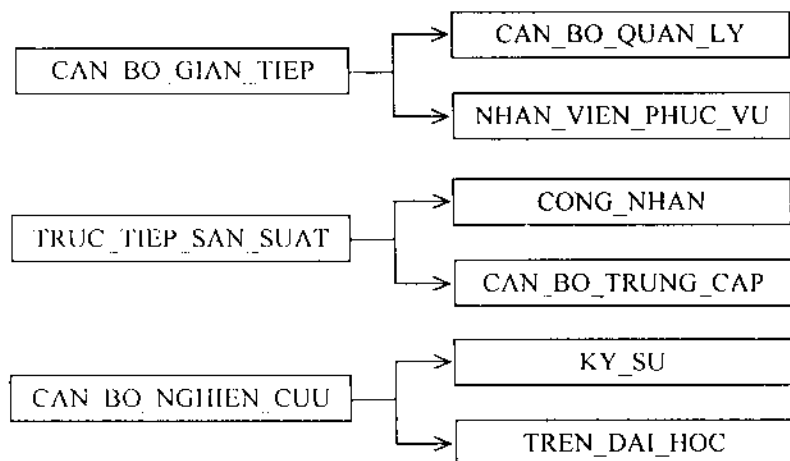
Ở trên chúng ta đã nghiên cứu mỗi quan hệ mà chủ yếu là quan hệ kế thừa của các lớp đối tượng. Ở đây chúng ta dựa vào những mối quan hệ đó

để xây dựng cấu trúc phân cấp trên nguyên tắc sử dụng lại tối đa các thuộc tính và hàm của những lớp đã được thiết kế trước.

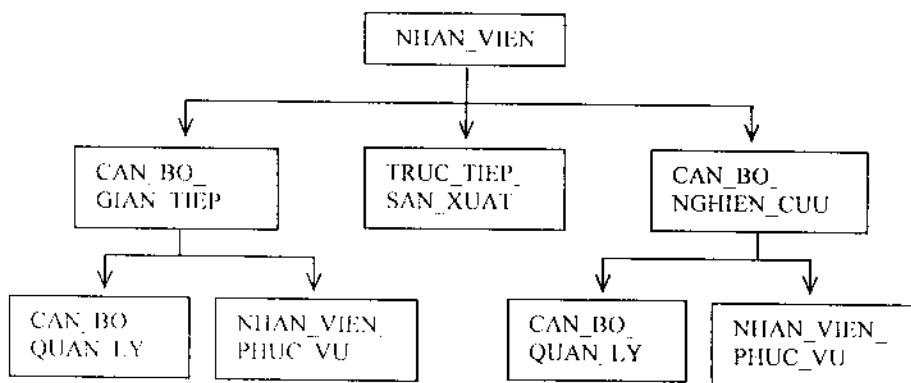
Tổ chức phân cấp các lớp là tập trung phân tích từng nhóm lớp có liên quan để xác định được những thuộc tính, hàm chung nhất của cả nhóm và sau đó kết hợp chúng lại để tạo ra lớp mới. Lớp mới được gọi là lớp trừu tượng và cũng là lớp cơ sở để cho các lớp trong cùng nhóm kế thừa. Lớp trừu tượng có thể có hoặc không có thể hiện là đối tượng trong không gian bài toán. Nó được tạo ra thuần túy bằng cách gộp những thuộc tính chung lại ở nhiều mức trừu tượng khác nhau cho đến khi cảm thấy chắc chắn không còn một lớp nào mới được tạo ra nữa.

Ví dụ: Sau khi phân tích kỹ bài toán quản lý nhân sự của một xí nghiệp chúng ta có được các lớp đối tượng: CAN_BO_QUAN_LY, NHAN_VIEN_PHUC_VU, CONG_NHAN, CAN_BO_TRUNG_CAP, KY_SU, TREN_DAI_HOC.

Ở mức thứ nhất chúng ta thấy hai lớp : CAN_BO_QUAN_LY, NHAN_VIEN_PHUC_VU, có thể gộp những đặc tính chung về những thuộc tính, chức năng liên quan đến công việc phục vụ, quản lý xí nghiệp để tạo ra một lớp mới là CAN_BO_GIAN_TIEP. Tương tự hai lớp CONG_NHAN, CAN_BO_TRUNG_CAP có những thuộc tính, chức năng chung là tham gia trực tiếp sản xuất ra sản phẩm của xí nghiệp vì vậy có thể gộp chung lại để tạo ra lớp mới TRUC_TIEP_SAN_XUAT. Những cán bộ thuộc lớp TREN_DAI_HOC và KY_SU có chức năng chung là nghiên cứu để phát triển sản xuất nên có thể gộp lại thành lớp CAN_BO_NGHIEN_CUU. Các mối quan hệ đó được thể hiện như sau:

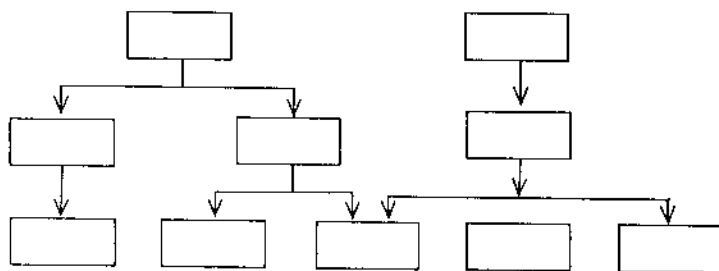


Các lớp mới được xây dựng: CAN_BO_GIAN_TIEP, TRUC_TIEP_SAN_XUAT, CAN_BO_NGHIEN_CUU lại có những thuộc tính chung là cán bộ, nhân viên trong cùng một xí nghiệp nên có thể gộp những đặc tính chung lại để tạo ra một lớp trừu tượng mới là NHAN_VIEN. Những gì đã mô tả trong các lớp cơ sở thì không cần nêu lại trong các lớp dẫn xuất. Sau khi phân tích kỹ mối quan hệ giữa các đối tượng để thiết kế lớp, chúng ta sẽ được cấu trúc phân cấp các lớp của hệ thống quản lý nhân sự như sau (theo nguyên lý tổng quát hoá).



Hình 2-2. Cấu trúc phân cấp các lớp (dạng cây)

Cấu trúc mà chúng ta thu được ở hình 2-2 có dạng cấu trúc cây. Tuy nhiên trong thực tế có nhiều hệ thống trong đó các lớp trừu tượng ở mức cuối không có những đặc tính chung để gộp tạo thành một lớp mới. Khi đó chúng ta có một dạng cấu trúc phân cấp dạng rừng (có nhiều hơn một nút gốc) như hình 2-3.



Hình 2-3. Cấu trúc phân cấp dạng rừng cây

Thiết kế các lớp

Trong bước phân tích, chúng ta đã xác định được các lớp với các thuộc tính và tập tối thiểu các hàm chính thực hiện trên các thuộc tính mô tả đối

tượng của lớp đó. Để xây dựng được thiết kế tổng thể cho hệ thống, chúng ta cần xem xét các lớp ở mức độ chi tiết, bổ sung thêm những thuộc tính, hàm cần thiết cho các lớp đối tượng. Ngoài những hàm thể hiện đặc tính cơ bản của đối tượng trong một lớp, chúng ta cần bổ sung các hàm phục vụ sau:

1. Những hàm quản lý lớp trả lời cho các câu hỏi sau:

+ Một đối tượng được tạo lập như thế nào?

+ Một đối tượng được huỷ bỏ như thế nào?

2. Những hàm thực hiện cài đặt lớp

+ Những phép toán nào được thực hiện trên dữ liệu kiểu lớp?

3. Những hàm truy nhập vào lớp

+ Làm thế nào chúng ta nhận được thông tin về các biến nội bộ của một lớp.

4. Hàm xử lý lỗi

+ Làm thế nào xử lý được các lỗi xuất hiện khi thao tác với các đối tượng.

Để thiết kế các lớp chúng ta cần phải biết rõ cách thức trao đổi thông tin giữa các đối tượng, các mối quan hệ về kế thừa, quan hệ thành phần và quan hệ về sử dụng lại trong các lớp. Chúng ta phải trả lời được những câu hỏi:

1. Các loại điều khiển truy nhập cần thiết cho các lớp cơ sở?

2. Những hàm nào là những hàm ảo? Hàm ảo là những hàm có cùng tên trong lớp cơ sở và lớp dẫn xuất, sẽ được phân biệt trong quá trình thực hiện bởi từng đối tượng cụ thể.

3. Những thư viện các lớp nào được sử dụng để thiết kế lớp.

Kết quả thiết kế lớp sẽ ảnh hưởng rất lớn đến chất lượng phần mềm. Vì vậy khi thiết kế lớp chúng ta cần chú ý những vấn đề sau:

1. Các lớp chỉ trao đổi với nhau thông qua các hàm.

2. Một đối tượng của một lớp không được gửi thông báo trực tiếp cho đối tượng của lớp khác.

3. Hàm được khai báo là chung (public) chỉ khi nó được sử dụng chung cho nhiều đối tượng của một lớp.

4. Mỗi hàm làm nhiệm vụ truy nhập hoặc làm thay đổi một số dữ liệu của lớp mà nó biểu diễn.

5. Sự phụ thuộc của một lớp vào các lớp khác càng ít càng tốt.

6. Tương tác giữa các lớp phải luôn luôn tường minh.

7. Lớp dẫn xuất là một trường hợp của lớp cơ sở, được bổ sung thêm một số đặc tính riêng để mô tả chi tiết hơn về lớp con của lớp cơ sở.

8. Lớp trên cùng của cấu trúc phân cấp biểu diễn mô hình khái niệm trừu tượng của hệ thống.

Thông thường khi thiết kế các lớp, hàm và chương trình chính, chúng ta nên sử dụng ngôn ngữ lập trình sẽ được chọn để cài đặt (tốt nhất là chọn ngôn ngữ lập trình hướng đối tượng như C++) để mô tả.

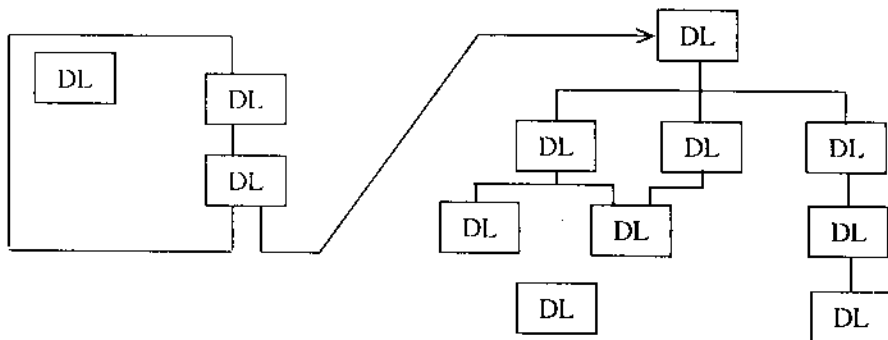
Thiết kế hàm thành phần

Đến giai đoạn này chúng ta đã xây dựng được:

1. Các đối tượng và lớp
2. Các thành phần dữ liệu
3. Giao diện giữa các đối tượng
4. Sự phụ thuộc của các lớp
5. Cấu trúc phân cấp các lớp

Bây giờ là lúc chúng ta cần xem xét đến việc thiết kế các hàm thành phần, những phép toán thực hiện trên các dữ liệu của đối tượng. Các hàm này cũng giống như các hàm trong ngôn ngữ lập trình C vì vậy chúng ta có thể sử dụng kỹ thuật phân rã chức năng trên-xuống (top-down) để thiết kế chúng.

Ví dụ: Một đối tượng có hai hàm thành phần F1, F2 tác động lên vùng dữ liệu DL, trong đó F2 lại có thể phân tích thành các khối chức năng nhỏ hơn như trong hình 2-4.



Hình 2.4. Thiết kế top-down các hàm thành phần

Trong mỗi khối chúng ta lại có thể sử dụng kỹ thuật thiết kế có cấu trúc để tạo ra cấu trúc phân cấp về chức năng cho những hàm phức tạp. Nhiều ngôn ngữ lập trình phổ dụng, như C, C++ đã được xây dựng để hỗ trợ cho phương pháp mô tả thiết kế và lập trình có cấu trúc. Chúng ta có thể cài đặt các đơn thể được cấu thành từ những cấu trúc tuần tự, tuyển chọn và vòng

lập. Có thể thiết kế các hàm thành phần theo phương pháp có cấu trúc. Kết quả của thiết kế có cấu trúc cho một hàm là một cấu trúc có một lối vào và một lối ra được tổ hợp từ một trong ba cấu trúc cơ bản: cấu trúc tuần tự, tuyển chọn và vòng lặp.

Thiết kế chương trình chính

Bước cuối cùng trong khâu thiết kế hệ thống là xây dựng chương trình chính, giống như chương trình main() trong ngôn ngữ C⁺⁺. Hệ thống được bắt đầu và kết thúc tại chương trình chính. Do vậy nhiệm vụ của chương trình chính là:

1. Nhập dữ liệu từ người sử dụng.
2. Tạo ra các đối tượng theo định nghĩa các lớp.
3. Tổ chức thực hiện trao đổi thông tin giữa các đối tượng.
4. Lưu trữ kết quả xử lý hoặc hiện lên màn hình, máy in, thiết bị ngoại vi theo yêu cầu người sử dụng.

Mọi hoạt động, xử lý trong quá trình thực hiện chương trình đều là kết quả của sự trao đổi, tương tác giữa các đối tượng. Vì vậy nhiệm vụ chủ yếu của thiết kế chương trình là xác định thứ tự logic của quá trình trao đổi thông tin giữa các đối tượng trong hệ thống.

Chương trình chính liên quan trực tiếp đến người sử dụng. Vì vậy trong thiết kế chúng ta cũng cần đề cập đến thiết kế giao diện thân thiện với người sử dụng.

Thiết kế giao diện người máy

Có nhiều kiểu thiết kế giao diện đã được tạo ra nhằm phục vụ cho người sử dụng khai thác hệ thống phần mềm sao cho có hiệu quả nhất. Mỗi kiểu đều có những đặc tính và khả năng khác nhau. Điều quan trọng là thiết kế giao diện phải phù hợp với lĩnh vực ứng dụng và những công việc của người sử dụng, những người tham gia trực tiếp đối thoại với máy tính. Nhìn chung, các hệ giao diện với người sử dụng đều cần phải có những tính chất sau:

- + Dễ sử dụng: Giao diện thân thiện, dễ sử dụng ngay cả với những người sử dụng không có kinh nghiệm.
- + Dễ học: Các lệnh, thao tác hệ thống được xây dựng theo những quy định chung, dễ tiếp thu và dễ nhớ.
- + Tốc độ thao tác nhanh, hợp lý: Các bước thao tác, ấn nút trên bàn phím,

con chuột nhanh gọn, tiện lợi cho người sử dụng. Thời gian thực hiện và trả lời trên máy tính nhanh và chính xác.

+ Đảm bảo an toàn: Kiểm soát được các tình huống, những thao tác cố tình hay vô ý của người sử dụng đều được xử lý tốt.

Dễ phát triển: Hệ thống có tính mở, có khả năng thay đổi, bổ sung theo yêu cầu của người sử dụng.

Dưới đây chúng ta sẽ đề cập đến một số kiểu thiết kế giao diện: dạng hỏi đáp, thực đơn và biểu tượng. Bạn đọc nào quan tâm sâu về thiết kế giao diện với người sử dụng có thể tham khảo cuốn "Phân tích, thiết kế và cài đặt hệ thông tin quản lý, Viện Tin học".

Thiết kế giao diện đối thoại: Việc thiết kế đối thoại bắt đầu bằng việc chia các chức năng về giao diện của hệ thống thành những đơn thể. Mỗi đơn thể sẽ đảm nhận đúng một chức năng của hệ thống. Ví dụ, đơn thể nhập dữ liệu chỉ làm nhiệm vụ kiểm soát dữ liệu nhập vào sao cho phù hợp với qui định của người thiết. Bảng đối thoại thường bao gồm một loạt những câu hỏi, thông báo nhắc về những công việc của hệ thống cần thực hiện. Thiết kế hỏi đáp phải bao quát hết các trường hợp, có đầy đủ chú thích, hướng dẫn trợ giúp người sử dụng. Cách thiết kế này phù hợp với những người sử dụng ít kinh nghiệm.

Thiết kế bảng thực đơn (Menu): Bảng thực đơn cho biết tất cả các công việc để lựa chọn. Thông thường bảng thực đơn nên tổ chức thành cấu trúc phân cấp. Mỗi mục trong bảng thực đơn chính lại được tổ chức thành bảng thực đơn con gồm một số mục để lựa chọn và được thể hiện ở dạng màn hình cửa sổ (Window).

Thiết kế biểu tượng: Biểu tượng được sử dụng để giới thiệu các chức năng của hệ thống trên màn hình. Mỗi chức năng được biểu diễn bằng một biểu tượng (hình vẽ tương ứng) sao cho dễ nhớ và dễ hình dung nhất.

Ở trên chúng ta đã nêu tất cả các bước của quá trình thiết kế hướng đối tượng. Phần còn lại của chương này chúng ta xây dựng thiết kế cho hai hệ: hệ quản lý kết quả học tập của học sinh và hệ điều khiển hệ thống điều hoà nhiệt độ.

2.3. Ví dụ

2.3.1. Bài toán thứ nhất: Thiết kế hệ thống quản lý

Bài toán đặt ra: Hãy xây dựng chương trình chạy trên máy tính để theo dõi kết quả học tập của sinh viên trong một lớp, và tìm kiếm theo kết quả điểm trung bình các môn thi. Giáo viên muốn quản lý sinh viên thông qua:

Họ và tên, các điểm thi: học kỳ 1, học kỳ 2, thi cuối năm và điểm trung bình qua các kỳ thi của các môn học.

Phân tích kỹ bài toán ở trên chúng ta xác định được hai đối tượng là: SINH_VIEN và MON_HOC. Tương ứng với các đối tượng này trong thiết kế là các lớp SINH_VIEN và MON_HOC. Vì đối tượng là thể hiện của lớp, nên hai lớp SINH_VIEN và MON_HOC xác định hai loại đối tượng cơ bản trong hệ thống theo dõi kết quả học tập mà giáo viên cần phải phát triển.

Bước tiếp theo trong thiết kế là mô tả chi tiết các lớp đối tượng và mối quan hệ giữa chúng. Theo yêu cầu của giáo viên (người sử dụng) thì mỗi đối tượng trong lớp SINH_VIEN được mô tả bởi các thuộc tính: Ho_ten, Diem_thi_ky1, Diem_thi_ky2, Diem_thi_CN và Diem_TB; còn MON_HOC cho biết danh sách các đối tượng SINH_VIEN theo môn học đó. Mỗi thành phần dữ liệu có một kiểu xác định và giá trị các dữ liệu thành phần mô tả trạng thái của đối tượng. Cuối cùng chúng ta có danh sách các thuộc tính mô tả lớp SINH_VIEN và MON_HOC như sau:

```
class SINH_VIEN
{
    // Danh sách các thuộc tính mô tả SINH_VIEN
    a. Ho_ten : string (kiểu xâu ký tự)
    b. Diem_thi_ky1 : float
    c. Diem_thi_ky2 : float
    d. Diem_thi_CN : float
    e. Diem_TB : float
}

class MON_HOC
{
    // Danh sách các thuộc tính mô tả MON_HOC
    a. Mon_hoc : string // tên môn học
    b. Danh_sach_SV : SINH_VIEN // Bảng danh sách các
                                // sinh viên theo học
    c. Si_so : integer // Số học sinh trong lớp
}
```

Sau khi xác định được các đối tượng, lớp và danh sách các thuộc tính mô tả đối tượng, bước tiếp theo là xác định các hàm thành phần của các lớp. Hàm thành phần mô tả hành vi của đối tượng bao gồm:

- + Truy nhập vào thành phần dữ liệu của đối tượng
- + Cho phép cập nhật, thay đổi dữ liệu thành phần
- + Kiểm tra dữ liệu theo nhiều cách khác nhau
- + Hiện các thông tin dữ liệu lên màn hình, thiết bị ngoại vi

Mục tiêu của thiết kế lớp là tạo ra các lớp cho nhiều ứng dụng khác nhau

nhưng cũng liên quan đến một loại dữ liệu. Người thiết kế các lớp muốn xây dựng các class đáp ứng được mọi nhu cầu của người lập trình (khách hàng). Quan hệ giữa người phát triển với người lập trình, cũng giống như quan hệ chủ hàng - khách hàng, dẫn đến trường hợp là chủ hàng mong sao có nhiều người sử dụng nhất, nghĩa là lớp sẽ được xây dựng với một số lượng lớn các hàm thành phần. Nhưng khách hàng lại chỉ sử dụng những hàm nào liên quan đến ứng dụng của họ mà thôi. Do vậy chúng ta phải cố gắng xây dựng được những hàm thành phần cơ bản, đặc trưng nhất cho lớp đang xét, sau đó sử dụng quan hệ kế thừa để tạo ra những lớp mới thích hợp cho nhiều ứng dụng khác nhau.

Trong bài toán của chúng ta, các hàm thành phần có thể là:

1. Các hàm thành phần của lớp SINH_VIEN

- | | |
|--------------------------------------|--------------|
| a. Hàm khởi tạo đối tượng sinh viên: | SINH_VIEN() |
| b. Hàm huỷ bỏ đối tượng sinh viên: | ~SINH_VIEN() |
| c. Hàm đọc tên sinh viên: | DOC_TEN() |
| d. Hàm gán tên của một sinh viên: | GAN_TEN() |
| e. Hàm đọc dữ liệu: | DOC_DL() |
| f. Hàm tính trung bình: | T_BINH() |
| g. Hàm hiện kết quả: | DISPLAY() |

2. Các hàm của lớp MON_HOC

- | | |
|------------------------------------|------------|
| a. Hàm khởi tạo môn học: | MON_HOC() |
| b. Hàm huỷ bỏ đối tượng môn học: | ~MON_HOC() |
| c. Hàm đọc tên môn học: | DOC_TEN() |
| d. Hàm đọc dữ liệu: | DOC_DL() |
| e. Hàm tính trung bình: | T_BINH() |
| f. Hàm bổ sung sinh viên vào lớp: | THEM_SV() |
| g. Hàm sắp xếp sinh viên theo tên: | SAP_XEP() |
| h. Hàm tìm kiếm một sinh viên: | TIM() |

Hàm làm nhiệm vụ khởi tạo các đối tượng của một lớp thường có cùng tên của lớp đối tượng đó và được gọi là cấu tử, còn hàm huỷ bỏ một đối tượng cũng cùng tên nhưng có dấu "~" đứng trước được gọi là huỷ tử. Hai hàm này sẽ được gọi thực hiện một cách tự động khi cần thiết. Hàm cấu tử được gọi khi cần định nghĩa, tạo lập một đối tượng, còn hàm huỷ tử được gọi khi chương trình thoát ra khỏi phạm vi mà đối tượng đó được định nghĩa. Một lớp có thể có nhiều hàm cấu tử và chúng được phân biệt bởi

đanh sách và kiểu của các tham biến. Nhưng chỉ có duy nhất một hàm hủy từ cho một lớp và là hàm không có tham số.

2.3.2. Bài toán thứ hai: Thiết kế hệ thống điều hoà nhiệt độ

Phát biểu bài toán: Một toà nhà có N phòng, mỗi phòng có máy điều hoà, máy cảm nhiệt và máy điều nhiệt. Hãy xây dựng hệ thống phần mềm điều khiển các máy điều hoà nhiệt độ theo yêu cầu người sử dụng.

Theo cách tiếp cận truyền thống thì chúng ta nghĩ ngay đến việc dùng cấu trúc mảng một chiều để lưu trữ nhiệt độ mà người sử dụng yêu cầu, nhiệt độ và trạng thái của máy điều hoà hiện tại trong các phòng. Khi đó hệ thống điều hoà nhiệt độ các phòng sẽ được thiết kế dựa theo thuật toán sau:

1. Lần lượt xem xét các phòng $I=1 \dots N$.

2. Nếu nhiệt độ trong phòng khác với nhiệt độ yêu cầu thì máy điều hoà chuyển sang trạng thái On (đóng) nếu nó đang ở trạng thái Off (tắt).

3. Ngược lại nếu nhiệt độ trong phòng làm việc đã đạt được như nhiệt độ người sử dụng yêu cầu thì máy điều hoà chuyển sang trạng thái Off nếu nó đang ở trạng thái On.

Cách thứ hai chúng ta thực hiện thiết kế theo cách tiếp cận hướng đối tượng. Nhiệm vụ trước tiên của chúng ta là xác định các đối tượng thành phần cơ sở của hệ thống điều hoà nhiệt độ. Phân tích bài toán chúng ta thấy ở đây có hai lớp đối tượng:

1. Đối tượng PHONG

2. Đối tượng DIEU_HOA

Trong mỗi phòng: nhiệt độ hiện thời được xác định bởi máy điều nhiệt. Máy điều hoà có hai trạng thái On (đóng) và Off (tắt). Trạng thái của máy điều hoà sẽ thay đổi tùy thuộc vào thông báo của PHONG: nhiệt độ trong phòng có cần phải thay đổi hay không.

Sau khi xác định được các lớp, các thuộc tính mô tả trạng thái của các đối tượng và mối quan hệ giữa hai lớp đối tượng chúng ta cần xác định các hàm thành phần của chúng.

Đối với lớp PHONG chúng ta có:

1. Hàm khởi tạo đối tượng: PHONG()
2. Hàm xoá bỏ đối tượng: ~PHONG()
3. Hàm xác định nhiệt độ qua máy cảm nhiệt: NHiet_KE()
4. Hàm đặt nhiệt độ yêu cầu vào máy điều nhiệt: T_YEU_CAU()

Đối với lớp DIEU_HOA:

1. Hàm khởi tạo đối tượng: DIEU_HOA()
2. Hàm xoá bỏ đối tượng: ~DIEU_HOA()
3. Hàm xác định trạng thái On, Off: DONG_MO()
4. Hàm xác định số phòng đặt máy điều hoà: PHONG_MAY()

Hệ điều khiển hệ thống điều hoà nhiệt độ các phòng không nhất thiết phải có sự tương ứng 1-1 giữa số phòng và máy điều hoà như điều kiện ban đầu bài toán đặt ra. Trong trường hợp cần thay đổi, ví dụ như đối với hội trường lớn thì cần phải đặt nhiều máy mới đủ công suất điều hoà nhiệt độ hoặc nhiều phòng dùng chung một máy.

Khi có sự thay đổi như trên thì thiết kế theo cách tiếp cận thứ nhất phải thay đổi ít nhiều, chương trình gần như phải viết lại toàn bộ vì chúng ta đã giả thiết rằng mỗi phòng có một máy điều hoà. Nhưng cách tiếp cận thứ hai (thiết kế hướng đối tượng) thì chỉ cần sửa đổi ở những đối tượng liên quan. Để sửa đổi hệ thống theo yêu cầu thì hàm DONG_MO() của đối tượng DIEU_HOA phải thiết kế lại sao cho nó có thể bật tắt nhiều hơn một máy và hàm T_YEU_CAU() của các đối tượng PHONG dùng chung một DIEU_HOA cần được thiết kế lại sao cho xác định được nhiệt độ bình quân của các phòng đó, còn cấu trúc hệ thống, mô hình tính toán không có gì thay đổi.

2.3.3. Bài toán thứ 3: Thiết kế hệ thống khí tượng thủy văn

Nội dung bài toán: Hệ thống gồm nhiều trạm đo số liệu và thu nhận thông tin dữ liệu về khí tượng thủy văn để dự báo thời tiết. Hệ thống thu nhận dữ liệu theo định kỳ, xử lý dữ liệu cục bộ và truyền những dữ liệu, thông tin cần thiết về cho máy tính trung tâm xử lý tiếp. Dữ liệu mà hệ thống tập hợp bao gồm: nhiệt độ khí quyển, lòng đất; tốc độ, hướng gió; áp suất và lưu lượng mưa.

Một trong những ưu điểm của cách tiếp cận hướng đối tượng là có thể tiến hành thiết kế ngay khi chưa có đủ tất cả các đặc tả yêu cầu và sau đó có thể dễ dàng thay đổi khi cần bổ sung hay sửa đổi các yêu cầu đã nêu. Bài toán mô tả ở trên chưa cho biết về tần số thu thập dữ liệu cũng như cách xử lý dữ liệu, nhưng chúng ta có thể tiến hành ngay xây dựng thiết kế cho hệ thống.

Trước tiên chúng ta cần xác định những thành phần cơ bản của hệ thống, nghĩa là các thực thể sẽ tương ứng với khái niệm đối tượng của hệ thống phần mềm mà chúng ta cần xây dựng. Nói chung, việc xác định các đối tượng được thực hiện theo cách làm mịn dần trong quá trình thiết kế. Hệ thống mà chúng ta xây dựng ở đây là hệ khí tượng thủy văn sẽ có cả đối

tượng "cứng" lẫn đối tượng "mềm". Đối tượng "cứng" không phải là hoàn toàn máy móc, thiết bị phần cứng mà là những thực thể liên quan đến máy móc, thiết bị. Những đối tượng này được nhúng vào hệ thống phần mềm để điều khiển các phần cứng tương ứng. Ngược lại, đối tượng "mềm" là những đối tượng chỉ tương tác với các đối tượng khác trong hệ thống, nghĩa là giúp các đối tượng "cứng" trao đổi thông tin với nhau. Phân tích kỹ bài toán chúng ta có thể xác định được các đối tượng "cứng" như sau:

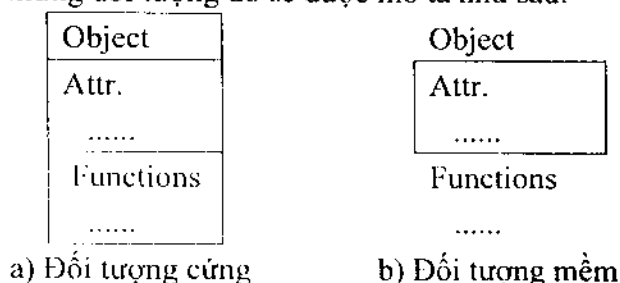
- + Trạm khí tượng thủy văn
- + Máy đo nhiệt độ khí quyển
- + Máy đo nhiệt độ lòng đất
- + Máy đo sức gió
- + Máy đo hướng gió
- + Máy đo áp suất khí quyển
- + Máy đo lưu lượng mưa
- + Đồng hồ (xác định thời gian thu nhận dữ liệu và truyền thông báo)
- + Modem (truyền và nhận thông báo từ máy tính trung tâm)

Nhiệm vụ của từng đối tượng được mô tả như trong Bảng 2.1.

Bảng 3.1. Xác định công việc của các đối tượng "cứng"

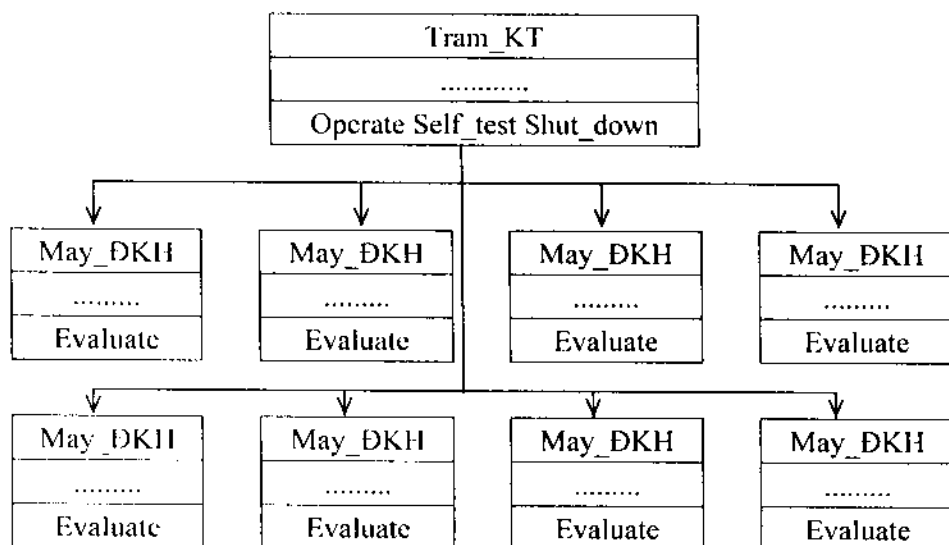
Đối tượng	Nhiệm vụ	Mô tả công việc cụ thể
TRAM_KT	Operate	Khởi động bằng cách bấm nút khởi động máy. TRAM_KT thu nhận thông tin và làm báo cáo để dự báo thời tiết.
	Self_test	Kiểm tra lại các kết quả đã thu nhận
	Shut_down	Bấm nút để dừng hoạt động của máy
May_DKH	Evaluate	Đo nhiệt độ không khí
May_DLD	Evaluate	Đo nhiệt độ lòng đất
May_DSG	Evaluate	Đo sức gió
May_DHG	Evaluate	Đo hướng gió, tính theo độ đo góc
May_DLA	Evaluate	Đo áp suất của không khí
May_DLM	Evaluate	Xác định lượng nước mưa kể từ lúc Reset lại
	Reset	Đặt lại máy đo, xóa các thông số cũ
DONG_HO	Time_now()	Xác định thời gian theo yêu cầu
	Reset	Đặt lại thời gian theo yêu cầu
Modem	Transmit	Truyền thông báo cho máy tính trung tâm
	Receive	Nhận thông báo từ máy tính trung tâm

Để dễ theo dõi, chúng ta nên sử dụng sơ đồ khối để mô tả thiết kế. Ở trên chúng ta đã xác định là trong hệ thống sẽ có hai loại đối tượng: cứng và mềm. Những đối tượng đó sẽ được mô tả như sau:



Các đối tượng "cứng" có quan hệ với nhau trong hệ thống như trong hình 2-6. Đối tượng TRAM_KT thu nhận các thông tin từ các đối tượng con là May_DKH, May_DLD, May_DSG, May_DHG, May_DKA, May_DLM, DONG_HO và Modem để xử lý sơ bộ rồi chuyển thông báo về cho máy tính trung tâm. Một điều chúng ta cần lưu ý ở đây là những đối tượng "cứng" không thể trao đổi thông tin trực tiếp với nhau mà phải thông qua các đối tượng "mềm".

Bước tiếp theo là xác định các đối tượng "mềm". Để làm được điều đó chúng ta cần tìm hiểu thêm một số thông tin về tần suất thu nhận dữ liệu và cách xử lý dữ liệu của hệ thống. Trong quá trình tiếp xúc với những chuyên gia, những người sử dụng hệ thống khí tượng thủy văn để xác định bài toán, chúng ta có thêm những mô tả như sau:



Hình 2-6. Cấu trúc phân cấp của phần cứng

"Tất cả các tham số về thời tiết, ngoại trừ lượng mưa sẽ được thu thập theo chu kỳ từng phút một. Sau mỗi giờ (sau 60 lần nhận dữ liệu) thì nhiệt độ, tốc độ gió, áp suất sẽ được xử lý để tính giá trị trung bình, giá trị cực đại, cực tiểu và trong 1 giờ đó, lượng mưa cũng được ghi nhận. Các hướng gió được đo theo độ đo góc α ($\alpha < 15^\circ$) cũng sẽ được ghi nhận. Tất cả những thông số đó sẽ được tính toán sơ bộ và được truyền về máy tính trung tâm để xử lý và dự báo về thời tiết"

Phân tích mô tả bài toán chúng ta thấy có những đối tượng "mềm" sau:

- + SUC_GIO Xác định tốc độ của gió
- + H_GIO Xác định hướng gió
- + ND_KH Đo nhiệt độ không khí
- + ND_LD Đo nhiệt độ lòng đất
- + I_MUA Đo lưu lượng nước mưa
- + A_SUAT Đo áp suất khí quyển

Những đối tượng này cung cấp thông tin cho một đối tượng làm nhiệm vụ tổng hợp dữ liệu là:

- + TH_DL Tạo ra các bản ghi thông tin từ các đối tượng con, kiểm tra, truyền và nhận thông báo.

Chức năng và nhiệm vụ của các đối tượng trên được mô tả chi tiết trong Bảng 2-2.

Bảng 2-2. Mô tả công việc của các đối tượng mềm

Đối tượng	Thao tác	Mô tả công việc
TH_DL	Create	Tạo lập DL để dự báo thời tiết
	Check	Kiểm tra tính hợp lý của DL
	Transmit	Gửi thông báo về trung tâm
SUC_GIO	Collect	Ghi nhận tốc độ của gió theo từng phút
	Max	Xác định tốc độ cực đại của gió
	Min	Xác định tốc độ cực tiểu của gió
	Mean	Xác định tốc độ trung bình của gió
H_GIO	Collect	Ghi nhận các hướng gió
	Variances	Danh sách hướng gió lớn hơn α
	Mean	Xác định độ đo trung bình

Đối tượng	Thao tác	Mô tả công việc
DN_KH	Collect Max Min Mean	Ghi nhận nhiệt độ không khí theo chu kỳ Xác định giá trị cực đại của nhiệt độ Xác định giá trị cực tiểu của nhiệt độ Xác định giá trị trung bình của nhiệt độ
DN_LD	Collect Max Min Mean	Ghi nhận nhiệt độ của lòng đất Xác định giá trị cực đại Xác định giá trị cực tiểu Xác định độ đo trung bình
L_MUA	Collect Max Min Mean	Ghi nhận lượng nước mưa Xác định giá trị cực đại Xác định giá trị cực tiểu Xác định độ đo trung bình
A_SUAT	Collect Max Min Mean	Đo đo áp suất không khí Xác định giá trị cực đại Xác định giá trị cực tiểu Xác định độ đo trung bình

Nhiệm vụ tiếp theo của thiết kế hướng đối tượng là thiết kế các lớp và xác định mối quan hệ giữa chúng. Ở đây chúng ta sử dụng C++ để đặc tả thiết kế các lớp đối tượng.

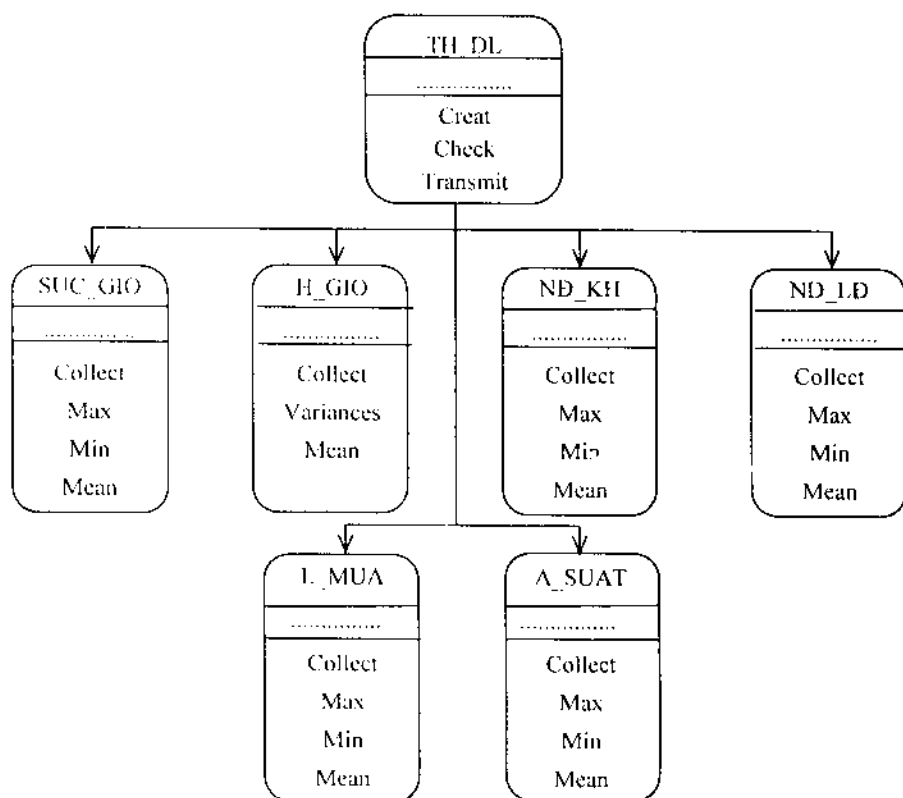
Trước tiên chúng ta xét lớp TH_DL. Lớp này làm nhiệm vụ tổng hợp dữ liệu, tạo ra các record dữ liệu nhận từ các đối tượng con về nhiệt độ, sức, hướng gió, lượng nước mưa, áp suất khí quyển, kiểm tra tính hợp lý của dữ liệu, truyền và nhận thông báo từ trung tâm. Lưu ý rằng, TH_DL không cần biết gì về cách thu nhập dữ liệu của các đối tượng con. Trong C++ chúng ta mô tả TH_DL như sau:

```
class TH_DL
{
private:
    ... // Khai báo các biến lưu trữ nhiệt độ, sức gió
    ... // hướng gió, áp suất, lượng mưa...
public:
    void Create(void); // Tạo lập các record dữ liệu
    void Check(void); // Kiểm tra dữ liệu
    void Transmit(void); // Nhận và truyền dữ liệu
}
```

Tương tự, chúng ta có thể xây dựng các lớp con SUC_GIO, H_GIO, ND_KH, ND_LD, L_MUA, A_SUAT. Ví dụ, lớp ND_KH có thể sẽ được mô tả như sau:

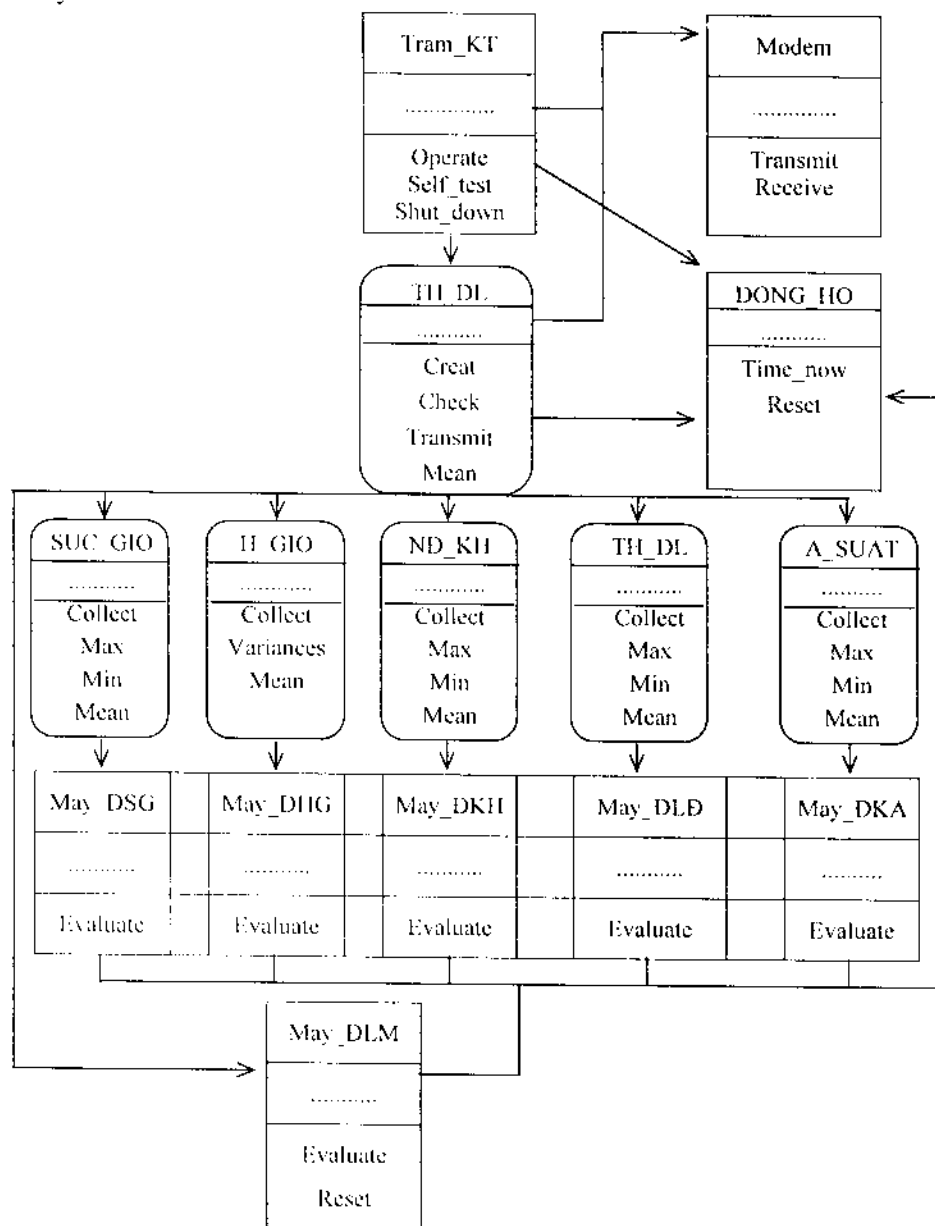
```
class ND_KH
{
    private:
        float t[60]; // Bảng số liệu đo nhiệt độ của không khí
                      // ghi được trong một chu kỳ 1 giờ
    public:
        float *Collect(); // Thu nhận dữ liệu từ
                          // đối tượng "cứng"
        float Max(); // Tính giá trị cực đại
                     // của nhiệt độ đo được
        float Min(); // Tính giá trị cực tiểu
        float Mean(); // Tính giá trị trung bình
}
```

Các lớp SUC_GIO, H_GIO, ND_KH, ND_LD, L_MUA, A_SUAT cung cấp dữ liệu cho lớp TH_DL và quan hệ giữa chúng được mô tả như trong hình 2-7.



Hình 2-7. Mô tả quan hệ của các lớp đối tượng "mềm"

Ở trên chúng ta đã khẳng định: các đối tượng "cứng" muốn trao đổi thông báo với nhau phải sử dụng các giao diện là các đối tượng "mềm". Dựa vào các chức năng, nhiệm vụ của từng lớp đối tượng và nhiệm vụ chung của hệ thống chúng ta có thể đưa ra thiết kế tổng thể cho hệ thống khí tượng thủy văn như sau:



Hình 2-8. Cấu trúc hệ thống khí tượng thủy văn

§3. LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

3.1. Giới thiệu

Với lập trình có cấu trúc, những hệ thống lớn, phức hợp, thì độ phức tạp của chương trình sẽ tăng lên, sự phụ thuộc của nó vào các kiểu dữ liệu mà nó xử lý cũng tăng theo. Các kiểu dữ liệu được xử lý trong nhiều thủ tục bên trong chương trình có cấu trúc, và khi có sự thay đổi trong kiểu dữ liệu thì cũng phải thực hiện thay đổi ở mọi nơi mà dữ liệu đó được sử dụng. Một nhược điểm nữa của lập trình có cấu trúc là khi có nhiều người tham gia xây dựng chương trình, mỗi người được giao viết một số hàm riêng biệt nhưng lại có thể sử dụng chung dữ liệu. Khi có nhu cầu cần thay đổi dữ liệu sẽ ảnh hưởng rất lớn đến công việc của nhiều người.

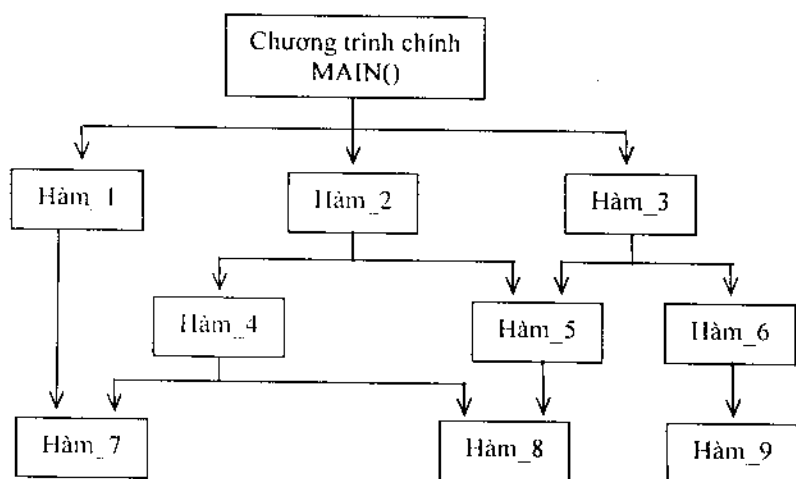
Lập trình hướng đối tượng dựa trên nền tảng là các đối tượng. Đối tượng được xây dựng trên cơ sở gắn cấu trúc dữ liệu với các phép toán sẽ thể hiện được đúng cách mà chúng ta suy nghĩ, bao quát về thế giới thực. Chẳng hạn, ô tô có bánh xe, di chuyển được và hướng của nó thay đổi được bằng cách thay đổi tay lái. Tương tự, cây là loại thực vật có thân gỗ và lá. Cây không phải là ô tô và những gì thực hiện được với ô tô sẽ không làm được với cây.

Lập trình hướng đối tượng cho phép chúng ta kết hợp những tri thức bao quát về các quá trình với những khái niệm trừu tượng được sử dụng trong máy tính. Chương trình hướng đối tượng xác định chính xác các đặc trưng và hành vi của các kiểu dữ liệu, trong đó có thể tạo ra những đối tượng mới được xây dựng từ những khuôn khổ có sẵn hay tổ hợp để tạo ra những đặc trưng mới.

Trong chương này chúng ta sẽ giới thiệu những khái niệm cơ bản và các bước cần thực hiện trong lập trình hướng đối tượng.

3.2. Lập trình hướng thủ tục (chức năng)

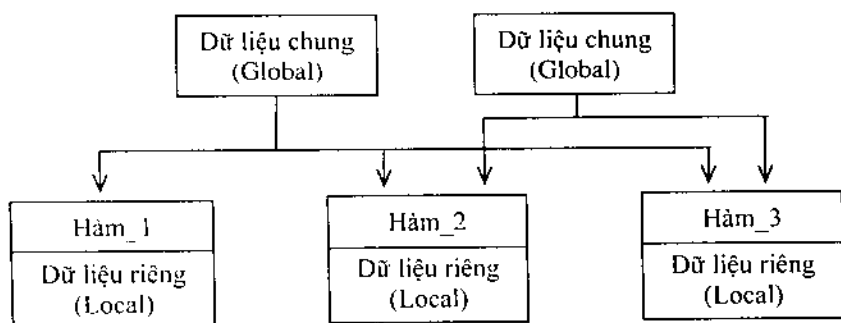
Những ngôn ngữ lập trình bậc cao truyền thống như COBOL, FOTRAN, PASCAL, C v.v..., được gọi chung là ngôn ngữ lập trình hướng thủ tục. Theo cách tiếp cận hướng thủ tục thì một hệ thống phần mềm được xem như là dãy các công việc cần thực hiện như đọc dữ liệu, tính toán, xử lý, lập báo cáo và in ấn kết quả v.v... Mỗi công việc đó sẽ được thực hiện bởi một số hàm nhất định. Như vậy trọng tâm của cách tiếp cận này là các hàm chức năng. Cấu trúc của chương trình được xây dựng theo cách tiếp cận hướng thủ tục có dạng như hình 3-1.



Hình 3-1. Cấu trúc của chương trình hướng thủ tục

Lập trình hướng thủ tục (LTHTT) sử dụng kỹ thuật phân rã hàm chức năng theo cách tiếp cận top-down để tạo ra cấu trúc phân cấp. Chương trình được xây dựng theo cách tiếp cận hướng thủ tục thực chất là danh sách các câu lệnh mà theo đó máy tính cần thực hiện. Danh sách các lệnh đó được tổ chức thành từng nhóm theo đơn vị cấu trúc cú pháp của ngôn ngữ đặc tả hay ngôn ngữ lập trình và được gọi là hàm (hay thủ tục). Để mô tả các hoạt động của các hàm, các dòng điều khiển và dữ liệu từ hoạt động này sang hoạt động khác người ta thường dùng sơ đồ khối.

Khi tập trung vào trọng tâm phát triển các hàm thì chúng ta lại ít chú ý đến dữ liệu, những cái mà các hàm sử dụng để thực hiện công việc của mình. Cái gì sẽ xảy ra đối với dữ liệu và gắn dữ liệu với các hàm như thế nào? cùng nhiều vấn đề khác cần phải giải quyết khi chúng ta muốn xây dựng các phương pháp thích hợp để phát triển hệ thống trong thế giới thực. Trong chương trình có nhiều hàm, thường thì có nhiều thành phần dữ liệu quan trọng sẽ được khai báo tổng thể (global) để cho nhiều hàm có thể truy nhập, đọc và làm thay đổi giá trị của biến tổng thể. Mỗi hàm có thể có vùng dữ liệu riêng còn gọi là dữ liệu cục bộ (local). Mỗi quan hệ giữa dữ liệu và hàm trong chương trình hướng thủ tục được mô tả trong hình 3-2.



Hình 3-2. Quan hệ dữ liệu và hàm trong LTHTT

Nhiều hàm có thể truy nhập, sử dụng dữ liệu chung, làm thay đổi giá trị của chúng và vì vậy rất khó kiểm soát. Nhất là đối với các chương trình lớn, phức tạp thì vấn đề càng trở nên khó khăn hơn.

Khi chúng ta muốn thay đổi, bổ sung cấu trúc dữ liệu dùng chung cho một số hàm thì chúng ta phải thay đổi hầu như tất cả các hàm liên quan đến dữ liệu đó.

Ngoài những trở ngại mà chúng ta đã nêu ở trên thì mô hình được xây dựng theo cách tiếp cận hướng thủ tục không mô tả được đầy đủ, trung thực hệ thống trong thực tế. Bởi vì cách đặt trọng tâm vào hàm là hướng tới hoạt động sẽ không thực sự tương ứng với các thực thể trong hệ thống của thế giới thực.

Tóm lại những đặc tính chính của lập trình hướng thủ tục là:

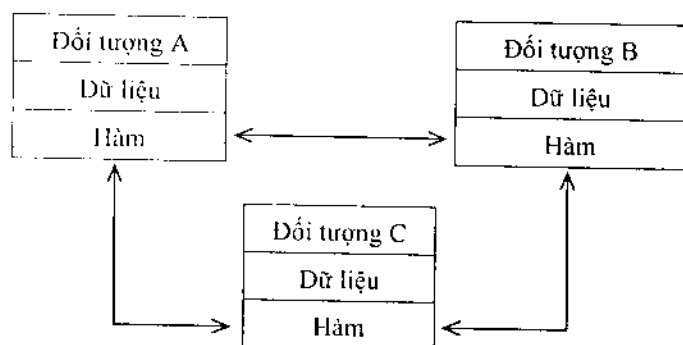
- + Tập trung vào công việc cần thực hiện (thuật toán).
- + Chương trình lớn được chia thành các hàm nhỏ hơn.
- + Phần lớn các hàm sử dụng dữ liệu chung.
- + Dữ liệu trong hệ thống được chuyển động từ hàm này sang hàm khác.
- + Hàm biến đổi dữ liệu từ dạng này sang dạng khác.
- + Sử dụng cách tiếp cận top-down trong thiết kế chương trình.

3.3. Lập trình hướng đối tượng

Như ở các phần trước chúng ta đã nêu, để giải quyết được những vấn đề tồn tại trong công nghệ phần mềm thì chúng ta cần phải sử dụng những phương pháp, công cụ thích hợp để phát triển phần mềm. Trong các mục §1 và §2 chúng ta đã đề cập đến phương pháp phân tích, thiết kế hướng đối

tượng. Trong mục này chúng ta tiếp tục nghiên cứu về phương pháp lập trình hướng đối tượng.

Lập trình hướng đối tượng đặt trọng tâm vào đối tượng, yếu tố quan trọng trong quá trình phát triển chương trình và nó không cho phép dữ liệu chuyển động tự do trong hệ thống. Dữ liệu được gắn chặt với từng hàm thành các vùng riêng mà các hàm đó tác động lên và nó được bảo vệ cấm các hàm bên ngoài không được truy nhập một cách tùy tiện. LTHĐT cho phép chúng ta phân tích bài toán thành tập các thực thể được gọi là các đối tượng và sau đó xây dựng các dữ liệu cùng với các hàm xung quanh các đối tượng đó. Tổ chức dữ liệu và hàm trong các chương trình hướng đối tượng được mô tả như trong hình 3-3.



Hình 3-3. Tổ chức dữ liệu và hàm trong chương trình HĐT

Dữ liệu của một đối tượng chỉ có thể được truy nhập bởi chính các hàm xác định trong đối tượng đó. Tuy nhiên các hàm của đối tượng này có thể truy nhập tới các hàm của đối tượng khác, nghĩa là các đối tượng trao đổi với nhau thông qua việc trao đổi thông báo. Lập trình hướng đối tượng có những đặc tính chủ yếu sau:

1. Tập chung vào dữ liệu thay cho các hàm.
2. Chương trình được chia thành các đối tượng.
3. Các cấu trúc dữ liệu được thiết kế sao cho đặc tả được các đối tượng.
4. Các hàm xác định trên các vùng dữ liệu của đối tượng được gắn với nhau trên cấu trúc dữ liệu đó.
5. Dữ liệu được bao bọc, che giấu và không cho phép các hàm ngoại lai truy nhập tự do.
6. Các đối tượng trao đổi với nhau thông qua các hàm.

7. Dữ liệu và các hàm mới có thể dễ dàng bổ sung vào đối tượng nào đó khi cần thiết.

8. Chương trình được thiết kế theo cách tiếp cận bottom-up (dưới-lên).

Lập trình hướng đối tượng là khái niệm mới và được hiểu rất khác nhau đối với nhiều người. Ở đây chúng ta có thể hiểu lập trình hướng đối tượng là cách tiếp cận để phân chia chương trình thành các đơn thể (module) bằng cách tạo ra các vùng bộ nhớ cho cả dữ liệu lẫn hàm và chúng sẽ được sử dụng như các mẫu để tạo ra bản sao từng đơn thể khi cần thiết. Đối tượng ở đây được xem như là vùng phân chia bộ nhớ trong máy tính để lưu trữ dữ liệu và tập các hàm tác động trên dữ liệu gắn với chúng. Bởi vì các vùng phân hoạch bộ nhớ là độc lập với nhau nên các đối tượng có thể sử dụng bởi nhiều chương trình khác nhau mà không ảnh hưởng lẫn nhau.

Khái niệm "hướng đối tượng" nhiều người hiểu rất khác nhau. Vì vậy, để hiểu rõ bản chất và có thể đi đến thống nhất quan điểm, chúng ta cần phải nghiên cứu kỹ những khái niệm cơ bản trong LTHĐT. Trong phần này chúng ta đề cập đến những khái niệm sau:

1. Đối tượng
2. Lớp
3. Trừu tượng hoá dữ liệu
4. Kế thừa
5. Tương ứng bội
6. Liên kết động
7. Truyền thông báo

Đối tượng

Trong các mục trước chúng ta đã nêu cách xác định đối tượng trong quá trình phân tích và thiết kế hướng đối tượng. Ở đây chúng ta tìm hiểu chi tiết hơn để hiểu rõ vai trò của đối tượng trong cách tiếp cận hướng đối tượng nói chung và LTHĐT nói riêng.

Đối tượng là thực thể được xác định trong thời gian hệ thống hướng đối tượng hoạt động. Như vậy đối tượng có thể biểu diễn là con người, vật, hay một bảng dữ liệu hoặc bất kỳ một hạng thức nào đó cần xử lý trong chương trình. Đối tượng cũng có thể là các dữ liệu được định nghĩa bởi người sử dụng (người lập trình) như vector, danh sách, các record v.v... Nhiệm vụ của LTHĐT là phân tích bài toán thành các đối tượng và xác định được bản chất của sự trao đổi thông tin giữa chúng. Đối tượng trong chương trình cần phải

được chọn sao cho nó thể hiện được một cách gần nhất so với những thực thể trong thế giới thực.

Khi chương trình thực hiện, các đối tượng sẽ trao đổi với nhau bằng cách gửi hay nhận thông báo. Ví dụ BAN_DOC và CHO_MUON là hai đối tượng trong hệ thống thư viện, đối tượng BAN_DOC có thể gửi một thông báo (bằng phiếu yêu cầu chẳng hạn) cho đối tượng CHO_MUON yêu cầu mượn cuốn "LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG VỚI C++". Mỗi đối tượng có dữ liệu và các hàm để xử lý dữ liệu đó. Các đối tượng trao đổi với nhau mà không cần biết chi tiết về dữ liệu và các thuật toán xử lý của đối tượng khác. Để trao đổi được với nhau, mỗi đối tượng chỉ cần biết kiểu thông báo mà nó nhận và kiểu thông báo mà nó sẽ gửi cho các đối tượng khác.

Các lớp đối tượng

Như trên chúng ta đã xác định, đối tượng trong chương trình gồm cả dữ liệu và các hàm xử lý trên dữ liệu đó. Một tập dữ liệu và các hàm của một đối tượng có thể được xem như một kiểu dữ liệu được định nghĩa bởi người sử dụng. Kiểu dữ liệu ở đây được gọi là lớp (class). Trong lập trình, các đối tượng là các biến có kiểu class. Khi một lớp được định nghĩa, thì nó có thể tạo ra số lượng các đối tượng tùy ý của lớp đó. Như vậy, TOA, LE, BUOI, CAM là các loại quả trong lớp HOA_QUA. Lớp là kiểu được người sử dụng định nghĩa và nó cũng có các tính chất như các kiểu chuẩn integer, float trong các ngôn ngữ lập trình. Tương tự như kiểu dữ liệu đã được định nghĩa trong chương trình, lệnh khai báo

HOA_QUA TAO;

sẽ tạo ra đối tượng TAO trong lớp HOA_QUA.

Trừu tượng hoá dữ liệu và bao gói thông tin

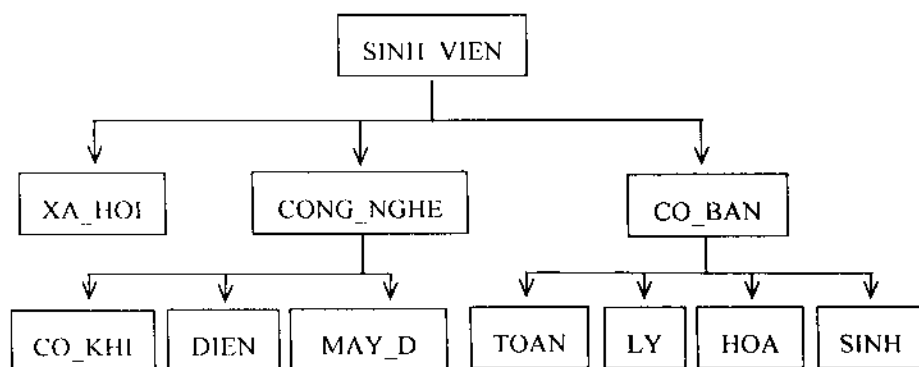
Việc đóng gói dữ liệu và các hàm vào một đơn vị cấu trúc (được gọi là lớp) được xem như một nguyên tắc bao gói (che giấu) thông tin. Dữ liệu được tổ chức sao cho thế giới bên ngoài (các đối tượng ở lớp khác) không truy nhập được vào mà chỉ cho phép các hàm trong cùng lớp hoặc trong những lớp có quan hệ kế thừa với nhau được quyền truy nhập. Chính các hàm thành phần của lớp sẽ đóng vai trò như là giao diện giữa dữ liệu của đối tượng và phần còn lại của chương trình. Nguyên tắc bao gói dữ liệu để ngăn cản sự truy nhập trực tiếp trong lập trình được gọi là sự che giấu thông tin.

Trừu tượng hoá là cách biểu diễn những đặc tính và bỏ qua những chi tiết vụn vặt hoặc những giải thích. Để xây dựng các lớp, chúng ta phải sử dụng khái niệm trừu tượng hoá. Ví dụ chúng ta có thể định nghĩa một lớp là danh sách các thuộc tính trừu tượng như là kích thước, hình dáng, màu và các

hàm xác định trên các thuộc tính này để mô tả các đối tượng trong không gian hình học. Trong lập trình, lớp sử dụng như kiểu dữ liệu trừu tượng.

Kế thừa

Kế thừa là quá trình mà các đối tượng của lớp này được quyền sử dụng một số tính chất của các đối tượng của lớp khác. Nguyên lý kế thừa hỗ trợ cho việc tạo ra cấu trúc phân cấp các lớp. Ví dụ, một trường đại học đào tạo sinh viên theo ba khối: Xã hội, Công nghệ, và Khoa học cơ bản. Mỗi khối lại có các khoa. Khối công nghệ có các khoa: Cơ khí, Điện, Máy dân dụng; còn khối Khoa học cơ bản có các khoa: Toán, Lý, Hoá, Sinh. Hệ thống sẽ tổ chức thành cấu trúc phân cấp các lớp kế nhau như sau:



Hình 3.4. Cấu trúc phân cấp các lớp trong quân hệ kế thừa

Lớp `SINH_VIEN` mô tả những thuộc tính chung nhất của sinh viên tất cả các khối trong trường ví dụ như: Họ và tên, quê, tuổi. Những đặc tính đó được kế thừa ở trong các lớp `XA_HOI`, `CONG_NGHE`, `CO_BAN`. Các lớp dẫn xuất đó được bổ sung thêm những thuộc tính, các hàm tương ứng mô tả cho sinh viên. Các lớp `XA_HOI`, `CONG_NGHE`, `CO_BAN` được bổ sung thêm những thuộc tính mới để phân biệt giữa các khối với nhau. Trong khối `CO_BAN` lại chia thành nhiều khoa như: `TOAN`, `LY`, `HOA`, `SINH`; khối `CONG_NGHE` chia thành các khoa: `CO_KHI`, `DIEN` và `MAY_DD`. Những lớp sau có những thuộc tính mô tả cho sinh viên của từng khoa.

Trong LTHĐT, khái niệm kế thừa kéo theo ý tưởng sử dụng lại. Nghĩa là từ một lớp đã được xây dựng chúng ta có thể bổ sung thêm một số tính chất tạo ra một lớp mới kế thừa lớp cũ mà không làm thay đổi những cái đã có.

Khái niệm kế thừa được hiểu như cơ chế sao chép ảo không đơn điệu. Trong thực tế, mọi việc xảy ra tựa như những lớp cơ sở đều được sao vào

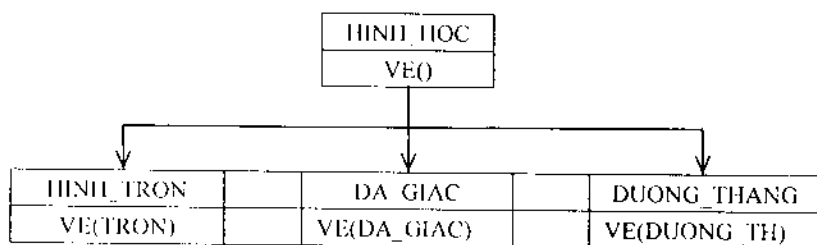
trong lớp con (lớp dẫn xuất) mặc dù điều này không được cài đặt tường minh (nên gọi là sao chép ảo) và việc sao chép chỉ thực hiện đối với những thông tin chưa được xác định trong các lớp cơ sở (sao chép không đơn điệu). Do vậy, có thể diễn đạt cơ chế kế thừa như sau:

1. Lớp A kế thừa lớp B sẽ có (không tường minh) tất cả các thuộc tính, hàm đã được xác định trong B.

2. Bổ sung thêm một số thuộc tính, hàm để mô tả được đúng các hành vi của những đối tượng mà lớp A quản lý.

Tương ứng bội

Một khái niệm quan trọng nữa trong LTHDT là khái niệm tương ứng bội. Tương ứng bội là khả năng của một khái niệm (như các phép toán) có thể được xuất hiện ở nhiều dạng khác nhau. Ví dụ, phép + có thể biểu diễn cho phép "cộng" các số nguyên (int), số thực (float), số phức (complex) hoặc xâu ký tự (string) v.v... Hành vi của phép toán tương ứng bội phụ thuộc vào kiểu dữ liệu mà nó sử dụng để xử lý. Hình 3-5 cho chúng ta thấy hàm có tên là VE có thể sử dụng để vẽ các hình khác nhau phụ thuộc vào tham số (được phân biệt bởi số lượng, kiểu của tham số) khi gọi để thực hiện.



Hình 3-5. Tương ứng bội của hàm VE()

Hàm VE() là hàm tương ứng bội và nó được xác định tùy theo ngữ cảnh khi sử dụng.

Tương ứng bội đóng vai trò quan trọng trong việc tạo ra các đối tượng có cấu trúc bên trong khác nhau nhưng có khả năng cùng dùng chung một giao diện bên ngoài (như tên gọi). Điều này có nghĩa là một lớp tổng quát các phép toán được định nghĩa theo cùng một cách giống nhau. Tương ứng bội là mở rộng khái niệm sử dụng lại trong nguyên lý kế thừa.

Liên kết động

Liên kết động là dạng liên kết các hàm, thủ tục khi chương trình thực hiện các lời gọi tới các hàm, thủ tục đó. Như vậy trong liên kết động, nội dung của đoạn chương trình ứng với thủ tục, hàm sẽ không được biết cho

đến khi thực hiện lời gọi tới thủ tục, hàm đó. Liên kết động liên quan chặt chẽ tới tương ứng bội và kế thừa. Chúng ta hãy lưu ý hàm VE() trong Hình 4-5. Theo nguyên lý kế thừa thì mọi đối tượng đều có thể sử dụng hàm này để vẽ hình theo yêu cầu. Tuy nhiên, thuật toán thực hiện hàm VE() là duy nhất đối với từng đối tượng HINH_TRON, DA_GIAC, DUONG_THANG và vì vậy hàm VE() sẽ được định nghĩa lại khi các đối tượng tương ứng được xác định. Khi thực hiện, ví dụ như khi vẽ một hình tròn, đoạn chương trình ứng với hàm VE() hình tròn được gọi ra để thực hiện.

Truyền thông báo

Chương trình hướng đối tượng (được thiết kế và lập trình theo hướng đối tượng) bao gồm một tập các đối tượng và mối quan hệ giữa các đối tượng với nhau. Vì vậy, lập trình trong ngôn ngữ hướng đối tượng bao gồm các bước sau:

1. Tạo ra các lớp xác định các đối tượng và hành vi của chúng.
2. Tạo ra các đối tượng theo định nghĩa của các lớp.
3. Xác định sự trao đổi giữa các đối tượng.

Các đối tượng gửi và nhận thông tin với nhau giống như con người trao đổi với nhau. Chính nguyên lý trao đổi thông tin bằng cách truyền thông báo cho phép chúng ta dễ dàng xây dựng được hệ thống mô phỏng gần hơn những hệ thống trong thế giới thực. Truyền thông báo cho một đối tượng tức là báo cho nó phải thực hiện một việc gì đó. Cách ứng xử của đối tượng sẽ được mô tả ở trong lớp thông qua các hàm (hay còn được gọi là lớp dịch vụ).

Trong chương trình, thông báo gửi đến cho một đối tượng chính là yêu cầu thực hiện một công việc cụ thể, nghĩa là sử dụng những hàm tương ứng để xử lý dữ liệu đã được khai báo trong đối tượng đó. Vì vậy, trong thông báo phải chỉ ra được hàm cần thực hiện trong đối tượng nhận thông báo. Hơn thế nữa, thông báo truyền đi phải xác định tên đối tượng, tên hàm (thông báo) và thông tin truyền đi. Ví dụ, lớp CONG_NHAN có thể hiện là đối tượng cụ thể được đại diện bởi Ho_Ten nhận được thông báo cần tính lương thông qua hàm TINH_LUONG đã được xác định trong lớp CONG_NHAN. Thông báo đó sẽ được xử lý như sau:

CONG_NHAN.TINH_LUONG (Ho_Ten)


 Đối tượng


 Thông báo


 Thông tin

Mỗi đối tượng chỉ tồn tại trong thời gian nhất định. Đối tượng được tạo ra khi nó được khai báo và sẽ bị huỷ bỏ khi chương trình ra khỏi miền xác định của đối tượng đó. Sự trao đổi thông tin chỉ có thể thực hiện trong thời gian đối tượng tồn tại.

Các ưu điểm của lập trình hướng đối tượng

Như trên chúng ta đã phân tích, lập trình hướng đối tượng đem lại một số lợi thế cho cả người thiết kế lẫn người lập trình. Cách tiếp cận hướng đối tượng giải quyết được nhiều vấn đề tồn tại trong quá trình phát triển phần mềm và tạo ra được những sản phẩm phần mềm có chất lượng cao. Những phương pháp này mở ra một triển vọng to lớn cho những người lập trình. Hy vọng sẽ có nhiều sản phẩm phần mềm tốt hơn, đáp ứng được những tính chất về sản phẩm chất lượng cao trong công nghệ phần mềm và nhất là bảo trì hệ thống ít tốn kém hơn. Những ưu điểm chính của LTHĐT là:

1. Thông qua nguyên lý kế thừa, chúng ta có thể loại bỏ được những đoạn chương trình lặp lại, dư thừa trong quá trình mô tả các lớp và mở rộng khả năng sử dụng các lớp đã được xây dựng.

2. Chương trình được xây dựng từ những đơn thể (đối tượng) trao đổi với nhau nên việc thiết kế và lập trình sẽ được thực hiện theo quy trình nhất định chứ không phải dựa vào kinh nghiệm và kỹ thuật như trước. Điều này đảm bảo rút ngắn được thời gian xây dựng hệ thống và tăng năng suất lao động.

3. Nguyên lý giấu thông tin giúp người lập trình tạo ra được những chương trình an toàn không bị thay bởi những đoạn chương trình khác.

4. Có thể xây dựng được ánh xạ các đối tượng của bài toán vào đối tượng của chương trình.

5. Cách tiếp cận thiết kế đặt trọng tâm vào đối tượng, giúp chúng ta xây dựng được mô hình chi tiết và gần với dạng cài đặt hơn.

6. Những hệ thống hướng đối tượng dễ mở rộng, nâng cấp thành những hệ lớn hơn.

7. Kỹ thuật truyền thông báo trong việc trao đổi thông tin giữa các đối tượng giúp cho việc mô tả giao diện với các hệ thống bên ngoài trở nên đơn giản hơn.

8. Có thể quản lý được độ phức tạp của những sản phẩm phần mềm.

Không phải trong hệ thống hướng đối tượng nào cũng có tất cả các tính chất nêu trên. Khả năng có các tính chất đó còn phụ thuộc vào lĩnh vực ứng

dụng của dự án tin học và vào phương pháp thực hiện của người phát triển phần mềm.

3.4. Các ngôn ngữ hướng đối tượng

Lập trình hướng đối tượng không là đặc quyền của một ngôn ngữ nào đặc biệt. Cũng giống như lập trình có cấu trúc, những khái niệm trong lập trình hướng đối tượng có thể cài đặt trong những ngôn ngữ lập trình như C hoặc Pascal. Tuy nhiên, đối với những chương trình lớn, phức hợp thì vấn đề lập trình sẽ trở nên phức tạp, nếu sử dụng những ngôn ngữ không phải là ngôn ngữ hướng đối tượng thì phải thực hiện nhiều thỏa hiệp. Những ngôn ngữ được thiết kế đặc biệt, hỗ trợ cho việc mô tả, cài đặt các khái niệm của phương pháp hướng đối tượng được gọi chung là ngôn ngữ hướng đối tượng.

Dựa vào khả năng đáp ứng các khái niệm về hướng đối tượng, chúng ta có thể chia ra làm hai loại:

1. Ngôn ngữ lập trình dựa trên đối tượng (object - based)
2. Ngôn ngữ lập trình hướng đối tượng (object - oriented)

Lập trình dựa trên đối tượng là kiểu lập trình hỗ trợ chính cho việc bao gói, che giấu thông tin và định danh các đối tượng. Lập trình dựa trên đối tượng có những đặc tính sau:

- + Bao gói dữ liệu
- + Cơ chế che giấu và truy nhập dữ liệu
- + Tự động tạo lập và xoá bỏ các đối tượng
- + Phép toán tải bội

Ngôn ngữ hỗ trợ cho kiểu lập trình trên được gọi là ngôn ngữ lập trình dựa trên đối tượng. Ngôn ngữ trong lớp này không hỗ trợ cho việc thực hiện kế thừa và liên kết động. Ada là ngôn ngữ lập trình dựa trên đối tượng.

Lập trình hướng đối tượng là kiểu lập trình dựa trên đối tượng và bổ sung thêm nhiều cấu trúc để cài đặt những quan hệ về kế thừa và liên kết động. Vì vậy đặc tính của LTHĐT có thể viết một cách ngắn gọn như sau:

Các đặc tính dựa trên đối tượng + kế thừa + liên kết động.

Ngôn ngữ hỗ trợ cho những đặc tính trên được gọi là ngôn ngữ LTHĐT, ví dụ như C++, Smalltalk, Object Pascal hay Eiffel v.v...

Việc chọn một ngôn ngữ để cài đặt phần mềm phụ thuộc nhiều vào các đặc tính và yêu cầu của bài toán ứng dụng, vào khả năng sử dụng lại của những chương trình đã có và vào tổ chức của nhóm tham gia xây dựng

phần mềm. Tương tự như ngôn ngữ lập trình C, C⁺⁺ đang được sử dụng rộng rãi, và rất thành công trong việc sử dụng để cài đặt các hệ thống phần mềm phức tạp.

3.5. Những ứng dụng của LTHĐT

LTHĐT là một trong những thuật ngữ được nhắc đến nhiều nhất hiện nay trong công nghệ phần mềm và nó được ứng dụng để phát triển phần mềm trong nhiều lĩnh vực khác nhau. Trong số đó, có ứng dụng quan trọng và nổi tiếng nhất hiện nay là lĩnh vực thiết kế giao diện với người sử dụng, ví dụ như Windows. Hàng trăm hệ thống với giao diện Windows đã được phát triển dựa trên kỹ thuật LTHĐT. Những hệ thống tin doanh nghiệp trong thực tế rất phức tạp, chứa nhiều đối tượng với các thuộc tính và hàm phức tạp. Để giải quyết những hệ thống phức hợp thế thì LTHĐT lại tỏ ra khá hiệu quả. Tóm lại, những lĩnh vực ứng dụng của kỹ thuật LTHĐT bao gồm:

- + Những hệ thống làm việc theo thời gian thực.
- + Trong lĩnh vực mô hình hoá hoặc mô phỏng các quá trình.
- + Các cơ sở dữ liệu hướng đối tượng.
- + Những hệ siêu văn bản, multimedia.
- + Lĩnh vực trí tuệ nhân tạo và các hệ chuyên gia.
- + Lập trình song song và mạng nơ-ron.
- + Những hệ tự động hoá văn phòng và trợ giúp quyết định.
- + Những hệ CAM/CAM.

Với nhiều đặc tính phong phú của LTHĐT nói riêng, của phương pháp phát triển hướng đối tượng nói chung, chúng ta hy vọng nền công nghệ phần mềm sẽ cải tiến không những về chất lượng mà còn gia tăng nhanh về số lượng trong tương lai. Kỹ nghệ hướng đối tượng sẽ làm thay đổi cách suy nghĩ và cách thực hiện quá trình phân tích, thiết kế và cài đặt các hệ thống, góp phần giải quyết những vấn đề tồn tại trong công nghệ phần mềm.

MỘT SỐ CHƯƠNG TRÌNH ĐỀ QUY

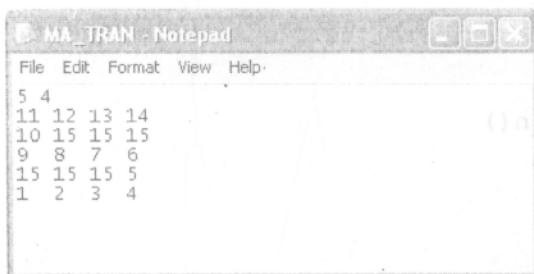
Trong phụ lục này sẽ trình bày một số chương trình minh họa thuật toán đề quy kiểu quay lui có tính chất kinh điển. Chương trình được tổ chức thành nhiều lớp. Các bài toán này thường có mặt trong các kỳ thi Olympic Tin học, thi cao học.

Bài toán 1: Tìm đường đi đơn điệu giảm dài nhất.

Mô tả cụ thể bài toán như sau: Cho một ma trận. Yêu cầu đặt ra là: Cho phép chọn một tọa độ (hàng, cột) của một phần tử của ma trận và từ đó hãy chỉ ra đường đi đơn điệu giảm dài nhất từ phần tử đó (nếu có).

Ghi chú: Chương trình cần tệp MA_TRAN.TXT

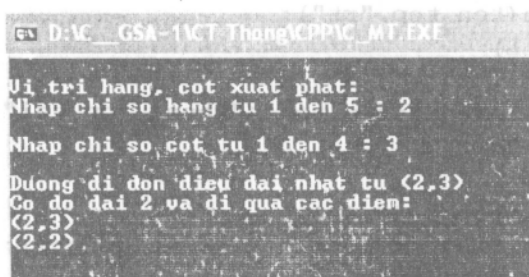
Chẳng hạn, với nội dung của tệp MA_TRAN.TXT như sau (5 hàng, 4 cột):



```

File Edit Format View Help
5 4
11 12 13 14
10 15 15 15
9 8 7 6
15 15 15 5
1 2 3 4
  
```

Khi đó, nếu chọn phần tử (2, 3) thì chương trình sẽ chạy như trong hình minh họa sau:



```

D:\C_GSA-1\CT Thong\CPP\IC_MT.EXE
Vị trí hàng, cột xuất phát:
Nhập chỉ số hàng từ 1 đến 5 : 2
Nhập chỉ số cột từ 1 đến 4 : 3
Đường đi đơn điệu giảm dài nhất từ (2,3)
Go dọc dài 2 và đi qua các điểm:
(2,3)
(2,2)
  
```

```

// C_MT.CPP
// Cho vi tri xuat phat (u,v) - hang u, cot v
// Tim duong di dai nhat xuat phat tu (u,v) qua
// cac diem ke va tao thanh day don dieu khong giam
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
int kmax=0;
int hmax[50], cmax[50];
class MT
{
    private:
    int m, n;
    int a[50][50];
    public:
    int get_m();
    int get_n();
    int doc_tep(char *ten_tep);
    //void xuat();
    friend void di_tiep(int *h, int *c, int k);
    friend int chap_nhan(int *h, int *c, int k,
                        int i, int j);
};
MT x;
int MT::get_m()
{
    return m;
}
int MT::get_n()
{
    return n;
}

int MT::doc_tep(char *ten_tep)
{
    FILE *fp;
    int i,j;
    fp=fopen(ten_tep,"rt");
    if(fp==NULL)
    {
        cout<<"\nTep"<<ten_tep<<"khong ton tai";
        return 0;
    }
    fscanf(fp,"%d%d",&m,&n);
    // cout << "\n" << m << " " << n; getch();
    for(i=1;i<=m;++i)

```

```

    for(j=1;j<=n;++j)
    {
        fscanf(fp,"%d",&a[i][j]);
    }
    return 1;
}

int chap_nhan(int *h, int *c, int k, int i, int j)
// 1 <= i <= m
// 1 <= j <= n
// a[i][j] >= a[h[k]][c[k]]
// (i,j) != (h[t],c[t]) t=1, ..., k
{
    int nhan=1;
    if(i<1 || i>x.m || j<1 || j>x.n) nhan=0;
    else if(x.a[i][j] < x.a[h[k]][c[k]])nhan=0;
    else
    {
        for(int t=1; t<=k; ++t)
            if(i==h[t] && j==c[t])
            {
                nhan=0; break;
            }
    }
    return nhan;
}

void di_tiep(int *h, int *c, int k)
// Cho h[1]=u, c[1]=v
// Gia su da co h[1],c[1],...,h[k],c[k] voi k>=1
// Yeu cau xac dinh h[k+1],c[k+1]
{
    int i=h[k], j=c[k];
    if(chap_nhan(h,c,k,i,j-1))
    {
        h[k+1]=i;c[k+1]=j-1;di_tiep(h,c,k+1);
    }
    if(chap_nhan(h,c,k,i,j+1))
    {
        h[k+1]=i;c[k+1]=j+1;di_tiep(h,c,k+1);
    }
    if(chap_nhan(h,c,k,i-1,j))
    {
        h[k+1]=i-1; c[k+1]=j;di_tiep(h,c,k+1);
    }
    if(chap_nhan(h,c,k,i+1,j))
    {

```

```

        h[k+1]=i+1; c[k+1]=j;di_liep(h,c,k+1);
    }
    // Nhan duoc mot duong di

if (k>kmax)
{
    kmax=k;
    for(int t=1; t<=k; t++)
    {
        hmax[t]=h[t]; cmax[t]=c[t];
    }
}

}

void main()
{
    int h[50], c[50];
    int u, v;
    clrscr();
    x.doc_lop("Ma_Lan.txt");
    cout<<"\nVi tri hang, cot xuat phat: ";
    cout << "\nNhap chi so hang tu 1 den " <<
        x.get_m() << " : ";
    cin >> u;
    cout << "\nNhap chi so cot tu 1 den " <<
        x.get_n() << " : ";
    cin >> v;
    h[1]=u; c[1]=v;
    di_liep(h,c,1);
    cout << "\nDuong di don dieu dai nhat tu (" <<
        u << "," << v << ")";
    cout << "\nCo do dai " << kmax <<
        " va di qua cac diem:";
    for(int i=1; i<=kmax; ++i)
        cout << "\n(" << hmax[i] << "," << cmax[i] << ")";
    getch();
}

```

Bài toán 2: Tìm đường đi ngắn nhất (thuật toán Dijkstra).

Mô tả cụ thể bài toán: Cho một số đỉnh. Tại mỗi đỉnh có thể có một hoặc nhiều đường đi có hướng đến các đỉnh khác (gọi đường đi này là cung). Yêu cầu: Khi cho phép chọn một đỉnh bất kỳ, hãy chỉ ra các đường đi từ nó đến các đỉnh khác (nếu có) và khi đó đường đi nào là ngắn nhất.

Ghi chú: Chương trình cần tệp DOTH1.TXT

Chương trình sau được chạy minh họa với tệp DOTH1.TXT có 5 đỉnh, 9 cung, như sau:

```

File Edit Format View Help
5 9
2 1 5
3 1 6
3 2 8
3 4 1
3 5 4
4 1 7
4 5 2
5 1 4
5 2 3

```

Khi đó, nếu chọn đỉnh xuất phát là đỉnh 2, chương trình sẽ chạy như trong hình minh họa sau:

```

D:\C_GSA-1\CT-Thong\CPP\C_DOTH1.EXE
Do thi co:
So dinh: 5 So cung: 9
Cung 2 -> 1 co do dai = 5.00
Cung 3 -> 1 co do dai = 6.00
Cung 3 -> 2 co do dai = 8.00
Cung 3 -> 4 co do dai = 1.00
Cung 3 -> 5 co do dai = 4.00
Cung 4 -> 1 co do dai = 7.00
Cung 4 -> 5 co do dai = 2.00
Cung 5 -> 1 co do dai = 4.00
Cung 5 -> 2 co do dai = 3.00
Nhap dinh xuất phát tu 1 den 5: 2
Tu dinh 2 co the di den 1 dinh khac
1 duong di ngan nhât tu dinh 2 la:
(0) Duong di toi dinh 1 co do dai: 5.0
Gach di: 2 -> 1

```

```

// C_DoThi.CPP
// Chương trình tìm đường đi ngắn nhất theo
// thuật toán: Dijkstra
#include <stdio.h>
#include <conio.h>
#include <iostream.h>
#include <stdlib.h>
int dinhtruoc[20], dinhchon[20], dachon[20];
float dai[20];
// Các mảng dinhtruoc, dinhchon, dachon, dai
// dùng trong thuật toán:
// dinhtruoc[1]=dai[1]=0

```

```

// dinhchon[1]=u, dachon[u]=1
// Voi k=2,...,n
// dinhchon[k] - chua dinh chon trong buoc k
// dinhtruoc[k] - chua dinh (da chon truoc) tu do di den
// dinhchon[k]
// dai[k] - chua do dai cua doan thang di den dinhchon[k]
// dachon[dinhchon[k]]=1 - danh dau cac dinh da chon
class DoThi
{
private:
    float a[20][20]; // Ma tran bieu dien do thi co huong:
                        // a[i,j] = 0 - khong co cung tu i den j
                        // a[i,j] > 0 - do dai cung i->j
                        // Chu y: a[i,j] <> a[j,i]
    int n; /* So dinh */
    int u; /* Dinh xuat phat (1 <= u <= n) */
public:
    void nhapdl_khoigan(char *tep);
    void min_hang(int i, int *jmin, float *gtmin);
    int thuattoan();
    void duongditoi(int k);
    void in_kq();
};

void DoThi::nhapdl_khoigan(char *tep)
{
    int i,j,k,m; float tg;
    FILE *fp;
    fp=fopen(tep,"r");
    if(fp==NULL)
    {
        printf("\nTep: %s khong co",tep); getch(); exit(0);
    }
    for(i=0;i<20;++i)
        for(j=0;j<20;++j)
            a[i][j]=0;
    fscanf(fp,"%d%d",&n,&m);
    cout << "\nDo thi co:\nSo dinh: " << n <<
        " So cung: " << m;
    for(k=0;k<m;++k)
    {
        fscanf(fp,"%d%d%f",&i,&j,&tg);
        printf("\nCung %d -> %d co do dai = %0.2f",i,j,tg);
        a[i][j]=tg;
    }
    fclose(fp);
    printf("\nNhap dinh xuat phat tu 1 den %d: ",n);
}

```

```

scanf("%d",&u);
for(i=1;i< n;++i) dachon[i]=0;
dachon[u]=1;
dai[1]=0.0;
dinhtruoc[1]=0;
dinhchon[1]=u;
}

void DoThi::min_hang(int i, int *jmin, float *gtmin)
{
    int j;
    *jmin=0; *gtmin=0.0;
    for(j=1;j<=n;++j)
        if(a[i][j]>0 && dachon[j]==0)
        {
            if((*jmin==0) || (*jmin >0 && a[i][j] < *gtmin))
            {
                *jmin = j ; *gtmin = a[i][j] ;
            }
        }
    }

}

int DoThi::thuattoan()
{
    int k;
    for(k=2;k<=n ; ++k) /* Buoc k */
    {
        // Buoc k - Xac dinh dchon_k, dtruoc_k, dai_k
        float dai_k=0.0 ;
        int dchon_k=0, dtruoc_k ;
        int m;
        for(m=1; m< k ; ++m)
            //Xet cac buoc truoc m=1,..., k-1
            {
                // Buoc m - Xuat phat tu dinhchon o buoc m
                // Xac dinh: dchon_m, dtruoc_m, dai_m
                int dchon_m, dtruoc_m ; float dai_m;
                dtruoc_m = dinhchon[m];
                min_hang(dtruoc_m,&dchon_m,&dai_m);
                dai_m += dai[m];
                if((dchon_m>0)
                    if((dchon_k==0) || (dchon_k>0 && dai_k>dai_m))
                    {
                        dchon_k=dchon_m;
                        dtruoc_k= dtruoc_m;
                        dai_k=dai_m;
                    }
                }
            }
    }
}

```

```

        }
    }
    dinhtruoc[k]=dtruoc_k; dinhchon[k]=dchon_k;
    dai[k]=dai_k; dachon[dchon_k] = 1;
    if (dinhchon[k]==0) return (k-1);
}
return n;
}

// Duong di tu dinhchon[1] den dinhchon[k], k > 2
void DoThi::duongditoi(int k)
{
    int d[20], i, j, m;
    d[1] = dinhchon[k];
    d[2] = dinhtruoc[k];
    i = 2; m = k - 1;
    while (d[i] != dinhchon[1])
    {
        while (dinhchon[m] != d[i]) --m;
        d[i+1] = dinhtruoc[m];
        ++i;
    }
    printf("\n\nDuong di toi dinh %d co do dai: %0.1f",
        dinhchon[k], dai[k]);
    printf("\nCach di: %d", d[i]);
    for (j = i - 1; j > 1; --j)
        printf(" -> %d", d[j]);
}

void DoThi::in_kq()
{
    int m;
    m = thuattoan();
    //cout << "\nm " << m; getch();
    if (m < 1)
        cout << "\nKhong co duong di tu dinh " <<
            dinhchon[1] << " den cac dinh khac";
    else
    {
        printf("\nTu dinh %d co the di den %d dinh khac",
            dinhchon[1], m - 1);
        printf("\n\n%d duong di nen nhat tu dinh %d la:",
            m - 1, dinhchon[1]);
        for (int k = 2; k <= m; ++k)
            duongditoi(k);
    }
}
}

```

```

void main()
{
    DoThi dt;
    clrscr();
    dt.nhapdl_khoigan("dothi.txt");
    dt.in_kq();
    getch();
}

```

Bài toán 3: Tháp Hà Nội.

Mô tả cụ thể bài toán: Có một tháp m tầng đang đặt tại vị trí A. Yêu cầu đặt ra là: hãy chuyển tháp tới vị trí B. Khi chuyển dùng vị trí trung gian C.

Ghi chú: Số thứ tự của tháp (1, 2, 3, ...) được đánh từ trên xuống.

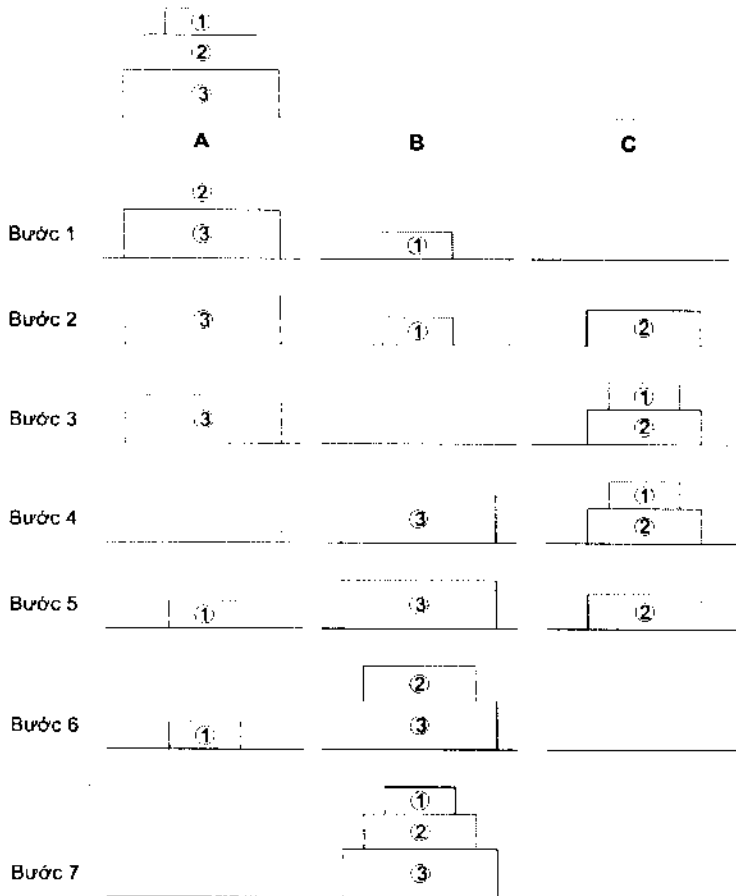
Chương trình sau được chạy minh họa với trường hợp $m = 3$:

```

D:\C_GSA-1\CT-Thong\CPP\C_THAP.EXE
Nhap so tang m = 3
Chuyen tang 1 tu A den B
Chuyen tang 2 tu A den C
Chuyen tang 1 tu B den C
Chuyen tang 3 tu A den B
Chuyen tang 1 tu C den A
Chuyen tang 2 tu C den B
Chuyen tang 1 tu A den B

```

Tương ứng với trường hợp này ($m = 3$), các bước thực hiện việc di chuyển từng tầng của tháp được minh họa chi tiết như sau:



```
// C_THAP.CPP
#include <iostream.h>
#include <conio.h>
class Thap
{
private:
    int m;//so tang
public:
    Thap(int m1=1)
    {
        m=m1;
    }
    void nhap();
    void chuyen(char vt_dau, char vt_cuoi,
                char vt_tg);
};

void Thap::chuyen(char vt_dau, char vt_cuoi,
                  char vt_tg)
```

```

{
    if(m==1)
        cout << "\nChuyen tang " << m <<
            " tu " << vt_dau << " den " << vt_cuoi;
    else
    {
        Thap(m-1).chuyen(vt_dau, vt_tg, vt_cuoi);
        cout << "\nChuyen tang " << m << " tu " <<
            vt_dau << " den " << vt_cuoi;
        Thap(m-1).chuyen(vt_tg, vt_cuoi, vt_dau);
    }
}
void Thap::nhap()
{
    cout << "\nNhap so tang m = ";
    cin >> this->m;
}
void main()
{
    Thap t;
    clrscr();
    t.nhap();
    t.chuyen('A', 'B', 'C') ;
    getch();
}

```

MỤC LỤC

CHƯƠNG 1. C++ & LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG.....	5
§1. TURBO C++ 3.0.....	5
§2. C++ VÀ C.....	6
§3. LẬP TRÌNH CẤU TRÚC & LẬP TRÌNH ĐỐI TƯỢNG.....	7
§4. MỘT SỐ MỞ RỘNG ĐƠN GIẢN CỦA C++ SO VỚI C.....	11
§5. VÀO RA TRONG C++.....	15
§6. CẤU TRÚC, HỢP VÀ KIỂU LIỆT KÊ.....	18
§7. CẤP PHÁT BỘ NHỚ.....	20
§8. CÁC HÀM TRONG C++.....	24
CHƯƠNG 2. HÀM TRONG C++.....	26
§1. BIẾN THAM CHIẾU (REFERENCE VARIABLE).....	26
§2. TRUYỀN GIÁ TRỊ CHO HÀM THEO THAM CHIẾU.....	29
§3. HÀM TRẢ VỀ CÁC THAM CHIẾU.....	33
§4. ĐỐI CÓ GIÁ TRỊ MẶC ĐỊNH.....	36
§5. CÁC HÀM TRỰC TUYẾN (INLINE).....	40
§6. ĐỊNH NGHĨA CHỒNG CÁC HÀM (OVERLOADING).....	43
§7. ĐỊNH NGHĨA CHỒNG CÁC TOÁN TỬ.....	49
§8. CÁC VÍ DỤ VỀ ĐỊNH NGHĨA CHỒNG TOÁN TỬ.....	53
§9. CÁC BÀI TOÁN VỀ MA TRẬN VÀ VÉC TƠ.....	57
CHƯƠNG 3. KHÁI NIỆM VỀ LỚP.....	63
§1. ĐỊNH NGHĨA LỚP.....	63
§2. BIẾN, MẢNG ĐỐI TƯỢNG.....	65
§3. CON TRỎ ĐỐI TƯỢNG.....	68
§4. ĐỐI CỦA PHƯƠNG THỨC, CON TRỎ THIS.....	70
§5. NÓI THÊM VỀ KIỂU PHƯƠNG THỨC VÀ KIỂU ĐỐI CỦA PHƯƠNG THỨC.....	75
§6. HÀM, HÀM BẠN.....	84
§7. PHẠM VI TRUY XUẤT.....	94
§8. CÁC PHƯƠNG THỨC TOÁN TỬ.....	95
CHƯƠNG 4. HÀM TẠO, HÀM HUỖ VÀ CÁC VẤN ĐỀ LIÊN QUAN.....	101
§1. HÀM TẠO (CONSTRUCTOR).....	101
§3. LỚP ĐA THỨC.....	107

§4. HÀM TẠO SAO CHÉP (COPY CONSTRUCTOR)	111
§5. HÀM HUY (DESTRUCTOR).....	118
§6. TOÁN TỬ GÁN.....	124
§7. PHÂN LOẠI PHƯƠNG THỨC, PHƯƠNG THỨC INLINE.....	130
§8. HÀM TẠO VÀ ĐỐI TƯỢNG THÀNH PHẦN.....	132
§9. CÁC THÀNH PHẦN TÍNH.....	138
§10. MẢNG ĐỐI TƯỢNG.....	143
§11. CẤP PHÁT BỘ NHỚ CHO ĐỐI TƯỢNG.....	146
§12. ĐỐI TƯỢNG HẰNG, PHƯƠNG THỨC HẰNG	150
§13. HÀM BẠN, LỚP BẠN.....	153
 CHƯƠNG 5. DẪN XUẤT VÀ THỪA KẾ	159
§1. SỰ DẪN XUẤT VÀ TÍNH THỪA KẾ	159
§2. HÀM TẠO, HÀM HUY ĐỐI VỚI TÍNH THỪA KẾ	165
§3. PHẠM VI TRUY NHẬP ĐẾN CÁC THÀNH PHẦN CỦA LỚP CƠ SỞ.....	169
§4. THỪA KẾ NHIỀU MỨC VÀ SỰ TRÙNG TÊN	171
§5. CÁC LỚP CƠ SỞ ẢO	174
§6. MỘT SỐ VÍ DỤ VỀ HÀM TẠO, HÀM HUY TRONG THỪA KẾ NHIỀU MỨC	176
§7. TOÁN TỬ GÁN CỦA LỚP DẪN XUẤT.....	181
§8. HÀM TẠO SAO CHÉP CỦA LỚP DẪN XUẤT.....	185
§9. PHÁT TRIỂN, HOÀN THIÊN CHƯƠNG TRÌNH	189
§10. BỔ SUNG, NÂNG CẤP CHƯƠNG TRÌNH	192
§11. TỪ KHÁI QUÁT ĐẾN CỤ THỂ.....	203
§12. TOÀN THỂ VÀ BỘ PHẬN	207
 CHƯƠNG 6. TƯƠNG ỨNG BỘI VÀ PHƯƠNG THỨC ẢO	208
§1. PHƯƠNG THỨC TÍNH	208
§2. SỰ HẠN CHẾ CỦA PHƯƠNG THỨC TÍNH.....	212
§3. PHƯƠNG THỨC ẢO VÀ TƯƠNG ỨNG BỘI.....	216
§4. SỰ TÍNH HOẠT CỦA PHƯƠNG THỨC ẢO TRONG PHÁT TRIỂN NÂNG CẤP CHƯƠNG TRÌNH	223
§5. LỚP CƠ SỞ TRỪU TƯỢNG	225
§6. SỬ DỤNG TƯƠNG ỨNG BỘI VÀ PHƯƠNG THỨC ẢO	230
§7. XỬ LÝ CÁC THUẬT TOÁN KHÁC NHAU	234
 CHƯƠNG 7. THAO TÁC TRÊN CÁC TẬP TÍN.....	239
§1. KIỂU NHẬP XUẤT NHỊ PHÂN VÀ VĂN BẢN	239

§2. GIỚI THIỆU CHUNG VỀ CÁC HÀM XỬ LÝ TỆP.....	242
§3. ĐÓNG MỞ TỆP, XOÁ VÙNG ĐỆM VÀ KIỂM TRA LỖI.....	243
§5. CÁC HÀM NHẬP XUẤT THEO KIỂU VĂN BẢN.....	249
§7. CÁC HÀM NHẬP XUẤT THEO KIỂU NHỊ PHÂN.....	254
§8. NHẬP XUẤT NGẪU NHIÊN.....	258
§9. TẠO TỆP, ĐÓNG MỞ TỆP VÀ KIỂM TRA LỖI.....	264
§11. CÁC HÀM write, read VÀ lseek.....	269
 CHƯƠNG 8. CÁC DÒNG TIN (STREAM).....	271
§1. CÁC LỚP STREAM.....	271
§2. DÒNG CẬP VÀ TOÁN TỬ NHẬP.....	272
§3. NHẬP KÍ TỰ VÀ CHUỖI KÍ TỰ TỪ BÀN PHÍM.....	274
§4. DÒNG CẬP VÀ TOÁN TỬ XUẤT.....	279
§5. CÁC PHƯƠNG THỨC ĐỊNH DẠNG.....	280
§6. CỜ ĐỊNH DẠNG.....	283
§7. CÁC BỘ PHẬN ĐỊNH DẠNG VÀ CÁC HÀM ĐỊNH DẠNG.....	287
§8. CÁC DÒNG TIN CHUẨN.....	291
§9. XUẤT RA MÁY IN.....	293
§11. GHI DỮ LIỆU LÊN TỆP.....	298
§12. ĐỌC DỮ LIỆU TỪ TỆP.....	305
§13. ĐỌC GHI ĐỒNG THỜI TRÊN TỆP.....	310
§14. XỬ LÝ LỖI.....	314
§15. NHẬP XUẤT NHỊ PHÂN.....	316
§16. ĐỌC GHI ĐỒNG THỜI THEO KIỂU NHỊ PHÂN.....	319
§17. XÂY DỰNG TOÁN TỬ NHẬP XUẤT ĐỐI TƯỢNG TRÊN TỆP.....	322
§18. HỆ THỐNG CÁC LỚP STREAM.....	326
 CHƯƠNG 9. ĐỒ HỌA.....	328
§1. KHÁI NIỆM ĐỒ HỌA.....	328
§2. KHỞI ĐỘNG HỆ ĐỒ HỌA.....	330
§3. LỖI ĐỒ HỌA.....	332
§4. MÀU VÀ MẪU.....	333
§5. VẼ VÀ TÔ MẪU.....	335
§6. CHỌN KIỂU ĐƯỜNG.....	340
§7. CỬA SỐ (VIEWPORT).....	343
§8. TÔ ĐIỂM, TÔ MIỀN.....	345
§9. XỬ LÝ VĂN BẢN TRÊN MÀN HÌNH ĐỒ HỌA.....	348
§10. CẮT HÌNH, DÁN HÌNH VÀ TẠO ẢNH CHUYỂN ĐỘNG.....	351

§11. MỘT SỐ CHƯƠNG TRÌNH ĐỒ HỌA.....	353
§12. IN ẢNH TỪ MÀN HÌNH ĐỒ HỌA.....	360
CHƯƠNG 10. TRUY NHẬP TRỰC TIẾP VÀO BỘ NHỚ.....	363
§1. CÁC HÀM TRUY NHẬP THEO ĐỊA CHỈ PHÂN ĐOẠN.....	363
§2. BỘ NHỚ MÀN HÌNH VĂN BẢN.....	364
§3. CHUYỂN ĐỔI ĐỊA CHỈ.....	366
§4. CÁC VÍ DỤ MINH HỌA.....	366
CHƯƠNG 11. KHUÔN HÌNH (TEMPLATE).....	372
§1. KHUÔN HÌNH HÀM.....	372
§2. CÁC VÍ DỤ VỀ KHUÔN HÌNH HÀM.....	373
§3. KHUÔN HÌNH LỚP.....	379
CHƯƠNG 12. MỘT SỐ CHƯƠNG TRÌNH HƯỚNG ĐỐI TƯỢNG	
TRÊN C++.....	383
§1 LỚP CỦA SỐ.....	383
§2 LỚP MENU.....	388
§3. LỚP HÌNH HỌC.....	392
§4. CÁC LỚP NGĂN XẾP VÀ HÀNG ĐỢI.....	396
§5. CÁC LỚP SẮP XẾP.....	404
§6. VÍ DỤ VỀ CÁC LỚP SẮP XẾP.....	409
PHỤ LỤC 1. THỨ TỰ ƯU TIÊN CỦA CÁC PHÉP TOÁN.....	413
PHỤ LỤC 2. CÁC TỪ KHOẢ CỦA C++.....	416
PHỤ LỤC 3. BẢNG MÃ ASCH VÀ MÃ QUÉT.....	417
PHỤ LỤC 4. HÀM VỚI ĐỐI SỐ BẤT ĐỊNH TRONG C.....	423
PHỤ LỤC 5. TÓM TẮT CÁC HÀM CỦA TURBO C THEO THỨ TỰ ABC.....	428
PHỤ LỤC 6. PHÂN TÍCH, THIẾT KẾ VÀ LẬP TRÌNH	
HƯỚNG ĐỐI TƯỢNG.....	433
§ 1. PHÂN TÍCH HƯỚNG ĐỐI TƯỢNG.....	433
§2. THIẾT KẾ HƯỚNG ĐỐI TƯỢNG.....	448
§3. LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG.....	469
PHỤ LỤC 7. MỘT SỐ CHƯƠNG TRÌNH ĐỆ QUY.....	481

NHÀ XUẤT BẢN HỒNG ĐỨC

A2, 261 Thụy Khuê, Q. Tây Hồ * ĐT : (04) 3728.1360

TT Bản thảo & bản quyền : (04) 3726.1582

Web : nxbhongduc.vn * Email : lienhe@nxbhongduc.vn

GIÁO TRÌNH C++ & LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Chịu trách nhiệm xuất bản:

Thạc sĩ HOÀNG CHÍ DŨNG

Biên tập:

THU HẰNG

Vẽ bìa:

TRỌNG KIÊN

Sửa bản in.

BIÊN TẬP.COM

PHÁT HÀNH TẠI

Nhà sách Huy Hoàng

95 Núi Trúc, Ba Đình, Hà Nội

Tel/Fax: (04) 3736.5859 - 3736.6075

Mobile: 0903.262626

Nhà sách Thành Vinh

59 Đường Trần Phú, TP. Vinh, Nghệ An

Tel/Fax: (0383) 591.167 - Mobile: 0912.109349

www.huyhoangbook.com.vn

In 1.000 cuốn, khổ 16x24 cm, tại Công ty Cổ Phần In Sao Việt

Số đăng ký KHXB: 323 – 2008 / CXB / 65 – 24 / HĐ.

In xong và nộp lưu chiểu Quý I năm 2009.

Giáo trình

C++

& Lập trình hướng đối tượng

ĐÓN ĐỌC :

1. **LẬP TRÌNH WINDOWS VỚI MFC**
GS. Phạm Văn Ất (Chủ biên) - Lê Trường Thông
2. **LẬP TRÌNH INTERNET VỚI ASP**
GS. Phạm Văn Ất (Chủ biên) - Lê Trường Thông
3. **LẬP TRÌNH INTERNET VỚI PHP**
Lê Trường Thông (Chủ biên) - Vũ Trọng Hiếu
4. **VISUAL C#.NET - HỌC TỪNG BƯỚC**
Nhóm tác giả Fbook® - Lê Trường Thông (Chủ biên)
5. **VISUAL C++.NET - HỌC TỪNG BƯỚC**
Nhóm tác giả Fbook® - Lê Trường Thông (Chủ biên)
6. **TURBO C - HỌC TỪNG BƯỚC**
Lê Trường Thông
7. **TURBO C++ - HỌC TỪNG BƯỚC**
Lê Trường Thông

12449



Nhà sách Huy Hoàng • 95 Núi Trúc, Kim Mã, Ba Đình, H
Tel/Fax: 3736 5859 - 3736 6075 • Website: www.huyhoangbook.co