LỜI NÓI ĐẦU

Lập trình cấu trúc là ph- ơng pháp tổ chức, phân chia ch- ơng trình thành các hàm, thủ tục, chúng đ- ợc dùng để xử lý dữ liệu nh- ng lại tách rời các cấu trúc dữ liệu. Thông qua các ngôn ngữ Foxpro, Pascal, C đa số những ng- ời làm Tin học đã khá quen biết với ph- ơng pháp lập trình này.

Lập trình h- ớng đối t- ợng dựa trên việc tổ chức ch- ơng trình thành các lớp. Khác với hàm và thủ tục, lớp là một đơn vị bao gồm cả dữ liệu và các ph- ơng 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. T- t- ởng lập trình h- ớng đối t- ợng đ- ợc áp dụng cho hầu hết các ngôn ngữ mới chạy trên môi tr- ờng Windows nh- Microsoft Access, Visual Basic, Visual C. Vì vậy việc nghiên cứu ph- ơng pháp lập trình mới này là rất cần thiết đối với tất cả những ng- ời quan tâm, yêu thích Tin học.

C 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 đ- ợc sử dụng rộng rãi trên nhiều loại máy tính khác nhau và đã trở thành một ngôn ngữ lập trình cấu trúc rất đ- ợc - a chuộng.

Để đ- a C vào thế giới h- ớng h- ớng đối t- ợng, năm 1980 nhà khoa học ng- ời Mỹ B. Stroustrup 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 đ- a vào tất cả các khái niệm, công cụ của lập trình h- ớng đối t- ợng mà còn đ- a vào nhiều khả năng mới mẻ cho hàm. Nh- vậy C++ là một ngôn ngữ lai cho phép tổ chức ch- ơng 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 h- ớng đối t- ợng và C++ đã trở thành ngôn ngữ h- ớng đối t- ợng nổi bật trong những năm 90.

Cuốn sách này sẽ trình bầy một cách hệ thống các khái niệm của lập trình h- ớng đối t- ợng đ- ợc cài đặt trong C++ nh- lớp, đối t- ợng, sự thừa kế, tính t- ơng ứng bội và các khả năng mới trong xây dựng, sử dụng hàm nh- : đối tham chiếu, đối mặc định, hàm trùng tên, hàm toán tử. Có một số vấn đề còn ít đ- ợc biết đến nh- cách xây dựng hàm với số đối bất định trong C cũng sẽ đ- ợc giới thiệu. Các ch- ơng từ 1 đến 10 với cách giải thích tỉ mỉ và với gần 100 ch- ơng trình minh hoạ sẽ cung cấp cho bạn đọc các khái niệm, ph- ơng pháp và kinh nghiệm lập trình h- ớng đối t- ợng trên C++. Mục lục cuối sách sẽ hệ thống ngắn gọn ph- ơng pháp phân tích, thiết kế và lập trình h- ớng đối t- ợng trên bình diện chung.

Cuốn sách gồm 10 ch- ơng và 6 phụ lục

Ch- ơng 1 h- ớng dẫn cách làm việc với phần mềm TC++ 3.0 để thử nghiệm các ch- ơng trình, trình bầy sơ l- ợc về các ph- ơng pháp lập trình và giới thiệu một số mở rộng đơn giản của C++ .

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

Ch- ơng 3 nói về một khái niệm trung tâm của lập trình h- ớng đối t- ợng là lớp gồm: Định nghĩa lớp, khai báo các biến, mảng đối t- ợng (kiểu lớp), ph- ơng thức, dùng con trỏ this trong ph- ơng thức, phạm vi truy xuất của các thành phần, các ph- ơng thức toán tử.

Ch- ơng 4 trình bầy các vấn đề tạo dựng, sao chép, huỷ bỏ các đối t- ợng và các vấn đề khác có liên quan nh- : Hàm tạo, hàm tạo sao chép, hàm huỷ, toán tử gán, cấp phát bộ nhớ cho đối t- ợng, hàm bạn, lớp bạn.

Ch- ơng 5 trình bầy một khái niệm quan trọng tạo nên khả năng mạnh của lập trình h- ớng đối t- ợng trong việc phát triển, mở rộng phần mềm, đó là khả năng thừa kế của các lớp.

Ch- ơng 6 trình bầy 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 l- ợc đồ thống nhất, đó là tính t- ơng ứng bội và ph- ơng thức ảo. Các công cụ này cho phép dễ dàng tổ chức ch- ơng trình quản lý nhiều dạng đối t- ợng khác nhau.

Ch- ơng 7 nói về việc tổ chức vào - ra trong C++. C++ đ- a 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 gắn dòng nhập xuất với thiết bị đó. Việc tổ chức vào ra theo cách nh- vậy là rất khoa học và tiện lợi vì nó có tính độc lập thiết bị.

Ch- ơng 8 trình bầy các hàm đồ hoạ sử dụng trong C và C++. Các hàm này đ- ợc sử dụng rải rác trong toàn bộ cuốn sách để xây dựng các đối t- ợng đồ hoạ.

Ch- ơng 9 trình bầy các hàm truy xuất trực tiếp vào bộ nhớ của máy tính, trong đó có bộ nhớ màn hình. Các hàm này sẽ đ- ợc sử dụng trong ch- ơng 10 để xây dựng các lớp menu và cửa sổ .

Ch- ơng 10 giới thiệu 5 ch- ơng trình t- ơng đối hoàn chỉnh nhằm minh hoạ thêm khả năng và kỹ thuật lập trình h- ớng đối t- ợng trên C++

Phụ lục 1 trình bầy các phép toán trong C++ và thứ tự - u của chúng.

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

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

Phụ lục 4 trình bầy một vấn đề quan trọng nh- ng còn ít đ- ợc 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 nh- các hàm printf và scanf của C.

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

Cuối cùng, phụ lục 6 trình bầy một cách ngắn gọn ph- ơng pháp phân tích, thiết kế và lập trình h- ớng đối t- ợng trên bình diện chung.

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

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

Tác giả

Chương 1

C++ và lập trình hướng đối tượng

Trong chương này trình bầy 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. Làm việc với TC++ 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 mầu 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("\nNhap 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("\nDien tich = %0.2f",s);
    getch();
  }
```

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

```
    Eror:  Funtion ‘sqrt’ should have a prototype
    Eror:  Funtion ‘getch’ should have a prototype
```

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

+ Đặt tên chương chường 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 và lập trình hướng 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, FOXPRO 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 toạ độ (x,y) của một dẫy 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 toạ độ dẫy điểm

+ Xây dựng 2 hàm:

Hàm nhapsl dùng để nhập toạ 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 bang: %0.2f",dmax);
    printf("\n Di qua 2 diem co chi so la %d va %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 nghiã 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 nẩy 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 sử, 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 dụng, các vấn đề của bài toán.

+ Các ngôn ngữ thuần tuý 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 hoạ 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 dẫy điểm. Các thành phần dữ liệu của lớp dẫy điểm gồm:

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

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

- Nhập toạ 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 bang: %0.2f",dmax);
    printf("\n Di qua 2 diem co chi so la %d va %d",imax,jmax);
    getch();
  }
```

§ 4. Một số mở rộng đơn giản của C++ so với C

Trong mục này trình bầy 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 dẫy 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("\n So phan tu cua day 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("\nDay sau khi sap xep\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 + ... + (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("\n So phan tu cua day N= ");
    scanf("%d",&n);
    float s=0.0;
    for (int i=1;i<=n;++i)
        s += float(i+1)/float(i) ; // Ep kieu 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 ;  // Toạ độ của điểm
              int  mau ;  // Mã mầu của điểm
            }  DIEM ;
const DIEM  d = {320, 240, 15};
```

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

```
#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("\nLoi do hoa: %s",grapherrormsg(loi));
        getch(); exit(0);
      }
```

```
    const DIEM gmh = {getmaxx()/2,getmaxy()/2,WHITE};
    putpixel(gmh.x,gmh.y,gmh.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 getmax 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 Turbo C 2.0 không cho phép dùng phép & để lấy địa chỉ các phần tử mảng thực 2 chiều. Vì vậy khi nhập một ma trân 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 hoạ 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 biet so hang va so cot cua ma tran: ") ;
    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\n Ma 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("\nco 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 dẫy 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 hoạ 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 << "\n Thi sinh " << i ;
        cout << "\n Ho 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 << "\n Ho 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 <<  "\n Ho ten: " << setw(25) << ts[i].ht;
    cout << "   Tong diem: "  << setw(5)<< ts[i].td;
  }
getch();
```

Chương trình dưới đây là một minh hoạ khác về việc sử dụng các toán tử nhập xuất và cách định dạng trong C++ . Chương trình nhập một ma trận thực cấp mxn. 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\n Ma 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] ;
   };
```

Khí 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, dsm[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 = vàng ;
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 hoạ 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 sữ 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 xẩy 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 xẩy ra lỗi sẽ gọi hàm kiểm_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 hoạ 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 hoạ

Chương trình thứ nhất minh hoạ 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 hoạ 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 rõi 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 << "]";
    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;++k)
      q[k] = new double[n];
    cout << "Khong loi";
    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 bầy 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 trương 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 on line

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 (on line) 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 ;

nghĩa chông toán tử). Sau đó có thê thay lời gọi hàm băng 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.

Ch□ơng 10

Một số ch□ơng trình h□ớng đối t□ợng trên C++

Ch- ơng này trình bầy thêm một số ch- ơng trình h- ớng đối t- ợng trên C++. Đây là các ch- ơng trình t- ơng đối phức tạp, hữu ích và sử dụng các công cụ mạnh của C++ nh- : Cách truy nhập trực tiếp bộ nhớ màn hình, kỹ thuật đồ hoạ, con trỏ void, tính kế thừa, lớp cơ sở trừu t- ợng, t- ơng ứng bội, ph- ơng thức ảo.

§ 1. Lớp cửa sổ

Ch- ơng trình gồm lớp cua_so 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 cửa sổ
int mau;     // mau = 16*mau_nen + mau_chu
int ra_mh; // Cho biết cửa sổ đã đ- ợc đ- a ra màn hình ch- a?
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
```

Ph□ơng thức gồm:

```
cua_so();
cua_so(int c,int r,byte mau_nen, byte mau_chu);
int push(int x,int y); // Đ- a 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
```

504

+ Lớp stack (dùng để quản lý một dẫy cửa sổ)

Thuộc tính gồm:

```
int max;  //Số cửa sổ cực đại có thể quản lý
int num;  //Số cửa sổ hiện có trong stack
cua_so **pcs; //Con trỏ trỏ đến vùng nhớ chứa
              //địa chỉ của các đối t- ợng cua_so
```

Ph□ơng thức gồm:

```
stack();
stack(int max_cs);
int accept(cua_so *cs,int x,int y); //Đ- a một cửa sổ
                    //vào stack, nó sẽ hiện lên màn hình
void del(); // Loại cửa sổ khỏi stack, nó sẽ bị xoá
            // khỏi màn hình
```

Nội dung ch□ơng trình:

+ Đầu tiên hiện cửa sổ thứ nhất nền GREEN chữa WHITE. Có thể soạn thảo trên đó.

+ Nếu bấm ESC kết thúc ch- ơng 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 ch- ơng 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 ch- ơng trình, nếu bấm F6 thì hiện cửa sổ tiếp theo (theo thứ tự vòng quanh: 1 -> 2 -> 3 -> 1).

Ch□ơng trình sử dụng ph- ơng pháp truy nhập trực tiếp bộ nhớ màn hình trình bầy trong ch- ơng 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 vptr=(VP)MK_FP(0xb800,0);
// Vi tri x,y tren man hinh
#define VPOS(x,y) (VP)(vptr + ((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 ph- ơng 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 t- ợng 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 t- ", "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 t-

2. Ph□ơng 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 mầu nền xác định bởi đối mau_nen và mầu 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 ph- ơng thức.

+ Sau khi thoát khỏi, ph- ơng thức trả về giá trị bằng số thứ tự (tính từ 1) của chức năng đ- ợc chọn.

Ch- ơng trình d- ới đây xây dựng lớp cmenu và minh hoạ 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);
// Vi tri x,y tren man hinh
#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 nhat", "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

Ch- ơng trình d- ới đây gồm:

+ Lớp “hinh” là lớp cơ sở trừu tượng

+ Và 3 lớp dẫn suất từ lớp “hình” là:

- Lớp “khoihop” biểu thị các khối hộp lập ph- ơng
- Lớp “duong” biểu thị các đoạn thẳng qua 2 điểm
- Lớp “tron” biểu thị các đ- ờng tròn

Ch- ơng trình minh hoạ cách dùng t- ợng ứng bội và ph- ơng thức ảo. Nội dung ch- ơng trình nh- sau:

+ Khi chạy ch- ơng trình sẽ thấy xuất hiện một khối hộp lập ph- ơng.

+ 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 đ- ờng tròn.

+ Có thể di chuyển đ- ờng tròn bằng các phím mũi tên.

+ Bấm phím Q sẽ kết thúc ch- ơng trình.

```
/*
  CT10_03.CPP
  LOP hinh hoc
  Minh hoa cach dung:
    + lop co so truu 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+a/2,y+a+a/2);
        }
  };
class duong:public  hinh
  {
    private:
      int x1,y1,x2,y2;
    public:
      duong(void):hinh()
        {
          x1=x2=y1=y1=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 duong::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);
    hinh  *h[3];
    khoihop  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]->dchuyen(10);
    closegraph();
  }
```

§ 4. Các lớp ngăn xếp và hàng đợi

Ch- ơng 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 đ- ợc chứa trong lớp s_list d- ới 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 đ- a 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 ph- ơng 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 là: Lớp stack hoạt động theo nguyên tắc LIFO (vào sau ra tr- ớc) còn lớp queue hoạt động theo nguyên tắc FIFO (vào tr- ớc ra tr- ớc) .

Ch- ơng trình sau minh hoạ cách dùng liên kết bội, ph- ơng thức ảo và con trỏ kiểu void để quản lý các kiểu dữ liệu khác nhau.

Hoạt động của ch□ơng trình nh□ sau:

+ Tr- ớc tiên lần lượt đưa địa chỉ của biến đối tượng ts1, chuỗi “HA NOI”, biến nguyên a, biến đối t- ợng ts2 và biến thực x vào ngăn xếp s1 và hàng đợi q1.

+ Thực hiện phép gán các biến đối t- ợng:
        s2 = s1 ;
        q2 = q1 ;
+ Lấy các phần tử trong ngăn xếp s2 theo trình tự ng- ợc với lúc đ- a vào.
+ Lấy các phần tử trong hàng đợi q2 theo trình tự nh- lúc đ- a vào.

```
/*
  CT10_05.CPP
  Lop vat chua (container)
  Lop danh sach moc noi
  Lop ngan xep
  Lop hang doi
  Chu y:
    1. constructor sao chep cua lop dan suat
    2. toan tu gan cua lop dan suat
    3. co the dung cac phuong thuc khac
       de viet constructor va destructor
    4. Dung con tro 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 &s1):container(s1)
  {
    copy(s1);
  }
s_list::~s_list()
  {
    this->empty();
  }
void s_list::operator=(const s_list &s1)
  {
    this->empty();
    count=s1.count;
    copy(s1);
  }
void s_list::empty()
  {
    listnode *q,*p;
    p = head; head=NULL; tail=NULL;
    while (p!=NULL)
      {
        q=p; p=p->next;
        delete q;
      }
  }
// Lop 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 d- ới 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 ch- ơng 6 thì các lớp ở đây tổng quát hơn ở chỗ:

+ Các lớp trong mục §7 ch- ơng 6 chỉ cho phép sắp xếp một dãy số nguyên theo thứ tự tăng dần.

+ Các lớp d- ới 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 t- ợng

+ 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  (*nho_hon)(void* pt1, void* pt2); // Con trỏ hàm
                                  // định nghĩa pt1 nhỏ hơn pt2
```

+ Các ph□ơng thức:

```
protected:
    void hoan_vi(int i, int j) ; // Hoán vị các phần tử thứ i và j
    void * dia_chi (int m);   // Cho địa chỉ của phần tử thứ m
public:
    virtual void sapxep(void *a1,int n,int itemsize,
              int (*ss_nho_hon)(void* ,void* )) ;  // Sắp xếp dãy
                 // n phần tử chứa trong vùng nhớ a1, mỗi phần tử
                 // có độ dài itemsize, thứ tự tăng đ- ợc quy định
```

// bởi hàm ss_nho_hon

2. Lớp select_sort dẫn xuất từ lớp sort. Lớp này sẽ thực hiện việc sắp xếp theo ph- ơng pháp chon (xem mục §7 ch- ơng 6).

+ Các ph□ơng thức:

```
public:
   virtual void sapxep(void *a1,int n,int itemsize,
            int (*ss_nho_hon)(void* ,void* )) ;  // thực hiện
      // sắp xếp theo ph- ơng pháp chọn
```

3. Lớp quick_sort dẫn xuất từ lớp sort. Lớp này sẽ thực hiện việc sắp xếp theo ph- ơng pháp quick sort (xem mục §7 ch- ơng 6)

+ Các ph□ơng thức:

```
private:
   void q_sort(int l, int r);
public:
   virtual void sapxep(void *a1,int n,int itemsize,
        int (*ss_nho_hon)(void* ,void* )) ;  // thực hiện
            // sắp xếp theo ph- ơng pháp quick sort
```

4. Lớp heap_sort dẫn xuất từ lớp sort. Lớp này sẽ thực hiện việc sắp xếp theo ph- ơng pháp heap sort (xem mục §7 ch- ơng 6).

+ Các ph□ơng 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 ph- ơng pháp heap sort
```

D- ới 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 (*nho_hon)(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 *a1,int n,int itemsize,
                                  int (*ss_nho_hon)(void*,void*))
 {
   sort::sapxep(a1,n,itemsize,ss_nho_hon);
   q_sort(1,n);
 }
class heap_sort : public sort
 {
   private:
      void shift(int i, int n);
   public:
      virtual void sapxep(void *a1,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(i), dia_chi(l)))
         hoan_vi(i,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 *a1,int n,int itemsize,
                       int (*ss_nho_hon)(void*,void*))
 {
   long i;
   sort::sapxep(a1,n,itemsize,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 bầy 2 ch- ơng trình minh hoạ cách dùng các lớp nói trên. Ch- ơng trình thứ nhất minh hoạ 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. Ch- ơng trình thứ hai minh hoạ cách dùng các lớp trong C_SORT.H để sắp xếp một dãy số nguyên theo chiều tăng và chiều giảm.

Ch□ơng trình 1

```
//CT10-08
```

```
// 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();
  }
```

Ch□ơng trình 2

```
//CT10-09
// Lop co so truu 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();
}
```

CH□ ƠNG 2

HÀM TRONG C++

Ch- ơng này trình bầy 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 tham chiếu hằng (const)

+ Đối có 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

Tr- ớc 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 đ- ợc 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 biễn x. Biến px dùng để chứa địa chỉ của một biến thực, ví dụ câu lệnh:

```
px = &x ;
```

sẽ l- u trữ địa chỉ của biễn 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 đ- ợc 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 đ- ợc 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. Nh- vậy 2 thuật ngữ trên đ- ợc hiểu nh- nhau.

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

+ Trong mọi câu lệnh, viết u hay viết r đều có ý nghĩa nh- 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 th- ờng đ- ợc 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 đ- a ra cảnh báo (warning):

```
Temporary used to initialize 's'
```

Tuy nhiên ch- ơng trình vẫn làm việc. Các câu lệnh d- ới đây vẫn thực hiện và cho kết quả nh- sau:

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

Ch- ơng trình d- ới đây minh hoạ cách dùng biến tham chiếu đến một phần tử mảng cấu trúc để nhập dữ liệu và thực hiện các phép tính trên các tr- ờng 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 << "\n Ho 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 <<  "\n Ho ten: " <<  ts[1].ht;
    cout <<  "\n Tong diem: "  <<  ts[1].td;
    getch();
```

}

1.3. Hằng tham chiếu (const)

Hằng tham chiếu đ- ợc khai báo theo mẫu:

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

Cũng giống nh- 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, nh- ng không cho phép thay đổi giá trị này.

Hằng tham chiếu th- ờng đ- ợc sử dụng làm đối của hàm để cho phép hàm sử dụng giá trị của các tham số trong lời gọi hàm, nh- ng 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.

Nh- vậy ch- ơng 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. Ph- ơng pháp này có 2 nh- ợc 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 đ- ợc 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 - u 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 ch□ơng trình minh hoạ

```
/*
Ch- ơng trình sau đ- ợc tổ chức thành 3 hàm:
Nhập dãy số double
Hoán vị 2 biến double
Sắp xếp dãy số double theo thứ tự tăng dần
Ch- ơng 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 <<"\n N= ";
    cin >> n;
    nhapds(x,n);
    sapxep(x,n);
    for (i=1;i<=n;++i)
      printf("\n%0.1lf",x[i]);
   getch();
  }

/*
Ch- ơng 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)

Ch- ơng 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 << "\nHo ten: " << setw(20) << ts.ht << setw(6) << ts.td ;
  }
void nhapsl(TS *ts,int n)
  {
    for (int i=1;i<=n;++i)
      {
        cout << "\n Thi sinh " << i ;
        cout << "\n Ho 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 ;
      }
  }
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 ;
    nhapsl(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();
  }
/*
Ch- ơng trình sau gồm các hàm:
Nhập một ma trận thực cấp mxn
In một ma trận thực d- ới dạng bảng
Tìm phần tử lớn nhất và phần  tử nhỏ nhất của dãy số th- c;
Ch- ơng 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 hamg 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
          %d",i,p[vtmax],vtmax);
          printf("\n Phan tu min= %6.1f tai cot %d", p[vtmin],vtmin);
        }
      getch();
    }
```

§ 3. 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. D- ới đây là một số ví dụ.

Ví dụ 1 trình bầy 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 bầy 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 << "\n Ho 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 <<  "\n Ho ten: " <<  ts.ht;
    cout <<  "\n Tong diem: "  <<  ts.td;
    getch();
  }
```

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

Hàm sẽ kiểm tra xem chỉ số mảng có v- ợt 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 << "\n So 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 << " (bam sai ket thuc CT) ";
        cin >> i;
```

```
        TS &h=f(i,n);
        cout <<  "\n Ho ten: " <<  h.ht;
        cout <<  "\n Tong 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 th- ờng số tham số trong lời gọi hàm phải bằng số đối của hàm. Mỗi đối sẽ đ- ợc khởi gán giá trị theo tham số t- ơng 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 t- ơng ứng trong lời gọi hàm. Khi không có tham số t- ơng ứng, đối đ- ợc khởi gán bởi giá trị mặc định.

Ví dụ hàm delay với đối số mặc định đ- ợc 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 đ- ợc 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 đ- ợc khởi gán lại cho các đối mặc định trong dòng đầu của định nghĩa hàm. Nếu vi phạm điều này thì Ch- ơng trình dịch sẽ thông báo lỗi.

+ Khi xây dựng hàm, nếu không khai báo nguyên mẫu, thì các đối mặc định đ- ợc 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 đinh

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 t- ơng 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) ;  // Đầy 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 d- ới) 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 d- ới đây trình bầy hàm hiển thị một chuỗi str trên màn hình đồ hoạ, tại vị trí (x,y) và có mầu 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(); // Luu mau hien tai
    setcolor(m);
    outtextxy(x,y,str) ;
    setcolor(mau_ht);  // Khoi phuc mau hien tai
  }
void main()
  {
    int mh=0, mode=0;
    initgraph(&mh,&mode,"");
    setbkcolor(BLUE);
    hiendc("HELLO");  // HELLO mầu đỏ giữa màn hình
    hiendc("CHUC MUNG",1,1);  // CHUC MUNG mầu đỏ tại vị
                                  // trí (1,1)
    hiendc("CHAO",1,400,YELLOW); // CHAO mầu vàng tại vị
                                      // trí (1,400)
    getch();
  }
```

Ví dụ d- ới đây trình bầy hàm tính tích phân xác định gồm 3 đối: f là hàm cần tính tích phân, a và b là các cận d- ới 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 ph- ơng), 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 << "\nTich phan tu 0 den 1 cua x*x= " << tp() ;
    cout << "\nTich phan tu 0 den 1 cua exp(x)= " << tp(exp);
    cout << "\nTich phan tu 0 den PI/2 cua sin(x) " <<
                                                  tp(sin,0,3.14/2);
    getch();
  }
```

§ 5. CÁC HÀM TRỰC TUYẾN (INLINE)

5.1. Ưu, nh□ợc điểm của hàm

Việc tổ chức ch- ơng trình thành các hàm có 2 - u điểm rõ rệt : Thứ nhất là chia ch- ơng trình thành các đơn vị độc lập, làm cho ch- ơng trình đ- ợc tổ chức một cách khoa học dễ kiểm soát dễ phát hiện lỗi, dễ phát triển, mở rộng.

Thứ hai là giảm đ- ợc kích th- ớc ch- ơng trình, vì mỗi đoạn ch- ơng trình thực hiện nhiệm vụ của hàm đ- ợc thay bằng một lời gọi hàm.

Tuy nhiên hàm cũng có nh- ợc điểm là làm chậm tốc độ ch- ơng trình do phải thực hiện một số thao tác có tính thủ tục mỗi khi gọi hàm nh- : 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 tr- ớc khi thoát khỏi hàm.

Các hàm trực tuyến trong C++ cho khả năng khắc phục đ- ợc nh- ợc điểm nói trên.

5.2. Các hàm trực tuyến

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

inline

vào tr- ớc khai báo nguyên mẫu hàm. Nếu không dùng nguyên mẫu thì viết từ khoá này tr- ớc 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 tr- ờng hợp, từ khoá inline phải xuất hiện tr- ớc các lời gọi hàm thì Trình biên dịch mới biết cần xử lý hàm theo kiểu inline.

Ví dụ hàm f trong ch- ơng 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 đ- ợc xây dựng sau lời gọi hàm thì bắt buộc phải khai báo nguyên mẫu hàm tr- ớc 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

Ch- ơng trình dịch xử lý các hàm inline nh- các macro (đ- ợc định nghĩa trong lệnh #define), nghĩa là nó sẽ thay mỗi lời gọi hàm bằng một đoạn ch- ơng trình thực hiện nhiệm vụ của hàm. Cách này làm cho ch- ơng trình dài ra, nh- ng tốc độ ch- ơng 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 t- ơng tự, tuy nhiên ng- ời ta thích dùng hàm trực tuyến hơn, vì cách này đảm bảo tính cấu trúc của ch- ơng trình, dễ sử dụng và tránh đ- ợc các sai sót lặt vặt th- ờng gặp khi dùng #define (nh- thiếu các dấu ngoặc, dấu chấm phẩy)

5.5. Khi nào thì nên dùng hàm trực tuyến

Ph- ơng án dùng hàm trực tuyến rút ngắn đ- ợc thời gian chạy máy nh- ng lại làm tăng khối l- ợng bộ nhớ ch- ơng trình (nhất là đối với các hàm trực tuyến có nhiều câu lệnh). Vì vậy chỉ nên dùng ph- ơng 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 th- ờng không xử lý theo cách inline nh- các hàm chứa biến static, hàm chứa các lệnh chu trình hoặc lệnh goto hoặc lệnh switch, hàm đệ quy. Trong tr- ờng 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 nh- Trình biên dịch thấy cần thiết (ví dụ đã có quá nhiều hàm inline làm cho bộ nhớ ch- ơng trình quá lớn)

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

Ph□ơng 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 << "\n So hinh chu hat: " ;
    cin >> n;
    for (int i=1;i<=n;++i)
      {
        cout << "\nNhap 2 canh cua hinh chu nhat thu " <<i<< ": ";
        cin >> a[i] >>  b[i] ;
        dtcvhcn(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();
  }
```

Ph□ơng án 2: Sử dụng khai báo nguyên mẫu. Khi đó từ khoá inline đặt tr- ớc nguyên mẫu.

Chú ý: Không đ- ợc đặt inline tr- ớc định nghĩa hàm. Trong ch- ơng trình d- ới đây nếu đặt inline tr- ớc định nghĩa hàm thì hậu quả nh- sau: Ch- ơng trình vẫn dịch thông, nh- ng khi chạy thì ch- ơng trình bị quẩn, không thoát đ- ợc.

```
#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 << "\n So hinh chu hat: " ;
  cin >> n;
  for (int i=1;i<=n;++i)
    {
      cout << "\nNhap 2 canh cua hinh chu nhat thu " <<i<< ": ";
      cin >> a[i] >>  b[i] ;
      dtcvhcn(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  dtcvhcn(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++.

Nh- đã biết, trong C và các ngôn ngữ khác (nh- PASCAL, FOXPRO,...) 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 nh- 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. Nh- vậy, sau khi biên dịch mỗi hàm sẽ có một tên khác nhau.

Từ đó cho thấy: các hàm đ- ợc định nghĩa trùng tên phải có tập đối khác nhau (về số l- ợng hoặc kiểu). Nếu 2 hàm hoàn toàn trùng tên và trùng đối thì Trình biên dịch sẽ không có cách nào phân biệt đ- ợc. 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, nh- ng kiểu hàm khác nhau. Hàm thứ nhất kiểu nguyên (trả về a*a), hàm thứ hai kiểu void (in giá trị a). Ch- ơng 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 l- ợng và kiểu của các tham số để gọi hàm có đúng tên và đúng bộ đối số t- ơng 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 nh thế nào

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

+ Chỉ nên định nghĩa chồng các hàm thực hiện những công việc nh- nhau nh- ng trên các đối t- ợng có kiểu khác nhau. Ví dụ trong ch- ơng trình cần xây dựng các hàm: cộng 2 ma trận vuông kiểu double, cộng 2 ma trận vuông kiểu int, cộng 2 ma trân 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 đ- ợc định nghĩa chồng. Vì nh- thế mới tránh đ- ợc 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 tinh_max đ- ợc khai báo nh- sau:

```
int tinh_max(int a, int b, int c) ;   // Max của 3 số nguyên
```

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

Vấn đề đặt ra là làm thế nào lấy đ- ợc địa chỉ của mỗi hàm. Câu trả lời nh- sau:

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

```
int  (*f1)(int , int, int );
f1 = tinh_max ;          // Lấy địa chỉ của hàm thứ nhất
double   (*f2)(double , double, double);
f2 = tinh_max ;          // Lấy địa chỉ của hàm thứ hai
int  (*f3)(int *, int  );
f3 = tinh_max ;          // Lấy địa chỉ của hàm thứ ba
double  (*f4)(double *, int  );
f4 = tinh_max ;          // Lấy địa chỉ của hàm thứ t-
```

6.6. Các ví dụ

Ví dụ 1: Ch- ơng trình giải bài toán tìm max của một dãy số nguyên và max của một dãy số thực. Trong ch- ơng 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 đ- ợc 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:

Ch- ơng 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 ch- ơng trình có 3 cặp hàm trùng tên để thực hiện 3 nhiệm vụ (nh- ng trên 2 đối t- ợng khác nhau là ma trận vuông và chữ nhật): Nhập ma trận, nhân 2 ma trận và in ma trân.

```
#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 n);
void inmt(MT a,char *ten, int n);
void nhanmt(MT a,MT b, MT c, 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 CHỒNG 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 d- % 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 t- ợng tự định nghĩa (nh- 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 ch- ơng trình C gồm các hàm nhập phân số, in phân số và thực hiện các phép cộng trừ nhân chia phân số. Ch- ơng trình sẽ nhập 5 phân số: p, q, z, u, v và tính phân số s theo công thức:

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

```
#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("\n Phan so s = ");  in(s);
    getch();
}
```

Nhận xét: Việc sử dụng các hàm để thực hiện các phép tính không đ- ợc 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 đ- ợc kết quả mong muốn hay không?

Trong C++ có thể đáp ứng đ- ợc 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 th- ờng 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 (nh- phép-) thì thứ tự đối là rất quan trọng.

Ví dụ các hàm toán tử cộng , trừ phân số đ- ợc khai báo nh- 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) đ- ợc khai báo nh- 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 trân
   } ;
MT  operator-(MT x) ;
```

7.3.3. Thân của hàm toán tử: Viết nh- thân của hàm thông th- ờng. Ví dụ hàm đổi dấu ma trận có thể đ- ợc định nghĩa nh- 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 trân
   } ;
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 nh- một hàm thông th- ờng 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 nh- 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 nh- 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 - u tiên của các phép tính vẫn tuân theo các quy tắc ban đầu của C++ . Chẳng hạn các phép * và / có thứ - u 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 ch- ơng 7).

Hàm operator<< có 2 đối kiểu ostream& và PS (Phân số). Hàm trả về giá trị kiểu ostream&. Hàm đ- ợc khai báo nh- sau:

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

T- ợng tự hàm operator>> đ- ợc khai báo nh- sau:

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

D- ới 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.

Ch- ơng trình d- ới đây có nội dung nh- ch- ơng trình trong §6.2, nh- ng 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: Ch- ơng trình đ- a 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

Ch- ơng 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();
  }
```

§ 9. CÁC BÀI TOÁN VỀ MA TRẬN VÀ VÉC TƠ

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

```
struct  MT
```

```
    {
      double a[20][20] ;  // Mang a chứa các phần tử ma trận
      int  n ;                  // Cấp ma trận
    } ;
struct VT
    {
      double b[20]; // Mang chua cac phan tu cua vec to
      int n ; // Cap vec to
    } ;
```

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 t- ơng đối quen thuộc không có gì phải bàn. Để nghịch đảo ma trận có nhiều cách, ở đây chúng ta dùng ph- ơng pháp Jordance nh- 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 đ- ợc tiến hành trên n b- ớc. Nội dung của b- ớc k (k = 1,...,n) nh- sau:

Tìm chỉ số r (k <= r <= n) sao cho

abs(x[r,k]) = max { abs(x[i,k] với i = k,...,n }

Nếu 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 tg = x[k,k] (mục đích làm cho x[k,k] = 1).

Biến đổi để cột k của x trơ 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:

(hàng i) = (hàng i) - x[i,k]*(hàng k) , với mọi i khác k

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

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

Nh- sẽ thấy trong hàm main() d- ới đây, nhờ các hàm toán tử mà câu lệnh tính v đ- ợc viết gần giống nh- công thức toán học nêu trên.

```
/* Ch- ơng 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, phep 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 tg;
    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();
  }
```

CH□ ƠNG 3

KHÁI NIỆM VỀ LỚP

Nh- đã nói ở trên, lớp là khái niệm trung tâm của lập trình h- ớng đối t- ợng, 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 (nh- cấu trúc), lớp còn chứa các thành phần hàm , còn gọi là ph- ơng thức (method) hay hàm thành viên (member function). Cũng giống nh- cấu trúc, lớp có thể xem nh- một kiểu dữ liệu. Vì vậy lớp còn gọi là kiểu đối t- ợng và lớp đ- ợc dùng để khai báo các biến, mảng đối t- ợng (nh- thể dùng kiểu int để khai báo các biến mảng nguyên). Nh- vậy từ một lớp có thể tạo ra (bằng cách khai báo) nhiều đối t- ợng (biến, mảng) khác nhau. Mỗi đối t- ợng có vùng nhớ riêng của mình. Vì vậy cũng có thể quan niệm lớp là tập hợp các đối t- ợng cùng kiểu.

Ch- ơng này sẽ trình bầy cách định nghĩa lớp, cách xây dựng ph- ơng thức, giải thích về phạm vi truy nhập, s- dụng các thành phần của lớp, cách khai báo biến, mảng cấu trúc, lời gọi tới các ph- ơng thức.

§ 1. ĐỊNH NGHĨA LỚP

1. Lớp đ- ợc định nghĩa theo mẫu:

```
class  tên_lớp
   {
     // Khai báo các thành phần dữ liệu (thuộc tính)
     // Khai báo các ph- ơng thức
   } ;
// Định nghĩa (xây dựng) các ph- ơng 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 tr- ớc (cấu trúc, hợp, lớp, ...) . Thuộc tính của lớp không thể có kiểu của chính lớp đó, nh- ng 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 ph- ơng 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 đ- ợc sử dụng bên trong lớp (trong thân của các ph- ơng thức của lớp). Các hàm không phải là ph- ơng thức của lớp không đ- ợc phép sử dụng các thành phần này.

Các thành phần public (công cộng) đ- ợc phép sử dụng ở cả bên trong và bên ngoài lớp.

3. Các thành phần dữ liệu th- ờng (nh- ng 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 ph- ơng thức th- ờng khai báo là public để chúng có thể đ- ợc gọi tới (sử dụng) từ các hàm khác trong ch- ơng trình.

5. Các ph- ơng thức có thể đ- ợc xây dựng bên ngoài hoặc bên trong định nghĩa lớp. Thông th- ờng, các ph- ơng thức ngắn đ- ợc viết bên trong định nghĩa lớp, còn các ph- ơng thức dài thì viết bên ngoài định nghĩa lớp.

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

+ Các thuộc tính của lớp A

+ Các ph- ơng thức của lớp A

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

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

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

+ Các thuộc tính của lớp gồm:

```
int  x ;   // hoành độ  (cột)
int  y ;   // tung độ    (hàng)
int  m ;  // mầu
```

+ Các ph- ơng thức:

Nhập dữ liệu một điểm

Hiển thị một điểm

Ẩn một điểm

Lớp điểm đ- ợc xây dựng nh- 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 mầu 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 ph- ơng thức (dù viết trong hay viết ngoài định nghĩa lớp) đều đ- ợc phép truy nhập đến các thuộc tính x, y và m của lớp.

+ Các ph- ơng thức viết bên trong định nghĩa lớp (nh- ph- ơng thức an()) đ- ợc viết nh- một hàm thông th- ờng.

+ Khi xây dựng các ph- ơng thức bên ngoài lớp, cần dùng thêm tên lớp và toán tử phạm vi :: đặt ngay tr- ớc tên ph- ơng phức để quy định rõ đây là ph- ơng thức của lớp nào.

§ 2. BIẾN, MẢNG ĐỐI T□ ỢNG

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

Tên_lớp danh sách đối ;

Tên_lớp danh sách mảng ;

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

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

Mỗi đối t- ợng sau khi khai báo sẽ đ- ợc 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 ph- ơng thức cho mỗi đối t- ợng. Các ph- ơng thức sẽ đ- ợc sử dụng chung cho tất cả các đối t- ợng cùng lớp. Nh- vậy về bộ nhớ đ- ợc cấp phát thì đối t- ợng giống cấu trúc. Trong tr- ơng hợp này:

sizeof(d1) = sizeof(d2) = sizeof(d3) = 3*sizeof(int) = 6

sizeof(d) = 20*6 = 120

Thuộc tính của đối t□ ợng:

Trong ví dụ trên, mỗi đối t- ợng 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 t- ợng, vì vậy không thể viết tên thuộc một cách riêng rẽ mà bao giờ cũng phải có tên đối t- ợng đi kèm, giống nh- cách viết trong cấu trúc của C hay bản ghi của PASCAL. Nói cách khác, cách viết thuộc tính của đối t- ợng nh- sau:

tên_đối_t- ợng.Tên_thuộc_tính

Với các đối t- ợng d1, d2, d3 và mảng d, có thể viết nh- sau:

```
d1.x    // Thuộc tính x của đối t- ợng d1
d2.x    // Thuộc tính x của đối t- ợng d2
d3.y    // Thuộc tính y của đối t- ợng 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 ph□ ơng thức

Cũng giống nh- hàm, một ph- ơng thức đ- ợc sử dụng thông qua lời gọi. Tuy nhiên trong lời gọi ph- ơng thức bao giờ cũng phải có tên đối t- ợng để chỉ rõ ph- ơng thức thực hiện trên các thuộc tính của đối t- ợng nào. Ví dụ lời gọi:

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

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

Câu lệnh

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

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

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

```
#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 hoanh do (cot) va tung do (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 T□ ỢNG

Con trỏ đối t- ợng dùng để chứa địa chỉ của biến, mảng đối t- ợng. Nó đ- ợc khai báo nh- 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 t- ợng d1, d2
DIEM  d[20] ;  // Khai báo mảng đối t- ợng
```

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 t- ợng và chứa địa chỉ của nó
                    // vào p3
```

Để sử dụng thuộc tính của đối t- ợng thông qua con trỏ, ta viết nh- sau:

Tên_con_trỏ->Tên_thuộc_tính

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

Nh- vậy sau khi thực hiện các câu lệnh trên thì:

p1->x và d2.x là nh- nhau

p2[i].y và d[i].y là nh- nhau

Tóm lại ta có quy tắc sau

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

Tên_đối_t- ợng.Tên_thuộc_tính

Tên_con_trỏ->Tên_thuộc_tính

Tên_mảng_đối_t- ợng[chỉ_số].Tên_thuộc_tính

Tên_con_trỏ[chỉ_số].Tên_thuộc_tính

Ch- ơng trình d- ới đây cũng sử dụng lớp DIEM (trong §1) để nhập một dãy điểm, hiển thị và ẩn các điểm vừa nhập. Ch- ơng trình dùng một con trỏ kiểu DIEM và dùng toán tử new để tạo ra một dãy đối t- ợng.

```
#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 hoanh do (cot) va tung do (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 *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();
  }
```

§ 4. ĐỐI CỦA PH□ ƠNG THỨC, CON TRỎ THIS

4.1. Con trỏ this là đối thứ nhất của ph□ơng thức

Chúng ta hãy xem lại ph- ơng thức nhapsl của lớp DIEM

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

Rõ ràng trong ph- ơng thức này chúng ta sử dụng tên các thuộc tính x, y và m một cách đơn độc. Điều này có vẻ nh- mâu thuẫn với quy tắc sử dụng thuộc tính nêu trong mục tr- ớc. Song sự thể nh- sau:

C++ sử dụng con trỏ đặc biệt **this** trong các ph- ơng thức. Các thuộc tính viết trong ph- ơng thức đ- ợc hiểu là thuộc một đối t- ợng do con trỏ this trỏ tới. Nh- vậy ph- ơng thức nhapsl() có thể viết một cách t- ờng minh nh- sau:

```
void DIEM::nhapsl()
  {
    cout << "\nNhap hoanh 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: Ph- ơng thức bao giờ cũng có ít nhất một đối là con trỏ this và nó luôn luôn là đối đầu tiên của ph- ơng thức.

4.2. Tham số ứng với đối con trỏ this

Xét một lời gọi tới ph- ơng thức nhapsl() :

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

Trong tr- ờng 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
```

Nh- vậy câu lệnh

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

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

Tham số truyền cho đối con trỏ this chính là địa chỉ của đối t- ợng đi kèm với ph- ơng thức trong lời gọi ph- ơng thức.

4.3. Các đối khác của ph□ơng thức

Ngoài đối đặc biệt this (đối này không xuất hiện một cách t- ờng minh), ph- ơng thức còn có các đối khác đ- ợc khai báo nh- trong các hàm. Đối của ph- ơng thức có thể có kiểu bất kỳ (chuẩn và ngoài chuẩn).

Ví dụ để xây dựng ph- ơng thức vẽ đ- ờng thẳng qua 2 điểm ta cần đ- a vào 3 đối: Hai đối là 2 biến kiểu DIEM, đối thứ ba kiểu nguyên xác định mã mầu. Vì đã có đối ngầm định this là đối thứ nhất, nên chỉ cần khai báo thêm 2 đối. Ph- ơng thức có thể viết nh- 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);
  }
```

Ch- ơng trình sau minh hoạ các ph- ơng thức có nhiều đối. Ta vẫn dùng lớp DIEM nh- ng có một số thay đổi:

+ Bỏ thuộc tính m (mầu)

+ Bỏ các ph- ơng thức hien và an

+Đ- a vào 4 ph- ơng 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)

Ch- ơng trình còn minh hoạ:

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

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

Nội dung ch- ơng 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 hoanh 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 ph□ơng thức và lời gọi ph□ơng thức

+ Quan sát nguyên mẫu ph- ơng thức:

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

sẽ thấy ph- ơng thức có 3 đối:

Đối thứ nhất là một đối t- ợng DIEM do this trỏ tới

Đối thứ hai là đối t- ợng DIEM d2

Đối thứ ba là biến nguyên mau

Nội dung ph- ơng thức là vẽ một đoạn thẳng đi qua các điểm *this và d2 theo mã mầu mau. Xem thân của ph- ơng sẽ thấy đ- ợc 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 tr- ơng hợp này, vai trò của this không cao lắm, vì nó đ- ợc đ- a vào chỉ cốt làm rõ đối thứ nhất. Trong thân ph- ơng thức có thể bỏ từ khoá this vẫn đ- ợc.

+ Vai trò của this trở nên quan trọng trong ph- ơng thức ve_tam_giac:

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

Ph- ơng thức này có 4 đối là:

this trỏ tới một đối t- ợng kiểu DIEM

d2 một đối t- ợng kiểu DIEM

d3 một đối t- ợng kiểu DIEM

mau một biến nguyên

Nội dung ph- ơng 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 đ- ợc vẽ nhờ sử dụng ph- ơng 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 tr- ờng này rõ ràng vai trò của this rất quan trọng. Nếu không dùng nó thì công việc trơ nên khó khăn, dài dòng và khó hiểu hơn. Chúng ta hãy so sánh 2 ph- ơng án:

Ph- ơng án dùng this trong ph- ơng 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);
  }
```

Ph- ơng án không dùng this trong ph- ơng 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 PH□ ƠNG THỨC VÀ KIỂU ĐỐI CỦA PH□ ƠNG THỨC

5.1. Kiểu ph□ơng thức

Ph- ơng 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 t- ợng, con trỏ đối t- ợng, tham chiếu đối t- ợng.

5.2. Đối của ph□ơng thức

Đối của ph- ơng thức (cũng giống nh- đối của hàm) có thể có kiểu bất kỳ:

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

+ Các kiểu ngoài chuẩn đã định nghĩa tr- ớc nh- đối t- ợng, cấu trúc, hợp, enum,... . Con trỏ hoặc tham chiếu đến các kiểu ngoài chuẩn này.

+ Kiểu đối t- ợng của chính ph- ơng thức, con trỏ hoặc tham chiếu đến kiểu đối t- ợng này.

5.3. Các ví dụ

Ví dụ 1 minh hoạ:

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

+ Ph- ơng thức có giá trị trả về kiểu đối t- ợng và con trỏ đối t- ợng.

Nội dung ch- ơng 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.

Ch- ơng trình đ- ợc tổ chức thành 2 lớp:

+ Lớp HINH_CN gồm:

- Các thuộc tính: d và r (chiều dài và chiều rộng)
- Các ph- ơng 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 t- ợng của lớp HINH_CN
```

- Các ph- ơng thức

```
    void nhapsl();         // Nhập một dãy hình chữ nhật
    HINH_CN hinh_dt_max() ; //Trả về hình chữ nhật có
                                    // diện tích max
    HINH_CN *hinh_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;  // chieu dai va chieu rong
```

```
    public:
        void nhapsl()
          {
            cout << " \nNhap chieu dai va chieu rong: " ;
            cin >> d >> r ;
          }
        void in()
          {
            cout << "\nchieu dai = " << d ;
            cout << " chieu rong= " << r;
          }
        int dien_tich()
          {
            return d*r;
          }
        int chu_vi()
          {
            return 2*(d+r);
          }
  } ;
class DAY_HINH_CN
  {
    private:
        int n; // So hinh ch nhat
        HINH_CN *h;
    public:
        void nhapsl();
        HINH_CN hinh_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 hoạ:

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

+ Ph- ơng thức có giá trị trả về kiểu đối t- ợng

+ Vai trò của con trỏ this (xem ph- ơng thức maxdt của lớp TAM_GIAC)

+ Ph- ơng thức tĩnh (xem ph- ơng thức tao_tg của lớp TAM_GIAC)

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

Ch- ơng trình đ- ợc tổ chức thành 2 lớp:

+ Lớp DIEM gồm:

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

```
    void nhapsl(); // Nhập toạ độ 3 đỉnh
    void in();        // In toạ độ 3 đỉnh
  // Tạo một đối t- ợng TAM_GIAC từ 3 đối t- ợng 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 ch- ơng trình là:
  - Ph- ơng th- c tĩnh tao_tg (sẽ giải thích bên d- ới)
  - Ph- ơng th- c maxdt

+ Thuật toán là:
  - Duyệt qua các tổ hợp 3 điểm.
  - Dùng ph- ơng thức tao_tg để lập tam giác từ 3 điểm
  - Dùng ph- ơng 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 t- ợng TAM_GIAC từ 3 đối t- ợng DIEM ta đã dùng ph- ơng 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;
  }
```

Ph- ơng thức tĩnh (sẽ nói thêm trong các mục bên d- ới) có các đặc điểm sau:

+ Nó giống ph- ơng thức thông th- ờng ở chỗ: Trong thân của nó có thể truy nhập tới các thành phần của lớp (cụ thể là lớp TAM_GIAC).

+ Nó khác ph- ơng thức thông th- ờng ở chỗ:

- Không có đối ngầm định xác định bởi con trỏ this (nh- ph- ơng thức thông th- ờng). Nh- vậy ph- ơng thức tao_tg có đúng 3 đối.

- Nó không gắn với một đối t- ợng cụ thể nào của lớp, nên trong lời gọi tới ph- ơng 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 ph- ơng thức tĩnh tao_tg bằng hàm, vì trong thân hàm không đ- ợc 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 d- ới đây ta sẽ xây dựng hàm tao_tg nh- một hàm bạn của lớp TAM_GIAC.

Chú ý 3: còn một giải pháp nữa là dùng hàm tạo (constructor) sẽ trình bầy trong các ch- ơng sau:

Ch- ơng trình d- ới đây có nội dung giống nh- ví dụ 2, nh- ng thay ph- ơng thức tĩnh tao_tg bằng hàm bạn tao_tg.

Ví dụ 3: Minh hoạ cách dùng hàm bạn. Nội dung ch- ơng trình giống nh- 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 dinh tam giac
      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 << "\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 = 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 (nh- ch- ơng trình trên) hoặc có thể khai báo bên trong và xây dựng bên ngoài định nghĩa lớp nh- sau:

```
class TAM_GIAC
   {
     private:
        DIEM d1,d2,d3; // 3 dinh tam giac
     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 ch- ơng trình, vì vậy hàm có thể đ- ợc gọi tới từ bất kỳ chỗ nào. Nh- vây trong các ph- ơng thức có thể sử dụng hàm.

+ Đối của hàm có thể là các đối t- ợng, 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 nh- 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 t- ợng d1 và d2 thuộc lớp DIEM.

+ Phạm vi sử dụng của các ph- ơng thức (public) là toàn ch- ơng trình, vì vậy trong thân hàm có thể gọi tới các ph- ơng 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 ph- ơng thức do_dai, ta có thể viết hàm tính diện tích tam giác có đỉnh là các đối t- ợng d1, d2, d3 của lớp DIEM nh- 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 ch- ơng trình tìm tam giác có diện tích lớn nhất (ở mục trên) một cách đơn giản hơn(bỏ đi lớp TAM_GIAC) nh- 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].nhapsl();
    }
  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: Ch- ơng 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. Nh- vây 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 t- ởng này đ- ợc thể hiện trong ví dụ sau.

Ví dụ 2: Ch- ơng trình dùng mảng đối t- ợng ngoài.

Chú ý: Khai báo mảng đối t- ợng phải đặt sau định nghĩa kiểu đối t- ợng (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].nhapsl();
      }
    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 nh- các hàm thông th- ờng (không dùng từ khoá friend). Mẫu viết nh- 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 nh- sau:

```
class A
   {
     private:
        // Khai báo các thuộc tính
     public:
        ...
        // Xây dựng các  hàm bạn của lớp A
        void f1(...)
          {
```

```
        ...
      }
    double  f2(...)
      {
        ...
      }
    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 t- ợng thuộc lớp này. Đây là sự khác nhau duy nhất giữa hàm bạn và hàm thông th- ờng. Chú ý rằng hàm bạn không phải là ph- ơng thức của lớp. Ph- ơng thức có một đối ẩn (ứng với con trỏ this) và lời gọi của ph- ơng thức phải gắn với một đối t- ợng nào đó (địa chỉ đối t- ợng này đ- ợc truyền cho con trỏ this). Lời gọi của hàm bạn giống nh- lời gọi của hàm thông th- ờng.

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

Ph□ơng án 1: Dùng ph- ơng 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);
```

Ph□ơng 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);
```

Ph□ơng 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;
  }
```

Ph- ơng án này không đ- ợc chấp nhận, Trình biên dịch sẽ báo lỗi vì trong thân hàm không đ- ợc quyền truy xuất đến các thuộc tính riêng (private) a, b của các đối t- ợng u, u1 và u2 thuộc lớp SP.

6.2.3. Một hàm có thể là bạn của nhiều lớp đ□ợc không? Câu trả lời là đ- ợc. Khi một hàm là bạn của nhiều lớp, thì nó có quyền truy nhập tới tất cả các thuộc tính của các đối t- ợng 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    B;  // Khai báo tr- ớc lớp A
class    B;  // Khai báo tr- ớc lớp B
class    C;  // Khai báo tr- ớc 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(...)
  {
    ...
  }
```

Ch- ơng trình sau đây minh hoạ cách dùng hàm bạn (bạn của một lớp và bạn của nhiều lớp). Ch- ơng trình đ- a 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 tich(const MT &a,const VT &x);
```

Nội dung ch- ơng 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.

```
// Ch- ơng 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];  // Toa do cua diem
      public:
          void nhapsl();
          friend void in(const VT &x);
          friend VT tich(const MT &a,const VT &x) ;
    } ;
class MT
```

```
    {
      private:
          int n;
          double a[20][20];
      public:
          friend VT tich(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 ph- ơng 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) đ- ợc xem là các thành phần private.

7.2. Các thành phần riêng của lớp chỉ đ- ợc sử dụng trong phạm vi của lớp (trong thân các ph- ơng thức của lớp). Chúng không thể đem ra sử dụng bên ngoài lớp.

+ Một thuộc tính private: Thuộc tính này (của một đối t- ợng nào đó) chỉ có thể đ- ợc sử dụng trong thân của các ph- ơng thức cùng lớp.

+ Một ph- ơng thức private: Chỉ đ- ợc sử dụng trong thân của các ph- ơng thức cùng lớp.

Ví dụ sau minh hoạ cách dùng ph- ơng thức private. Xét lớp PS (phân số) với 2 thuộc tính nguyên là t (tử) và m (mẫu). Giả sử cần xây dựng các ph- ơng thức để thực hiện các phép toán cộng trừ, nhân, chia phân số. Do các phép toán này cần dùng trong toàn bộ ch- ơng trình, nên các ph- ơng 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 ph- ơng 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 ch- ơng trình. Nh- vậy nếu một thuộc tính đ- ợc khai báo là public, thì nó có thể đ- ợc truy nhập trong thân của bất kỳ hàm nào trong ch- ơng trình.

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

Ph□ơng 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 đ- ợc.

Nhận xét: Các thuộc tính th- ờng khai báo là private để đảm bảo tính dấu kín, an toàn dữ liệu của lớp.

§ 8. CÁC PH□ ƠNG THỨC TOÁN TỬ

8.1. Cách đặt tên

Các ph- ơng thức toán tử đ- ợc xây dựng nh- các ph- ơng thức thông th- ờng, chỉ có khác cách đặt tên. Tên các ph- ơng thức toán tử (cũng giống nh- hàm toán tử) đ- ợc tạo bằng cách ghép từ khoá operator với một phép toán, ví dụ:

```
operator+
operator<<
operator>>
```

8.2. Con trỏ this

Cũng giống nh- ph- ơng thức thông th- ờng, ph- ơng thức toán tử có đối đầu tiên (đối không t- ờng minh) là con trỏ this.

8.3. Toán tử một toán hạng

Các ph- ơng thức toán tử một toán hạng: Dùng ngay con trỏ this để biểu thị toán hạng duy nhất này, nên trong ph- ơng thức sẽ không có đối t- ờng minh. Ví dụ ph- ơng thức toán tử - (đổi dấu) một đối t- ợng kiểu SP (số phức) có thể viết nh- 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 ph- ơng thức toán tử hai toán hạng: Con trỏ this ứng với toán hạng thứ nhất, nên trong ph- ơng thức chỉ cần dùng một đối t- ờng minh để biểu thị toán hạng thứ hai. Ví dụ ph- ơng thức toán tử + (cộng) hai đối t- ợng kiểu SP (số phức) có thể viết nh- 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)

Ch- ơng trình sau sẽ định nghĩa lớp DT và đ- a vào các ph- ơng thức, hàm:

+ Các thuộc tính:

int n ; // bậc đa thức

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

+ Các ph- ơng 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 ph- ơng thức chỉ số và hàm tự do F

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

p[-1] = double(n)

p[i] = a[i] , i=0, 1, ..., n

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

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

+ Nội dung ch- ơng 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 ph- ơng thức operator^)
- Tính f(x2) (bằng cách dùng hàm F)

```
// Ch- ơng trình CT3_10.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:
        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();
  }
```

CH□ ƠNG 4

HÀM TẠO, HÀM HUỶ VÀ CÁC VẤN ĐỀ LIÊN QUAN

Ch- ơng này trình bầy một số vấn đề có tính chuyên sâu hơn về lớp nh- :

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

§ 1. HÀM TẠO (CONSTRUCTOR)

1.1. Công dụng

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

1.2. Cách viết hàm tạo

1.2.1. Điểm khác của hàm tạo và các ph□ơng thức thông th□ờng

Khi viết hàm tạo cần để ý 3 sự khác biệt của hàm tạo so với các ph- ơng thức khác nh- 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 ph□ơng thức thông th□ờng

Ngoài 3 điểm khác biệt trên, hàm tạo đ- ợc viết nh- các ph- ơng thức khác:

+ Hàm tạo có thể đ- ợc 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 nh- ng khác bộ đối).

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

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

và đ- a 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 x1, y1, m1 để khởi gán cho
// x, y, m
// Đối m1 có giá trị mặc định 15 (mầu trắng)
DIEM_DH(int x1, int y1, int m1=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 x1, int y1, int m1=15) ;
        // Các ph- ơng thức khác
    } ;
// Xây dựng hàm tạo bên ngoài định nghĩa lớp
DIEM_DH:: DIEM_DH(int x1, int y1, int m1)
  {
    x=x1; y=y1; m=m1;
  }
```

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 t- ợng đồng thời khởi gán cho các thuộc tính của đối t- ợng đ- ợc 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 t- ợng có thể sử dụng các tham số để khởi gán cho các thuộc tính của biến đối t- ợng.

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

+ Câu lệnh khai báo một biến đối t- ợng sẽ gọi tới hàm tạo 1 lần

+ Câu lệnh khai báo một mảng n đối t- ợng 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 t- ợng 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 t- ợng có thể dùng các tham số để khởi gán cho các thuộc tính của đối t- ợng, 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 t- ợng 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 t□ợng hằng

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

Với kiểu int chúng ta có các hằng int, nh- 356.

Với kiểu double chúng ta có các hằng double, nh- 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 t- ợng (hay còn gọi: Đối t- ợng hằng) chúng ta phải dùng tới hàm tạo. Mẫu viết nh- sau:

Tên_lớp(danh sách tham số) ;

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

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

Chú ý: Có thể sử dụng một hằng đối t- ợng nh- một đối t- ợng. Nói cách khác, có thể dùng hằng đối t- ợng để thực hiện một ph- ơng thức, ví dụ nếu viết:

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

thì có nghĩa là thực hiện ph- ơng thức in() đối với hằng đối t- ợng.

1.6. Ví dụ minh hoạ

Ch- ơng trình sau đây minh hoạ cách xây dựng hàm tạo và cách sử dùng hàm tạo trong khai báo, trong cấp phát bộ nhớ và trong việc biểu diễn các hằng đối t- ợng.

```
//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 t- ợng DIEM_DH
        friend void in(DIEM_DH d)
          {
            cout <<"\n " << d.x << " "<< d.y<<" " << d.m ;
          }
        // Ph- ơng thức dùng để in đối t- ợng 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 (mầu 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 đối
    DIEM_DH d2(200,200,10); // Gọi tới hàm tạo có đối
    DIEM_DH *d;
    d= new DIEM_DH(300,300); // Gọi tới hàm tạo có đối
    clrscr();
    in(d1);  //Gọi hàm bạn in()
    d2.in();//Gọi ph- ơng thức in()
    in(*d); //Gọi hàm bạn in()
    DIEM_DH(2,2,2).in();//Gọi ph- ơng thức in()
    DIEM_DH t[3];  // 3 lần gọi hàm tạo không đối
    DIEM_DH *q; // Gọi hàm tạo không đối
    int n;
    cout << "\nN= ";
    cin >> n;
    q=new DIEM_DH[n+1]; // (n+1) lần gọi hàm tạo không đối
    for (int i=0;i<=n;++i)
      q[i]=DIEM_DH(300+i,200+i,8);//(n+1) lần gọi hàm tạo có đối
    for (i=0;i<=n;++i)
      q[i].in();     // Gọi ph- ơng thức in()
    for (i=0;i<=n;++i)
      DIEM_DH(300+i,200+i,8).in();// Gọi ph- ơng thức in()
    getch();
  }
```

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

Các ch- ơng trình nêu trong ch- ơng 3 đều không có hàm tạo. Vậy khi đó các đối t- ợng đ- ợc hình thành nh- thế nào ?

2.1. Nếu lớp không có hàm tạo, Ch- ơng trình dịch sẽ cung cấp một hàm tạo mặc định không đối (default). Hàm này thực chất không làm gì cả. Nh- vậy một đối t- ợng tạo ra chỉ đ- ợc cấp phát bộ nhớ, còn các thuộc tính của nó ch- a đ- ợc xác định. Chúng ta có thể kiểm chứng điều này, bằng cách chạy ch- ơng trình sau:

```
//CT4_03.CPP
```

```
// Hàm tạo mặc định
#include <conio.h>
#include <iostream.h>
class DIEM_DH
    {
      private:
          int x,y,m;
      public:
          // Phuong thuc
          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 đ- ợc phát sinh nữa. Khi đó mọi câu lệnh xây dựng đối t- ợng mới đều sẽ gọi đến một hàm tạo của lớp. Nếu không tìm thấy hàm tạo cần gọi thì Ch- ơng trình dịch sẽ báo lỗi. Điều này th- ờng xẩy ra khi chúng ta không xây dựng hàm tạo không đối, nh- ng lại sử dụng các khai báo không tham số nh- ví dụ sau:

```
#include <conio.h>
#include <iostream.h>
class DIEM_DH
    {
      private:
          int x,y,m;
      public:
          // Ph- ơng thức dùng để in đối t- ợng 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 ch- a đ- ợc 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 ph- ơng án 2, ch- ơng trình có thể sửa nh- sau:

```
#include <conio.h>
#include <iostream.h>
class DIEM_DH
    {
      private:
          int x,y,m;
      public:
          // Ph- ơng thức dùng để in đối t- ợng 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

Ch- ơng trình d- ới đây là sự cải tiến ch- ơng trình trong mục 8.5 của ch- ơng 3 bằng cách đ- a 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 t- ợng 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 t- ợng (kiểu DT) tạo ra bởi các lệnh khai báo sẽ ch- a có bộ nhớ để chứa đa thức. Nh- vậy đối t- ợng tạo ra ch- a hoàn chỉnh và ch- a dùng đ- ợc. Để có một đối t- ợng hoàn chỉnh phải qua 2 b- ớc:

+ Dùng khai báo để tạo các đối t- ợng, ví dụ:

```
DT d;
```

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

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

Quy trình này đ- ợc áp dụng trong các ph- ơng thức toán tử của ch- ơng trình trong mục 8.5 ch- ơng 3. Rõ ràng quy trình này vừa dài vừa không tiện lợi, lại hay mắc lỗi, vì ng- ời lập trình hay quên không cấp phát bộ nhớ.

Việc dùng các hàm tạo để sản sinh ra các đối t- ợng hoàn chỉnh tỏ ra tiện lợi hơn, vì tránh đ- ợc các thao tác phụ (nh- cấp phát bộ nhớ) nằm bên ngoài khai báo. Ph- ơng án dùng hàm tạo sẽ đ- ợc sử dụng trong các ph- ơng thức toán tử của ch- ơng trình d- ới đây:

+ Nội dung ch- ơng 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 ph- ơng 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 t- ợng mới, ví dụ:

```
PS  p1, p2 ;
PS  *p = new PS ;
```

+ Ta cũng có thể dùng lệnh khai báo để tạo một đối t- ợng mới từ một đối t- ợng đã tồn tại, ví dụ:

```
PS  u;
PS  v(u) ; // Tạo v theo u
```

Ý nghĩa của câu lệnh này nh- sau:

- Nếu trong lớp PS ch- a 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 t- ơng ứng của v. Nh- vậy các vùng nhớ của u và v sẽ có nội dung nh- nhau. Rõ ràng trong đa số các tr- ờng 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 t- ợng mới có nội dung nh- một đối t- ợng cho tr- ớc) 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 t- ợng mới v, sau đó gọi tới hàm tạo sao chép để khởi gán v theo u.

Ví dụ sau minh hoạ cách dùng hàm tạo sao chép mặc định:

Trong ch- ơng trình đ- a vào lớp PS (phân số):

+ Các thuộc tính gồm: t (tử số) và m (mẫu).

+ Trong lớp không có ph- ơng thức nào cả mà chỉ có 2 hàm bạn là các hàm toán tử nhập (>>) và xuất (<<).

+ Nội dung ch- ơng trình là: Dùng lệnh khai báo để tạo một đối t- ơng u (kiểu PS) có nội dung nh- đối t- ợng 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 << " -  Nhap tu va mau:  " ;
           is >> p.t >> p.m  ;
           return is;
          }
   };
```

```
void main()
  {
    PS d;
    cout << "\n Nhap 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 t□ợng** để khởi gán cho đối t- ợng mới. Hàm tạo sao chép đ- ợc viết theo mẫu:

```
Tên_lớp  (const Tên_lớp  & dt)
    {
      // Các câu lệnh dùng các thuộc tính của đối t- ợng dt
      // để khởi gán cho các thuộc tính của đối t- ợng mới
    }
```

+ **Ví dụ** có thể xây dựng hàm tạo sao chép cho lớp PS nh- 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 ch- a đáp ứng đ- ợc yêu cầu. Ví dụ lớp DT (đa thức) trong §3:

```
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);
      ....
  } ;
```

Bây giờ chúng ta hãy theo rõi xem việc dùng hàm tạo mặc định trong đoạn ch- ơng trình sau sẽ dẫn đến sai lầm nh- thế nào:

```
DT   d ;
// Tạo đối t- ợng d kiểu DT
cin  >> d ;
/* Nhập đối t- ợng d , gồm: nhập một số nguyên d- ơng và
   gán cho d.n, cấp phát vùng nhớ cho d.a, nhập các hệ số
   của đa thức và chứa vào vùng nhớ đ- ợc cấp phát
*/
DT  u(d) ;
/* Dùng hàm tạo mặc định để xây dựng đối t- ợng u theo d
    Kết quả: u.n = d.n và u.a = d.a. Nh- 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 t- ợng u giống nh- d, nh- ng độc lập với d. Nghĩa là khi d thay đổi thì u không bị ảnh h- ởng gì. Thế nh- ng mục tiêu này không đạt đ- ợc, 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 tr- ờng hợp nữa cũng dẫn đến lỗi là khi một trong 2 đối t- ợng u và d bị giải phóng (thu hồi vùng nhớ chứa đa thức) thì đối t- ợng còn lại cũng sẽ không còn vùng nhớ nữa.

Ví dụ sau sẽ minh hoạ nhận xét trên: Khi d thay đổi thì u cũng thay đổi và ng- ợc 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 << "\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();
  }
```

4.4. Ví dụ về hàm tạo sao chép

Trong ch- ơng trình trên đã chỉ rõ: Hàm tạo sao chép mặc định là ch- a thoả mãn đối với lớp DT. Vì vậy cần viết hàm tạo sao chép để xây dựng đối t- ợng mới (ví dụ u) từ một đối t- ợng 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 đ- ợc (d.n + 1) hệ số.

+ Gán các hệ số chứa trong vùng nhớ của d.a sang vùng nhớ của u.a

Nh- vậy chúng ta sẽ tạo đ- ợc đối t- ợng u có nội dung ban đầu giống nh- d, nh- ng độc lập với d.

Để đáp ứng các yêu cầu nêu trên, hàm tạo sao chép cần đ- ợc xây dựng nh- 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];
  }
```

Ch- ơng trình sau sẽ minh hoạ điều này: Sự thay đổi của d không làm ảnh h- ởng đến u và ng- ợc lại sự thay đổi của u không làm ảnh h- ởng 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;   // 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];
            }
          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 HUỶ (DESTRUCTOR)

5.1. Công dụng của hàm huỷ

Hàm huỷ là một hàm thành viên của lớp (ph- ơng thức) có chức năng ng- ợc với hàm tạo. Hàm huỷ đ- ợc gọi tr- ớc khi giải phóng (xoá bỏ) một đối t- ợng để thực hiện một số công việc có tính "dọn dẹp" trước khi đối t- ợng đ- ợc huỷ bỏ, ví dụ nh- giải phóng một vùng nhớ mà đối t- ợng đang quản lý, xoá đối t- ợng khỏi màn hình nếu nh- nó đang hiển thị, ...

Việc huỷ bỏ một đối t- ợng th- ờng xẩy ra trong 2 tr- ờng hợp sau:

+ Trong các toán tử và các hàm giải phóng bộ nhớ, nh- delete, free,...

+ Giải phóng các biến, mảng cục bộ khi thoát khỏi hàm, ph- ơng 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 đ- ợc phát sinh. Đối với nhiều lớp thì hàm huỷ mặc định là đủ, và không cần đ- a 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 nh- hàm tạo là hàm không có kiểu, không có giá trị trả về.

+ Tên hàm: Tên của hàm huỷ gồm một dấu ngã (đứng tr- ớc) 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 nh- 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 ch□ơng trình trong §3

Ch- ơng 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 t- ợng kiểu DT (cần huỷ) đang quản lý.

Chúng ta hãy phân tích các khiếm khuyết của ch- ơng trình này:

+ Khi ch- ơng trình gọi tới một ph- ơng thức toán tử để thực hiện các phép tính cộng, trừ, nhân đa thức, thì một đối t- ợng trung gian đ- ợc tạo ra. Một vùng nhớ đ- ợc cấp phát và giao cho nó (đối t- ợng trung gian) quản lý.

+ Khi thực hiện xong phép tính sẽ ra khỏi ph- ơng thức. Đối t- ợng trung gian bị xoá, tuy nhiên chỉ vùng nhớ của các thuộc tính của đối t- ợng này đ- ợc giải phóng. Còn vùng nhớ (chứa các hệ số của đa thức) mà đối t- ợng trung gian đang quản lý thì không hề bị giải phóng. Nh- vậy số vùng nhớ bị chiếm dụng vô ích sẽ tăng lên.

5.4.2. Cách khắc phục

Nh- ợc điểm trên dễ dàng khắc phục bằng cách đ- a vào lớp DT hàm huỷ viết trong 5.3 (mục trên).

5.5. Lớp hình tròn đồ hoạ

Ch- ơng trình d- ới đây gồm:

Lớp HT (hình tròn) với các thuộc tính:

```
int r;    // Bán kính
int m ; // Mầu 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 ph□ơng 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 l- u ảnh hình tròn vào vùng nhớ của pht

+ Hàm huỷ

  ~HT();

  thực hiện các việc:

  - Xoá hình tròn khỏi màn hình (nếu đang hiển thị)
  - Giải phóng bộ nhớ đã cấp cho pht

+ Ph- ơng thức

  void hien(int x, int y);

  có nhiệm vụ hiển thị hình tròn tại (x,y)

+ Ph- ơng thức

  void an();

  có nhiệm vụ làm ẩn hình tròn

Các hàm độc lập:

```
void ktdh();  //Khởi tạo đồ hoạ
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 ch- ơng 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 ktdh();
void ve_bau_troi();
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 ve_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 ph- ơng thức an().

2. Điều gì xẩy ra khi bỏ đi hàm huỷ:

+ Khi gọi hàm ht_di_dong_xuong() thì có 2 đối t- ợng kiểu HT đ- ợc tạo ra. Trong thân hàm sử dụng các đối t- ợng này để vẽ các hình tròn di chuyển xuống. Khi thoát khỏi hàm thì 2 đối t- ợng (tạo ra ở trên) đ- ợc giải phóng. Vùng nhớ của các thuộc tính của chúng bị thu hồi, nh- ng vùng nhớ cấp phát cho thuộc tính pht ch- a đ- ợc giải phóng và ảnh của 2 hình tròn (ở phía d- ới màn hình) vẫn không đ- ợc cất đi.

+ Điều t- ơng tự xẩy ra sau khi ra khỏi hàm ht_di_dong_len() : vùng nhớ cấp phát cho thuộc tính pht ch- a đ- ợc giải phóng và ảnh của 2 hình tròn (ở phía trên màn hình) vẫn không đ- ợc thu dọn.

§ 6. TOÁN TỬ GÁN

6.1. Toán tử gán mặc định

Toán tử gán (cho lớp) là một tr- ờng hợp đặc biệt so với các toán tử khác. Nếu trong lớp ch- a định nghĩa một ph- ơng 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 t- ợng của lớp, ví du:

```
HT  h1, h2(100,6);
h1 = h2 ;  // Gán h2 cho h1
```

Toán tử gán mặc định sẽ sẽ sao chép đối t- ợng nguồn (h2) vào đối t- ợng đích (h1) theo từng bit một.

Trong đa số các tr- ờng 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 ph- ơng thức toán tử gán cho lớp. Nh- ng đối với các lớp có thuộc tính con trỏ nh- 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 nh- các ph- ơng thức khác, ph- ơng thức toán tử gán dùng đối con trỏ this để biểu thị đối t- ợng đích và dùng một đối t- ờng minh để biểu thị đối t- ợng nguồn. Vì trong thân của toán tử gán không nên làm việc với bản sao của đối t- ợng nguồn, mà phải làm việc trực tiếp với đối t- ợng nguồn, nên kiểu đối t- ờng minh nhất thiết phải là kiểu tham chiếu đối t- ợng.

Ph- ơng 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 ch- ơng trình không đ- ợc phép viết câu lệnh gán liên tiếp nhiều đối t- ợng, nh- :

```
u = v = k = h ;
```

Nếu ph- ơng thức toán tử gán trả về tham chiếu của đối t- ợng nguồn, thì có thể dùng toán tử gán thể thực hiện các phép gán liên tiếp nhiều đối t- ợng.

Ví dụ đối với lớp HT (trong mục tr- ớc), có thể xây dựng toán tử gán nh- 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 t- ợng nguồn cho một đối t- ợng đích.

Nh- 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 t- ợng nguồn nh- 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 t- ợng nguồn cho nhiều đối t- ợng đích. Nh- vậy các câu lệnh sau là đ- ợc:

```
    HT  u, v, h ;
    u = v = h ;
```

6.3. Toán tử gán và hàm tạo sao chép

+ Toán tử gán không tạo ra đối t- ợng mới, chỉ thực hiện phép gán giữa 2 đối t- ợng đã tồn tại.

+ Hàm tạo sao chép đ- ợc dùng để tạo một đối t- ợng mới và gán nội dung của một đối t- ợng đã tồn tại cho đối t- ợng mới vừa tạo.

+ Nếu đã xây dựng toán tử gán mà lại dùng hàm tạo sao chép mặc định thì ch- a 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.

+ Nh- vậy đối với lớp có thuộc tính con trỏ, thì ngoài hàm tạo, cần xây dựng thêm:

- Hàm huỷ

- Hàm tạo sao chép
- Ph- ơng thức toán tử gán

Chú ý: Không phải mọi câu lệnh chứa có dấu = đều gọi đến toán tử gán. Cần phân biệt 3 tr- ờng 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 chep
3. Câu lệnh gán sẽ gọi đến toán tử gán, ví dụ:
   HT u;
   u=*h; // gọi đến ph- ơng thức toán tử gán

6.4. Ví dụ minh hoạ

Ch- ơng trình d- ới đây định nghĩa lớp HT (hình tròn) và minh hoạ:

+ Hàm tạo và hàm huỷ
+ Ph- ơng 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
// Lop do hoa
// Ham huy
// toan tu gan - tra ve tham chieu
// Ham tao sao chep
// Trong ham huy co the goi PT khac
#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) // 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 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 ph□ơng thức toán tử gán

Ch- ơng trình trên sẽ vẽ 5 hình tròn trên màn hình. Điều gì sẽ xẩy ra nếu bỏ đi ph- ơng thức toán tử gán và hàm tạo sao chép?

+ Nếu bỏ cả hai, thì chỉ xuất hiên 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 PH□ ƠNG THỨC, PH□ ƠNG THỨC INLINE

7.1. Phân loại các ph□ơng thức

Có thể chia ph- ơng thức thành các nhóm:

1. Các ph- ơng thức thông th- ờng
2. Các ph- ơng thức dùng để xây dựng và huỷ bỏ đối t- ợng gồm:
   + Hàm tạo không đối,
   + Hàm tạo có đối
   + Hàm tạo sao chép
   + Hàm huỷ

3. Các ph- ơng thức toán tử

7.2. Con trỏ this

Mọi ph- ơng thức đều dùng con trỏ this nh- đối thứ nhất (đối ẩn). Ngoài ra trong ph- ơng thức có thể đ- a vào các đối t- ờng minh đ- ợc khai báo nh- đối của hàm.

+ Với các ph- ơng thức thông th- ờng, thì đối ẩn biểu thị đối t- ợng chủ thể trong lời gọi ph- ơng thức.

+ Với các hàm tạo, thì đối ẩn biểu thị đối t- ợng mới đ- ợc hình thành.

+ Với các hàm huỷ, thì đối ẩn biểu thị đối t- ợng sắp bị huỷ bỏ.

+ Với các ph- ơng thức toán tử, thì đối ẩn biểu thị toán hạng đối t- ợng thứ nhất.

7.3. Ph□ơng thức inline.

Có 2 cách để biên dịch ph- ơng thức theo kiểu inline:

Cách 1: Xây dựng ph- ơng thức bên trong định nghĩa lớp.

Cách 2: Thêm từ khoá inline vào định nghĩa ph- ơng thức (bên ngoài định nghĩa lớp).

Chú ý là chỉ các ph- ơng thức ngắn không chứa các câu lệnh phức tạp (nh- chu trình, goto, switch, đệ quy) mới có thể trơ thành inline. Nếu có ý định biên dịch theo kiểu inline các ph- ơng thức chứa các câu lệnh phức tạp nói trên, thì Trình biên dịch sẽ báo lỗi.

Trong ch- ơng trình d- ới đây, tất cả các ph- ơng thức của lớp PS (phân số) đều là ph- ơng 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 T□ ỢNG THÀNH PHẦN

8.1. Lớp bao, lớp thành phần

Một lớp có thuộc tính là đối t- ợng 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 ph- ơng thức của lớp bao không cho phép truy nhập trực tiếp đến các thuộc tính của các đối t- ợng của các lớp thành phần.

+ Vì vậy, khi xây dựng hàm tạo của lớp bao, phải s- dụng các hàm tạo của lớp thành phần để khởi gán cho các đối t- ợng 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 t- ợng thành phần u và dùng các hàm tạo của lớp B để khởi gán cho các đối t- ợng 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 t- ợng thành phần của lớp bao, ta sử dụng mẫu:

tên_đối_t- ợng(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_t- ợng( danh sách giá trị),
...
tên_đối_t- ợng( danh sách giá trị)
    {
      // Các câu lệnh trong thân hàm tạo
    }
```

Chú ý là các dấu ngoặc sau tên đối t- ợng 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 t- ợng. Nếu danh sách giá trị là rỗng thì hàm tạo không đối sẽ đ- ợc sử dụng.

+ Các đối t- ợng 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 t- ợng không đ- ợc liệt kê trên dòng đầu hàm tạo của lớp bao, đều đ- ợc 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 t- ợng thành phần đ- ợc khởi gán nh- sau:

u đ- ợc khởi gán bằng hàm tạo không đối của lớp A

v đ- ợc khởi gán bằng hàm tạo 2 đối của lớp A

q đ- ợc khởi gán bằng hàm tạo 2 đối của lớp B

r đ- ợc khởi gán bằng hàm tạo 3 đối của lớp B

p (không có mặt) đ- ợc khởi gán bằng hàm tạo không đối của lớp B

8.4. Sử dụng các ph□ơng thức của lớp thành phần

Mặc dù lớp bao có các thành phần đối t- ợng, nh- ng trong lớp bao lại không đ- ợc phép truy nhập đến các thuộc tính của các đối t- ợng này. Vì vậy giải pháp thông th- ờng là:

+ Trong các lớp thành phần, xây dựng sẵn các ph- ơng thức để có thể lấy ra các thuộc tính của lớp.

+ Trong lớp bao dùng các ph- ơng thức của lớp thành phần để nhận các thuộc tính của các đối t- ợng thành viên cần dùng đến.

8.5. Các ví dụ

Hai ch- ơng trình d- ới đây minh hoạ 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
// Thuoc tinh doi tuong
#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 (Đ- ờng 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 đ- a vào ph- ơng thức vẽ đ- ờng tròn.

Trong lớp HTron đ- a vào ph- ơng thức vẽ và tô mầu hình tròn.

Khi xây dựng ph- ơng thức của lớp bao cần sử dụng các ph- ơng 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);
    DTron dt(100,100,80,6);
    HTron ht(300,300,150,15,300,300,4);
    dt.ve();
    ht.ve();
    getch();
    closegraph();
  }
```

§ 9. CÁC THÀNH PHẦN TĨNH

9.1. Thành phần dữ liệu tĩnh

+ Thành phần dữ liệu đ- ợc 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 đ- ợc cấp phát một vùng nhớ cố định. Nó tồn tại ngay cả khi lớp ch- a có một đối t- ợng nào cả.

+ Thành phần tĩnh là chung cho cả lớp, nó không phải là riêng của mỗi đối t- ợng. Ví dụ xét 2 đối t- ợng:

A u,v ; // Khai báo 2 đối t- ợng

thì giữa các thành phần x và ts có sự khác nhau nh- 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 ch- a khai báo

+ Để biểu thị thành phần tĩnh, ta có thể dùng tên lớp, ví du: Đối với ts thì 3 cách viết sau là t- ơng đ- ơng:

A::ts u.ts v.ts

+ Khai báo và khởi gán giá trị cho thành phần tĩnh

Thành phần tĩnh sẽ đ- ợc 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 ch- a khai báo thì thành phần tĩnh ch- a tồn tại. Ví dụ xét ch- ơng 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 ch- a khai báo, nên ch- a tồn tại. Vì vậy các câu lệnh in giá trị các thành phần này trong ph- ơng thức in là không logic. Khi dịch ch- ơng trình, sẽ nhận đ- ợc các thông báo lỗi (tại ph- ơng thức in) nh- sau:

Undefined symbol HDBH::tshd in module ...

Undefined symbol HDBH::tstienban in module ...

Có thể sửa ch- ơng trình trên bằng cách đ- a vào các lệnh khai báo các thành phần tĩnh tshd và tstienban nh- sau:

```
//CT4_14.CPP
// thanh phan tinh
// Lop HDBH (hoa don ban hang)
#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 đ- a 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 đ- ợc chọn là các thuộc tính thông th- ờng. Còn các thuộc tính tshd và tstienban là chung cho cả lớp nên chúng đ- ợc chọn là các thuộc tính tĩnh.

9.3. Ph□ơng thức tĩnh

+ Có 2 cách viết ph- ơng thức tĩnh:

Cách 1: Dùng từ khoá static đặt tr- ớc định nghĩa ph- ơng thức viết bên trong định nghĩa lớp (nh- ph- ơng th- c in() ví dụ cuối của mục 9.1).

Cách 2: Nếu ph- ơng thức xây dựng bên ngoài định nghĩa lớp, thì dùng từ khoá static đặt tr- ớc khai báo ph- ơng thức bên trong định nghĩa lớp. Chú ý không cho phép dùng từ khoá static đặt tr- ớc định nghĩa ph- ơng thức viết bên ngoài định nghĩa lóp.

+ Ph- ơng thức tĩnh là chung cho cả lớp, nó không lệ thuộc vào một đối t- ợng cụ thể, nó tồn tại ngay khi lớp ch- a có đối t- ợng nào (xem ví dụ trong mục 9.1).

+ Lời gọi ph- ơng thức tĩnh:

Có thể xuất phát từ một đối t- ợng nào đó (nh- vẫn dùng khi gọi các ph- ơng thức khác)

Có thể dùng tên lớp

Ví dụ xét lớp HDBH trong mục 9.1 và xét các câu lênh:

```
HDBH   u, v;
```

Khi đó để gọi ph- ơng thức tĩnh in() có thể dùng một trong các lệnh sau:

```
u.in();
v.in();
HDBH::in();
```

+ Vì ph- ơng thức tĩnh là độc lập với các đối t- ợng, nên không thể dùng ph- ơng thức tĩnh để xử lý dữ liệu của các đối t- ợng chủ thể trong lời gọi ph- ơng thức tĩnh. Nói cách khác không cho phép truy nhập tới các thuộc tính (tr- thuộc tính tĩnh) trong thân ph- ơng thức tĩnh. Điều đó cũng đồng nghĩa với việc không cho phép dùng con trỏ this trong ph- ơng thức tĩnh.

Ví dụ nếu lập ph- ơng thức tĩnh in() để in các thuộc tính của lớp HDBH nh- 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 ph- ơng thức tĩnh không cho phép truy nhập đến các thuộc tính tenhang và tienban.

9.4. Ví dụ minh hoạ việc dùng ph□ơng thức tĩnh

Xét bài toán quản lý hoá đơn bán hàng. Mỗi hoá đơn có 2 dữ liêu 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 l- u trữ, dùng một ph- ơng 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?

Ch- ơng trình d- ới 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 T□ ỢNG

10.1. Khai báo

Có thể dùng tên lớp để khai báo mảng đối t- ợng (giống nh- 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 t- ợng DIEM nh- 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 đ- ợc gọi 30 lần để tạo 30 phần tử mảng đối t- ợng.

10.2. Khai báo và khởi gán

Để khai báo mảng và khởi gán giá trị cho các phần tử mảng đối t- ợng, cần dùng các hàm tạo có đối theo mẫu sau:

```
Tên_lớp  tên_mảng[kích_cớ] = { 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 nhapsl();
      void ve_doan_thang(DIEM d2, int mau) ;
  };
```

Khi đó các câu lệnh khai báo d- ới 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 nh- 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 t- ợng, ta viết nh- sau:

```
Tên_mảng[chỉ số] . Tên_thuộc_tính
```

Để thực hiện ph- ơng thức đối với phần tử mảng ta viết nh- sau:

```
Tên_mảng[chỉ số] . Tên_ph- ơng_thức(danh sách tham số) ;
```

Ví dụ để vẽ đoạn thẳng nối điểm d[1] với d[2] theo mầu đỏ, ta có thể dùng ph- ơng thức ve_doan_thang nh- sau:

```
d[1].ve_doan_thang(d[2], 4);// Thực hiện ph- ơng thức đối với d[1]
```

10.4. Ví dụ

Ch- ơng trình d- ới đây đ- a 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. Ch- ơng trình minh hoạ:

+ Cách dùng mảng đối t- ợng.

+ Vai trò con trỏ this (trong ph- ơng thức hv(hoán vị)) .

+ Các hàm tạo, hàm huỷ.

+ Vai trò của toán tử gán (nếu sử dụng phép gán mặc định ch- ơng trình sẽ cho kết quả sai).

```
//CT4_15.CPP
// mang 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 t- ợng. Bộ nhớ cấp phát đ- ợc quản lý bởi một con trỏ kiểu đối t- ợng. Ví dụ sau khi đã định nghĩa lớp DIEM nh- trong mục trên, ta có thể thực hiện các lệnh cấp phát bộ nhớ nh- sau:

```
int n = 10;
DIEM  *p, *q, *r ;
p = new DIEM ;  // Cấp phát bộ nhớ cho một đối t- ợng
q = new DIEM[n] ; //Cấp phát bộ nhớ cho n đối t- ợng
r = new DIEM(200,100); // Cấp phát bộ nhớ và khởi gán cho
                       // một đối t- ợng
```

11.2. Làm việc với đối tượng thông qua con trỏ

+ Giả sử con trỏ p trỏ tới vùng nhớ của một đối t- ợng nào đó. Khi đó:

  - Để biểu thị một thành phần (thuộc tính hoặc ph- ơng thức) của đối t- ợng, ta dùng mẫu viết sau:

    p -> tên_thành_phần

  - Để biểu thị đối t- ợng, ta dùng mẫu viết sau:

    *p

+ Giả sử con trỏ q trỏ tới địa chỉ đầu vùng nhớ của một dãy đối t- ợng. Khi đó:

  - Để biểu thị một thành phần (thuộc tính hoặc ph- ơng thức) của đối t- ợng thứ i, ta dùng một trong các mẫu viết sau:

    q[i].tên_thành_phần

    (q+i)-> tên_thành_phần

  - Để biểu thị đối t- ợng 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 t- ợng để giải quyết bài toán: Nhập một danh sách thí sinh, sắp xếp danh sách theo thứ tự giảm của tổng điểm. D- ới đây sẽ đ- a ra ph- ơng án mới bằng cách dùng con trỏ và cấp phát bộ nhớ cho các đối t- ợng. Ch- ơng trình chỉ thay đổi hàm main() nh- 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

Ch- ơng trình d- ới đây định nghĩa lớp tự trỏ TS (lớp có thuộc tính kiểu *TS). Lớp này đ- ợc dùng để tổ chức danh sách móc nối. Ch- ơng 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 nh- sau:

+ Thêm thuộc tính:

```
TS *dc;  // Dùng để chứa địa chỉ của một đối t- ợng kiểu TS
```

+ Thêm các ph- ơng thức:

```
void setdc(TS *dc1) ;  // Gán dc1 cho thuộc tính dc
TS *getdc() ;          //  Nhận giá trị của dc
```

+ Ph- ơng thức nhap trong ch- ơng trình tr- ớc có kiểu void nay sửa là:

```
int nhap(int i);
```

Ph- ơng thức trả về 1 nếu họ tên nhập vào khác trống, trả về 0 nếu trái lại.

+ Bỏ đi các ph- ơng thức không dùng đến nh- : Toán tử gán, hoán vị.

```
//CT4_16.CPP
// Danh sách móc nối
// Lop TS (thi 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 *dc1)
            {
              dc=dc1;
            }
          TS *getdc()
            {
              return dc;
            }
    } ;
void TS::in()
  {
    cout << "\nHo ten: " << ht << "    Tong diem: " << td;
  }
int TS::nhap(int i)
  {
    cout << "\nNhap thi sinh " << i ;
    cout << "\nHo 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 T□ ỢNG HẰNG, PH□ ƠNG THỨC HẰNG

+ Cũng giống nh- các phần tử dữ liệu khác, một đối t- ợng có thể đ- ợc 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 t- ợng hằng
```

+ Khi khai báo cần sử dụng các hàm tạo để khởi gán giá trị cho đối t- ợng 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 ph- ơng thức có thể sử dụng cho các đối t- ợng hằng là hàm tạo và hàm huỷ. Về lý thuyết các đối t- ợng hằng không thể bị thay đổi, mà chỉ đ- ợc tạo ra hoặc huỷ bỏ đi.

Khi dùng một ph- ơng thức cho đối t- ợng hằng, thì CTBD (Ch- ơng trình biên dich) sẽ cảnh báo (warning):

Non-const function called for const object

Tuy nhiên, ch- ơng trình EXE vẫn đ- ợc tạo và khi thực hiện ch- ơng trình, thì nội dung các đối t- ợng hằng vẫn bị thay đổi. Ch- ơng trình d- ới đây sẽ minh hoạ điều này. Ch- ơng trình đ- a vào lớp PS (phân số). Ph- ơng thức toán tử ++ vẫn có thể làm thay đổi đối t- ợng 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 t- ợng hằng
    clrscr();
    p.in();
    ++p;
    p.in();
    getch();
  }
```

+ Ph□ơng thức const

Để biến một ph- ơng thức thành const ta chỉ việc viết thêm từ khoá const vào sau dòng đầu của ph- ơng thức.

Chú ý: Nếu ph- ơng thức đ- ợc khai báo bên trong và định nghĩa bên ngoài lớp, thì từ khoá const cần đ- ợc bổ sung cả trong khai báo và định nghĩa ph- ơng thức.

Trong thân ph- ơng thức const không cho phép làm thay đổi các thuộc tính của lớp. Vị vậy việc dùng ph- ơng thức const cho các đối t- ợng hằng sẽ đảm bảo giữ nguyên nội dung của các đối t- ợng hằng.

Đ- ơng nhiên các ph- ơng thức const vẫn dùng đ- ợc cho các đối t- ợng khác.

Ví dụ sau về lớp PS (phân số) minh hoạ việc dùng ph- ơng thức const.

```
// Đối t- ợng const
// Ph- ơng thức 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() 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, ch- ơng 3) của một lớp, tuy không phải là ph- ơng thức của lớp, nh- ng 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 đ□ợc khai báo là bạn của lớp B thì tất cả các ph- ơng 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 tr- ớc 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

Ch- ơng trình d- ới 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 ch- ơng trình sử dụng các ph- ơng thức trùng tên:

2 ph- ơng thức nhap():

nhập ma trận

nhập véc tơ

2 ph- ơng thức in():

in ma trận

in véc tơ

4 ph- ơng thức tich():

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 ch- ơng 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 tich(const VT &y);
          MT tich(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();
}
```

CH□ ƠNG 5

DẪN XUẤT VÀ THỪA KẾ

Có 2 khái niệm rất quan trọng đã làm nên toàn bộ thế mạnh của ph- ơng pháp lập trình h- ớng đối t- ợng đó là tính kế thừa (inheritance) và tính t- ơng ứng bội (polymorphism). Tính kế thừa cho phép các lớp đ- ợc xây dựng trên các lớp đã có. Trong ch- ơng này sẽ nói về sự thừa kế của các lớp.

§ 1. SỰ DẪN XUẤT VÀ TÍNH THỪA KẾ

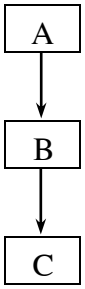
1.1. Lớp cơ sở và lớp dẫn xuất

Một lớp đ- ợc 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.

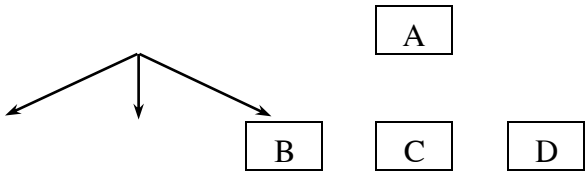
Lớp nào cũng có thể là một lớp cơ sở. Hơn thế nữa, một lớp có thể là cơ sở cho nhiều lớp dẫn xuất khác nhau. Đến l- ợt mình, lớp dẫn xuất lại có thể dùng làm cơ sở để xây dựng các lớp dân xuất khác. Ngoài ra một lớp có thể dẫn xuất từ nhiều lớp cơ sở.

D- ới đây là một số sơ đồ về quan hệ dẫn xuất của các lớp:

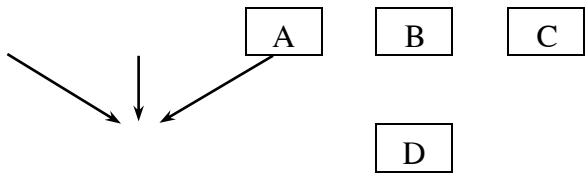
Sơ đồ 1: Lớp B dẫn xuất từ lớp A, lớp C dẫn xuất từ lớp B



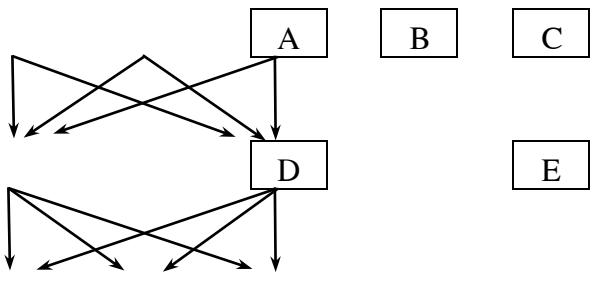
Sơ đồ 2: Lớp A là cơ sở của các lớp B, C và D

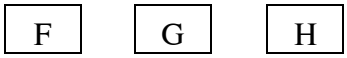


Sơ đồ 3: Lớp D dẫn xuất từ 3 lớp A, B, C



Sơ đồ 4: L- ợc đồ dẫn xuất tổng quát





Tính thừa kế: Một lớp dẫn xuất ngoài các thành phần của riêng nó, nó còn đ- ợc thừa kế tất cả các thành phần của các lớp cơ sở có liên quan. Ví dụ trong sơ đồ 1 thì lớp C đ- ợc thừa kế các thành phần của các lớp B và A. Trong sơ đồ 3 thì lớp D đ- ợc thừa kế các thành phần của các lớp A, B và C. Trong sơ đồ 4 thì lớp G đ- ợc thừa kế các thành phần của các lớp D, E, A, B và C.

1.2. Cách xây dựng lớp dân xuất

Giả sử đã định nghĩa các lớp A và B. Để xây dựng lớp C dân xuất từ A và B, ta viết nh- sau:

```
class C : public A, public B
   {
    private:
       // Khai báo các thuộc tính
    public:
       // Các ph- ơng 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 ph- ơng 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 trơ 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 đ- ợc thừa kế trong lớp dẫn xuất. Nh- 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 ph- ơng thức của C chỉ cho phép truy nhập trực tiếp tới các thuộc tính khai báo trong C.

1.5. Thừa kế ph□ơng thức

Trừ:

+ Hàm tạo
+ Hàm huỷ
+ Toán tử gán

các ph- ơng thức (public) khác của lớp cơ sở đ- ợc thừa kế trong lớp dẫn xuất.

Ví dụ: Trong ch- ơng trình d- ới đây:

+ Đầu tiên định nghĩa lớp DIEM có:
  Các thuộc tính x, y
  Hai hàm tạo
  Ph- ơng thức in()
+ Sau đó xây dựng lớp HINH_TRON dẫn xuất từ lớp DIEM, đ- a thêm:
  Thuộc tính r
  Hai hàm tạo
  Ph- ơng 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 t- ợng h kiểu HINH_TRON
  Sử dụng ph- ơng thức in() đối với h (sự thừa kế)
  Sử dụng ph- ơng 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 HUỶ 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 ph- ơng thức của lớp dẫn xuất có thể truy xuất đến các thuộc tính này. Vì vậy chúng th- ờng đ- ợc khởi gán bằng các câu lệnh gán viết trong thân hàm tạo.

2. Các thành phần kiểu đối t- ợng. Trong lớp dẫn xuất không cho phép truy nhập đến các thuộc tính của các đối t- ợng này. Vì vậy để khởi gán cho các đối t- ợng thành phần cần dùng hàm tạo của lớp t- ơng ứng. Điều này đã trình bầy trong mục §8 ch- ơng 4.

3. Các thuộc tính thừa kế từ các lớp cở sở. Trong lớp dẫn xuất không đ- ợc 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 nh- khởi gán cho các đối t- ợng thành phần, chỉ khác nhau ở chỗ: Để khởi gán cho các đối t- ợng thành phần ta dùng tên đối t- ợng 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_t- ợng_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 t- ợng của lớp dẫn xuất đ- ợc giải phóng (bị huỷ), thì các đối t- ợng thành phần và các đối t- ợng thừa kế từ các lớp cơ sở cũng bị giải phóng theo. Do đó các hàm huỷ t- ơng ứng sẽ đ- ợc gọi đến.

Nh- vậy khi xây dựng hàm huỷ của lớp dẫn xuất, chúng ta chỉ cần quan tâm đến các thuộc tính (không phải là đối t- ợng) khai báo thêm trong lớp dẫn xuất mà thôi. Ta không cần để ý đến các đối t- ợng 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, ph- ơng thức in() và hàm huỷ

+ Lớp MON_HOC gồm:

- Các thuộc tính

char *monhoc ; // Tên môn học

int st ; // Số tiết

- Hai hàm tạo, ph- ơng thức in() và hàm huỷ

+ Lớp GIAO_VIEN :

- Kế thừa từ lớp NGUOI
- Đ- a thêm các thuộc tính

char *bomon ; // Bộ môn công tác

MON_HOC mh ; // Môn học đang dậy

- Hai hàm tạo , ph- ơng thức in() và hàm huỷ

Hãy để ý cách xây dựng các hàm tạo, hàm huỷ của lớp dẫn xuất GIAO_VIEN. Trong lớp GIAO_VIEN có thể gọi tới 2 ph- ơng thức in():

GIAO_VIEN::in() // Đ- ợc xây dựng trong lớp GIAO_VIEN

NGUOI::in() // Thừa kế từ lớp NGUOI

Hãy chú ý cách gọi tới 2 ph- ơng thức in() trong ch- ơng trình d- ới đây.

```
//CT5-03
// Ham tao cua lop dan suat
#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();
  }
```

§ 3. 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 đ- ợc thừa kế tất cả các thành phần của lớp cơ sở, nh- ng trong lớp dẫn xuất không thể truy nhập tới tất cả các thành phần này. Giải pháp th- ờng dùng là sử dụng các ph- ơng thức của lớp cở sở để truy nhập đến các thuộc tính của chính lớp cơ sở đó. Cũng có thể sử dụng các giải pháp khác d- ới đây.

+ Các thành phần private của lớp cở 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 ch- ơng trình. Nh- vậy trong các lớp dẫn xuất có thể truy nhập đ- ợc 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, nh- ng hẹp hơn so với các thành phần public. Các thành phần protected của một lớp chỉ đ- ợc 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 nh- 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 cở để xây dựng lớp C. Thì trong C có thể truy nhập tới A::a1 và A::a2.

Thế nh- ng nếu sửa đổi để B dẫn xuất private từ A, thì cả A::a1 và A::a2 trơ thành private trong B, và khi đó trong C không đ- ợc phép truy nhập tới các thuộc tính A::a1 và A::a2.

Để biết t- ờng tận hơn, chúng ta hãy biên dịch ch- ơng trình:

```
//CT5-04
// Pham vi truy nhap
#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 đ- ợc 4 thông báo lỗi sau trong lớp C (tại hàm tạo có đối và ph- ơng 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 ch- ơng 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

Nh- 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 l- ợt 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.

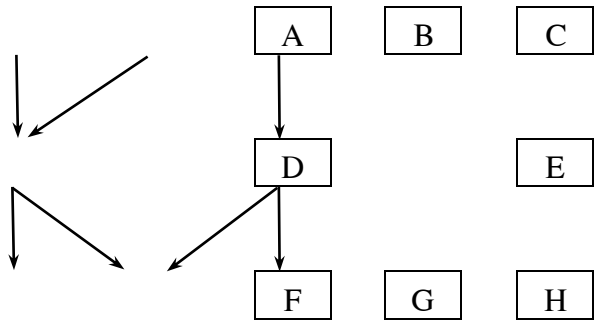
+ Sự tiếp tục theo cách trên là không hạn chế.

Sơ đồ trên chính là sự thừa kế nhiều mức. Ngoài ra chúng ta cũng đã biết:

+ Một lớp có thể đ- ợc dẫn xuất từ nhiều lớp cơ sở.

+ Một lớp có thể dùng làm cơ sở cho nhiều lớp.

Hình vẽ d- ới đây là một ví dụ về sơ đồ thừa kế khá tổng quát, thể hiện đ- ợc 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.

+ Nh- đã biết: Lớp dẫn xuất thừa kế tất cả các thành phần (thuộc tính và ph- ơng thức) của lớp cở sở, kể cả các thành phần mà lớp cơ sở đ- ợc 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

Nh- vậy các thành phần có thể sử 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 (đ- ợc thừa kế).

4.3. Sự trùng tên

Nh- 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 đ- ợc thừa kế từ các lớp D, E, A, B, C. Yêu cầu về cách đặt tên ở đây là:

+ Tên các lớp không đ- ợc trùng lặp

+ Tên các thành phần trong một lớp cũng không đ- ợc trùng lặp

+ Tên các thành phần trong các lớp khác nhau có quyền đ- ợc 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

Nh- 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 Ch- ơng trình dịch C++ dễ dàng phân biệt thành phần thuộc lớp nào. Ví dụ:

D h; // h là đối t- ợng 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 ph- ơng thức nhap() định nghĩa trong D

h.A::nhap() là ph- ơng thức nhap() định nghĩa trong A

Cách 2: Không dùng tên lớp, chỉ dùng tên thành phần. Khi đó Ch- ơng trình dịch C++ phải tự phán đoán để biết thành phần đó thuộc lớp nào. Cách phán đoán nh- sau: Tr- ớc 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 tr- ớc, các lớp quan hệ xa xét sau. Hãy chú ý tr- ờng 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 tr- ờng hợp này Ch- ơng trình dịch C++ không thể quyết định đ- ợc thành phần này thừa kế từ lớp nào và buộc phải đ- a ra một thông báo lỗi (xem ví dụ d- ới đây). Cách khắc phục: Tr- ờng hợp này phải sử dụng thêm tên lớp nh- trình bầy 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 đ- ợc 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 ph- ơng thức của các lớp A, B để xây dựng các ph- ơng thức của D

```
// Xây dựng ph- ơng thức nhap()
void D::nhap()
  {
    cout << "\n Nhap 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 ph- ơng 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 t- ợng của lớp dẫn xuất

```
    D  h ; // Khai báo h là đối t- ợng của lớp D
    h.nhap() ;  // t- ơng t- ơng 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() ;      // t- ơng đ- ơng 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 tr- ờng hợp cùng một lớp cơ sở đ- ợc 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 Ch- ơng trình dịch C++ không thể nhận biết đ- ợc thuộc tính a thừa kế thông qua B hay thông qua C và nó sẽ đ- a 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 nh- một lớp cơ sở kiểu virtual cho cả B và C. Khi đó B và C đ- ợc định nghĩa nh- sau:

```
class B : virtual public A
    {
      public:
          int  b;
    } ;
class C : virtual public A
    {
      public:
          int  c;
    } ;
```

Các lớp cơ sở ảo (virtual) sẽ đ- ợc 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. Nh- 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 HUỶ TRONG THỪA KẾ NHIỀU MỨC

Ví dụ 1. Ví dụ này minh hoạ cách xây dựng hàm tạo trong các lớp dẫn xuất. Ngoài ra còn minh hoạ cách dùng các ph- ơng thức của các lớp cơ sở trong lớp dẫn xuất và cách xử lý các đối t- ợng 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 t- ợng kiểu A.

```
//CT5-06
// Thua ke nhieu muc
// Ham tao
#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 hoạ cách xây dựng hàm huỷ trong lớp dẫn xuất. Ch- ơng trình trong ví dụ này lấy từ ch- ơng trình của ví dụ 1, sau đó đ- a thêm vào các hàm huỷ.

```
//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 goi toi cac ham huy cua cac lop 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 đ- ợc 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

+ Tr- ớc hết cần xây dựng toán tử gán cho các lớp cơ sở

+ Vấn đề mấu chốt là: Khi xây dựng toán tử gán cho lớp dẫn xuất thì làm thế nào để sử dụng đ- ợc các toán tử gán của lớp cơ sở. Cách giải quyết nh- sau:

- Xây dựng các ph- ơng thức (trong các lớp cơ sở) để nhận đ- ợc địa chỉ của đối t- ợng ẩn của lớp. Ph- ơng thức này đ- ợc viết đơn giản theo mẫu:

```
Tên_lớp *  get_DT  ( )
  {
    return this ;
  }
```

- Trong thân của toán tử gán (cho lớp dẫn xuất), sẽ dùng ph- ơng thức trên để nhận địa chỉ đối t- ợng của lớp cơ sở mà lớp dẫn xuất thừa kế. Sau đó thực hiện phép gán trên các đối t- ợng này.

Ph- ơng pháp nêu trên có thể minh hoạ một cách hình thức nh- sau: Giả sử lớp B dân xuất từ A. Để xây dựng toán tử gán cho B, thì:

1. Trong lớp A cần xây dựng toán tử gán và ph- ơng thức cho địa chỉ của đối t- ợng ẩn. Cụ thể A cần đ- ợc định nghĩa nh- sau:

```
class A
    {
      ...
      A & operator=(A& h)
        {
```

```
        //các câu lệnh để thực hiện gán trong A
      }
    // Ph- ơng thức nhận địa chỉ đối t- ợng ẩn của A
    A*   get_A()
      {
        return this;
      }
    ...
  } ;
```

2. Toán tử gán trong lớp B cần nh- sau:

```
class B : public A
  {
    ...
    B & operator=(B& h)
      {
        A  *u1, *u2;
        u1 = this->get_A();
        u2 = h.get_A();
        *u1 = *u2 ;  // Sử dụng phép gán trong A để gán các
                     // thuộc tính mà B kế thừa từ A
       //Các câu lệnh thực hiện gán các thuộc tính riêng của B
      }
    ...
  } ;
```

7.3. Ví dụ

Ch- ơng trình d- ới đây minh hoạ 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 t- ợng của lớp A.

```
//CT5-08
// Thua ke nhieu muc
// 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* getB()
          {
            return this;
          }
        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* getC()
        {
          return this;
        }
      C& operator=(C& h)
        {
          B *b1, *b2;
          b1= this->getB();
          b2= h.getB();
          *b1 = *b2;
          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)
          {
            this->u = h.u;
            C *c1,*c2;
            c1 = this->getC();
            c2 = h.getC();
            *c1 = *c2;
            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();
  }
```

§ 8. 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 đ- ợc 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

+ Tr- ớc hết cần xây dựng toán tử gán cho lớp dẫn xuất (xem §7).

+ Sau đó xây dựng hàm tạo sao chép cho lớp dẫn xuất theo mẫu:

```
Tên_lớp_dẫn_xuất (Tên_lớp_dẫn_xuất  &h )
    {
      *this = h ;
    }
```

8.3. Ví dụ

Ch- ơng trình d- ới đây minh hoạ 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 t- ợng của lớp A. Ch- ơng trình này dựa trên ch- ơng 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 dùng hàm tạo sao chép.

Để thấy rõ vai trò của hàm tạo sao chép chúng ta hãy so sánh kết quả nhận đ- ợc trong 2 tr- ờng 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& 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* getB()
            {
              return this;
            }
          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* getC()
          {
            return this;
          }
        C& operator=(C& h)
          {
            B *b1, *b2;
            b1= this->getB();
            b2= h.getB();
            *b1 = *b2;
            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(D& h) // Ham tao sao chep
      {
        *this=h;
      }
    D& operator=(D& h)
      {
        this->u = h.u;
        C *c1,*c2;
        c1 = this->getC();
        c2 = h.getC();
        *c1 = *c2;
        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;
    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 ch- ơng trình.

9.1. Ý tưởng của việc phát triển chương trình như sau: Sau khi đã xây dựng đ- ợc một lớp, ta sẽ phát triển lớp này bằng cách xây một lớp dân xuất trong đó đ- a thêm các thuộc tính và ph- ơng thức mới. Quá trình trên lại tiếp tục với lớp vừa nhận đ- ợc. Ta cũng có thể xây dựng các lớp mới có thuộc tính là đối t- ợng của các lớp cũ. Bằng cách này, sẽ nhận đ- ợc 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 ch- ơng trình vẽ một số hình phẳng. Chúng ta có thể phát triển ch- ơng trình này nh- sau:

Đầu tiên định nghĩa lớp DIEM (Điểm) gồm 2 thuộc tính x, y. Từ lớp DIEM xây dựng lớp DUONG_TRON (Đ- ơng tròn) bằng cách bổ sung 2 biến nguyên là r (bán kính) và md (mầu đ- ờng). Từ lớp DUONG_TRON xây dựng lớp HINH_TRON bằng cách thêm vào biến nguyêm mt (mầu tô). Đi theo một nhánh khác: Xây dựng lớp DOAN_THANG (Đoạn thẳng) gồm 2 đối t- ợng kiểu DIEM, và lớp TAM_GIAC gồm 3 đối t- ợng DIEM.

Ch- ơng trình d- ới đây cho phép vẽ các đ- ờng tròn, hình tròn, đoạn thẳng và hình tam giác.

Ch- ơng trình còn minh hoạ cách dùng con trỏ **this** trong lớp dẫn xuất để thực hiện các ph- ơng thức của lớp co sở. Ngoài ra còn minh hoạ cách dùng toán tử chỉ số [] để nhận các toạ độ x, y từ các đối t- ợng của lớp DIEM.

```
//CT5-10
// Phat trien chuong trinh
#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 md1)
        {
          d1=t1; d2 = t2; md=md1;
        }
      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 md1,int mt1)
        {
          d1=t1; d2=t2; d3 = t3; md=md1;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 ch- ơng trình.

10.1. Ý tưởng của việc nâng cấp chương trình như sau: Giả sử đã có một ch- ơng 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 ch- ơng 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    sotiet       ;   // Số tiết
    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 getsm() ;  // 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
  } ;
```

Ch- ơng 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 dậy.
3. In danh sách giáo viên.

Nội dung ch- ơng trình nh- 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 h- ớng 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 ch- ơng 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 nh- sau:

1. Bổ sung trong lớp GV ph- ơng thức:

```
GV*  getGV()
  {
    return this;
  }
```

Ph- ơng 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 đ- ợc 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) nh- 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 h- ớng dẫn
        LV  *lv;  // Danh sách luận văn
    public:
        GV2(); // Hàm tạo
        ~GV2() ; // Hàm huỷ
        GV2& operator=(GV2& g); // Gán
        void nhap(); // Nhập
        void xuat(); // Xuất
  } ;
```

Ch- ơng trình nâng cấp nh- sau:

```
//CT5-12B
// Nang cap chuong trinh
// CT nang cap
#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;
        }
  } ;
// Bo sung phuong thuc getGV cho lop GV
// dung de xay dung toan tu gan cho lop GV2
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;
        }
      // 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 th- ờng dùng để thiết kế các bài toán theo h- ớng từ khái quát đến cụ thể, từ chung đến riêng. Đầu tiên đ- a ra các lớp để mô tả những đối t- ợng chung, sau đó dẫn xuất tới các đối t- ợng ngày một cụ thể hơn.

Một tr- ờng hợp khác cũng th- ờng 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 đ- ợc mô tả bằng một lớp dẫn xuất từ lớp cơ sở này.

Sau đây là một số ví dụ minh hoạ:

Ví dụ 1 (minh hoạ t- t- ởng đi từ khái quát đến cụ thể) : Giả sử cần quản lý sinh viên của một tr- ờng 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 t- ợng sinh viên cụ thể hơn, ví dụ: SV Tin, SV Toán, SV Luật, SV Du lịch, ...

Các bài toán kiểu nh- vậy rất th- ờng gặp trong thực tế.

Ví dụ 2 (minh hoạ phần chung của nhiều thực thể). Giả sử cần xây dựng phần mềm để thực hiện các phép tính về ma trân 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.

D- ới đây là ch- ơng trình thực hiện các phép toán ma trận, véc tơ. Ch- ơng trình đ- ợc tổ chức thành 3 lớp:

Lớp CAP (Cấp ma trận, véc tơ) gồm một thành phần tĩnh n và ph- ơng thức nhập n.

Lớp VT (Véc tơ) có một thuộc tính là mảng một chiều (chứa các phần tử của véc tơ) và các ph- ơng thức nhập, xuất.

Lớp MT (Ma trận) có một thuộc tính là mảng 2 chiều (chứa các phần tử của ma trận) , các ph- ơng thức nhập, xuất và nhân. Lớp MT là bạn của lớp VT.

Ch- ơng 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<<"\nNhap ma tran A:";
  a.nhap();
  cout<<"\n\nNhap Vec to X:\n";
  x.nhap();
  y = a*x;
  cout<<"\n\nMa tran A";
  a.xuat();
  cout<<"\n\nVec to X";
  x.xuat();
  cout<<"\n\nVec to Y=AX";
  y.xuat();
  getch();
}
```

§ 12. TOÀN THỂ VÀ BỘ PHẬN

Thông th- ờng khi xem xét, giải quyết một bài toán, ta th- ờng 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 đ- ợc dẫn xuất từ các lớp nói trên.

Xét một thí dụ đơn giản là bài toán quản lý th- 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.

CH□ ƠNG 6

T□ƠNG ỨNG BỘI VÀ PH□ƠNG THỨC ẢO

T- ơng ứng bội và ph- ơng thức ảo là công cụ mạnh của C++ cho phép tổ chức quản lý các đối t- ợng khác nhau theo cùng một l- ợc đồ. Một khái niệm khác liên quan là: lớp cơ sở trừu t- ợng. Ch- ơng này sẽ trình bầy cách sử dụng các công cụ trên để xây dựng ch- ơng trình quản lý nhiều đối t- ợng khác nhau theo một l- ợc đồ thống nhất.

§ 1. PH□ ƠNG THỨC TĨNH

1.1. Lời gọi tới ph□ơng thức tĩnh

Nh- đã biết một lớp dẫn xuất đ- ợc thừa kế các ph- ơng 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 đ- ợc thừa kế các ph- ơng thức của A và B. Các ph- ơng thức mà chúng ta vẫn nói là các ph- ơng thức tĩnh. Để tìm hiểu thêm về cách gọi tới các ph- ơng thức tĩnh, ta xét ví dụ về các lớp A, B và C nh- 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 ph- ơng thức của A và B. Do đó một đối t- ợng của C sẽ có tới 3 ph- ơng thức xuat. Hãy theo rõi các câu lệnh sau:

```
C  h ; //  h là đối t- ợng kiểu C
h.xuat() ; // Gọi tới ph- ơng thức h.D::xuat()
h.B::xuat() ; // Gọi tới ph- ơng thức h.B::xuat()
h.A::xuat() ; // Gọi tới ph- ơng thức h.A::xuat()
```

Các lời gọi ph- ơng thức trong ví dụ trên đều xuất phát từ đối t- ợng h và mọi lời gọi đều xác định rõ ph- ơng thức cần gọi.

Bây giờ chúng ta hãy xét các lời gọi không phải từ một biến đối t- ợng 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 t- ợng kiểu A
B   b;    // b là đối t- ợng kiểu B
C   c;    // c là đối t- ợng 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 t- ợng của lớp dẫn xuất.

Nh- vậy cả 3 phép gán sau đều hợp lệ:

```
p =  &a ;
q =  &b ;
r =   &c ;
```

Chúng ta tiếp tục xét các lời gọi ph- ơng thức từ các con trỏ p, q, r:

```
p->xuat();
q->xuat();
r->xuat();
```

và hãy lý giải xem ph- ơng thức nào (trong các ph- ơng thức A::xuat, B::xuat và C::xuat) đ- ợc gọi. Câu trả lời nh- sau:

Cả 3 câu lệnh trên đều gọi tới ph- ơng thức A::xuat() , vì các con trỏ p, q và r đều có kiểu A.

Nh- vậy có thể tóm l- ợc cách thức gọi các ph- ơng thức tĩnh nh- sau:

Quy tắc gọi ph□ơng thức tĩnh: Lời gọi tới ph- ơng thức tĩnh bao giờ cũng xác định rõ ph- ơng thức nào (trong số các ph- ơng thức trùng tên của các lớp có quan hệ thừa kế) đ- ợc gọi:

1. Nếu lời gọi xuất phát từ một đối t- ợng của lớp nào, thì ph- ơng thức của lớp đó sẽ đ- ợc gọi.

2. Nếu lời gọi xuất phát từ một con trỏ kiểu lớp nào, thì ph- ơng thức của lớp đó sẽ đ- ợc gọi bất kể con trỏ chứa địa chỉ của đối t- ợng 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 ph- ơng thức xuat(). Xét hàm:

```
void hien(A *p)
  {
    p->xuat();
  }
```

Không cần biết tới địa chỉ của đối t- ợng nào sẽ truyền cho đối con trỏ p, lời gọi trong hàm luôn luôn gọi tới ph- ơng thức A::xuat() vì con trỏ p kiểu A. Nh- vậy bốn câu lệnh:

```
hien(&a);
hien(&b);
hien(&c);
hien(&d);
```

trong hàm main (của ch- ơng trình d- ới đây) đều gọi tới A::xuat().

```
//CT6-01
// Phuong thuc tinh
#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;
            }
          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 PH□ ƠNG THỨC TĨNH

Ví dụ sau cho thấy sự hạn chế của ph- ơng thức tĩnh trong việc sử dụng tính thừa kế để phát triển ch- ơng trình.

Giả sử cần xây dựng ch- ơng trình quản lý thí sinh. Mỗi thí sinh đ- a vào ba thuộc tính: Họ tên, số báo danh và tổng điểm. Ch- ơng 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). Ch- ơng trình d- ới đây sử dụng lớp TS (Thí sinh) đáp ứng đ- ợc yêu cầu đặt ra.

```
//CT6-02
// Han che phuong thuc tinh
// Lop 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 << "\nHo ten: " ;
              fflush(stdin); gets(ht);
              cout << "So bao danh: " ;
              cin >> sobd;
              cout << "Tong diem: " ;
              cin >> td;
            }
          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();
            }
    } ;
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 tr- ờng muốn quản lý thêm địa chỉ của thí sinh. Vì sự thay đổi ở đây là không nhiều, nên chúng ta không đả động đến lớp TS mà xây dựng lớp mới TS2 dẫn xuất từ lớp TS. Trong lớp TS2 đ- a thêm thuộc tính dc (địa chỉ) và các ph- ơng thức nhap, in. Cụ thể lớp TS2 đ- ợc định nghĩa nh- sau:

```
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(stdprn,"\nDia chi: %s", dc);
          }
   };
```

Trong lớp TS2 không xây dựng lại ph- ơng thức xem_in, mà sẽ dùng ph- ơng thức xem_in của lớp TS. Ch- ơng trình mới nh- 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(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(); //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(stdprn,"\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 ch- ơng trình này, chúng ta nhận thấy: Dữ liệu in ra vẫn không có địa chỉ. Điều này có thể giải thích nh- sau:

Xét câu lệnh (thứ 2 từ d- ới lên trong hàm main):

t[i].xem_in() ;

Câu lệnh này gọi tới ph- ơng thức xem_in của lớp TS2 (vì t[i] là đối t- ợng của lớp TS2). Nh- ng lớp TS2 không định nghĩa ph- ơng thức xem_in, nên ph- ơng thức TS::xem_in() sẽ đ- ợc gọi tới. Hãy theo rõi ph- ơng thức này:

```
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)
  }
```

Các lệnh đầu của ph- ơng thức sẽ in họ tên thí sinh. Nếu chọn có (bấm phím C), thì câu lệnh:

```
this->in() ;
```

sẽ đ- ợc thực hiện. Mặc dù địa chỉ của t[i] (là đối t- ợng của lớp TS2) đ- ợc truyền cho con trỏ this, thế nh- ng câu lệnh này luôn luôn gọi tới ph- ơng thức TS::in(), vì con trỏ this ở đây có kiểu TS và vì in() là ph- ơng thức tĩnh. Kết quả là không in đ- ợc địa chỉ của thí sinh.

Nh- vậy việc sử dụng các ph- ơng thức tĩnh in() (trong các lớp TS và TS2) đã không đáp ứng đ- ợc yêu cầu phát triển ch- ơng trình. Có một giải pháp rất đơn giản là: Định nghĩa các ph- ơng thức in() trong các lớp TS và TS2 nh- các ph- ơng thức ảo (virtual).

§ 3. PH□ ƠNG THỨC ẢO VÀ T□ ƠNG ỨNG BỘI

3.1. Cách định nghĩa ph□ơng thức ảo

Giả sử A là lớp cơ sở, các lớp B, C, D dẫn xuất (trực tiếp hoặc dán tiếp) từ A. Giả sử trong 4 lớp trên đều có các ph- ơng thức trùng dòng tiêu đề (trùng kiểu, trùng tên, trùng các đối). Để định nghĩa các ph- ơng thức này là các ph- ơng thức ảo, ta chỉ cần:

+ Hoặc thêm từ khoá virtual vào dòng tiêu đề của ph- ơng 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 đ- ợc đặt bên ngoài định nghĩa lớp. Ví dụ nếu viết nh- 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 nh- sau:

```
class A
  {
    ...
    virtual void  hien_thi() ;
  } ;
void  hien_thi()  // Đúng
  {
    cout << "\n Đây là  lớp A" ;
  };
```

3.2. Quy tắc gọi ph□ơng thức ảo

Để có sự so sánh với ph- ơng thức tĩnh, ta nhắc lại quy tắc gọi ph- ơng thức tĩnh nêu trong §1.

3.2.1. Quy tắc gọi ph□ơng thức tĩnh

Lời gọi tới ph- ơng thức tĩnh bao giờ cũng xác định rõ ph- ơng thức nào (trong số các ph- ơng thức trùng tên của các lớp có quan hệ thừa kế) đ- ợc gọi:

1. Nếu lời gọi xuất phát từ một đối t- ợng của lớp nào, thì ph- ơng thức của lớp đó sẽ đ- ợc gọi.

2. Nếu lời gọi xuất phát từ một con trỏ kiểu lớp nào, thì ph- ơng thức của lớp đó sẽ đ- ợc gọi bất kể con trỏ chứa địa chỉ của đối t- ợng nào.

3.2.2. Quy tắc gọi ph□ơng thức ảo

Ph- ơng thức ảo chỉ khác ph- ơng thức tĩnh khi đ- ợc gọi từ một con trỏ (tr- ờng hợp 2 nêu trong mục 3.2.1). Lời gọi tới ph- ơng thức ảo từ một con trỏ ch- a cho biết rõ ph- ơng thức nào (trong số các ph- ơng thức ảo trùng tên của các lớp có quan hệ thừa kế) sẽ đ- ợc gọi. Điều này phụ thuộc vào đối t- ợng cụ thể mà con trỏ đang trỏ tới: Con trỏ đang trỏ tới đối t- ợng của lớp nào thì ph- ơng thức của lớp đó sẽ đ- ợc 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 t- ợng:

```
A  *p ;  //  p là con trỏ kiểu A
A   a ;  //  a là biến đối t- ợng kiểu A
B   b ;  //  b là biến đối t- ợng kiểu B
C   c ;  //  c là biến đối t- ợng kiểu C
D   d ;  //  d là biến đối t- ợng kiểu D
```

Xét lời gọi tới các ph- ơng thức ảo hien_thi sau:

```
p = &a;            // p trỏ tới đối t- ợng a của lớp A
p->hien_thi() ;  // Gọi tới A::hien_thi()
p = &b;            // p trỏ tới đối t- ợng b của lớp B
p->hien_thi() ;  // Gọi tới B::hien_thi()
p = &c;            // p trỏ tới đối t- ợng c của lớp C
p->hien_thi() ;  // Gọi tới C::hien_thi()
p = &d;            // p trỏ tới đối t- ợng d của lớp D
p->hien_thi() ;  // Gọi tới D::hien_thi()
```

3.3. T□ơng ứng bội

Chúng ta nhận thấy cùng một câu lệnh

```
p->hien_thi();
```

t- ơng ứng với nhiều ph- ơng thức khác nhau. Đây chính là t- ơng ứng bội. Khả năng này rõ ràng cho phép xử lý nhiều đối t- ợng khác nhau, nhiều công việc, thậm chí nhiều thuật toán khác nhau theo cùng một cách thức, cùng một l- ợc đồ. Điều này sẽ đ- ợc minh hoạ trong các mục tiếp theo.

3.4. Liên kết động

Có thể so sánh sự khác nhau giữ ph- ơng thức tĩnh và ph- ơng thức ảo trên khía cạnh liên kết một lời gọi với một ph- ơng thức. Trở lại ví dụ trong 3.2:

```
A  *p ;  //  p là con trỏ kiểu A
A   a ;  //  a là biến đối t- ợng kiểu A
B   b ;  //  b là biến đối t- ợng kiểu B
C   c ;  //  c là biến đối t- ợng kiểu C
D   d ;  //  d là biến đối t- ợng kiểu D
```

Nếu hien_thi() là các ph- ơng thức tĩnh, thì dù p chứa địa chỉ của các đối t- ợng a, b, c hay d, thì lời gọi:

```
p->hien_thi() ;
```

luôn luôn gọi tới ph- ơng thức A::hien_thi()

Nh- vậy một lời gọi (xuất phát từ con trỏ) tới ph- ơng thức tĩnh luôn luôn liên kết với một ph- ơng thức cố định và sự liên kết này xác định trong quá trình biên dịch ch- ơng trình.

Cũng với lời gọi:

```
p->hien_thi() ;
```

nh- trên, nh- ng nếu hien_thi() là các ph- ơng thức ảo, thì lời gọi này không liên kết cứng với một ph- ơng thức cụ thể nào. Ph- ơng thức mà nó liên kết (gọi tới) còn ch- a xác định trong giai đoạn dịch ch- ơng trình. Lời gọi này sẽ:

+ liên kết với A::hien_thi() , nếu p chứa địa chỉ đối t- ợng lớp A
+ liên kết với B::hien_thi() , nếu p chứa địa chỉ đối t- ợng lớp B
+ liên kết với C::hien_thi() , nếu p chứa địa chỉ đối t- ợng lớp C
+ liên kết với D::hien_thi() , nếu p chứa địa chỉ đối t- ợng lớp D

Nh- vậy một lời gọi (xuất phát từ con trỏ) tới ph- ơng thức ảo không liên kết với một ph- ơng thức cố định, mà tuỳ thuộc vào nội dung con trỏ. Đó là sự liên kết động và ph- ơng thức đ- ợc liên kết (đ- ợc gọi) thay đổi mỗi khi có sự thay đổi nội dung con trỏ trong quá trình chạy ch- ơng trình.

3.5. Quy tắc gán địa chỉ đối t□ợng cho con trỏ lớp cơ sở

+ Nh- đã nói trong §1, C++ cho phép gán địa chỉ đối t- ợng của một lớp dẫn xuất cho con trỏ của lớp cơ sở. Nh- vậy các phép gán sau (xem 3.2) là đúng:

```
A  *p ;  //  p là con trỏ kiểu A
A   a ;   //  a là biến đối t- ợng kiểu A
B   b ;   //  b là biến đối t- ợng kiểu B
C   c ;   //  c là biến đối t- ợng kiểu C
D   d ;   //  d là biến đối t- ợng kiểu D
p = &a;    // p và a cùng lớp A
p = &b;   // p là con trỏ lớp cơ sở, b là đối t- ợng lớp dẫn xuất
p = &c;   // p là con trỏ lớp cơ sở, c là đối t- ợng lớp dẫn xuất
p = &d;   // p là con trỏ lớp cơ sở, d là đối t- ợng lớp dẫn xuất
```

+ *Tuy nhiên cần chú ý là:* Không cho phép gán địa chỉ đối t- ợng của lớp cở sở cho con trỏ của lớp dẫn xuất. Nh- vậy ví dụ sau là sai:

```
B  *q ;
A  a ;
q = &a;
```

Sai vì: Gán địa chỉ đối t- ợng của lớp cơ sở A cho con trỏ của lớp dẫn xuất B

3.6. Ví dụ

Ta sửa ch- ơng trình trong §1 bằng cách định nghĩa các ph- ơng thức xuat() là ảo. Khi đó bốn câu lệnh:

```
hien(&a);
hien(&b);
hien(&c);
hien(&d);
```

trong hàm main (của ch- ơng trình d- ới đây) sẽ lần l- ợt gọi tới 4 ph- ơng thức khác nhau:

```
A::xuat()
B::xuat()
C::xuat()
D::xuat()
```

```
//CT6-01B
// Phuong thuc ảo và t- ơng 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 ph□ơng thức ảo

Cũng giống nh- các ph- ơng thức thông th- ờng khác, ph- ơng thức ảo cũng có tính thừa kế. Chẳng hạn trong ch- ơng trình trên (mục 3.4) ta bỏ đi ph- ơng thức xuat() của lớp D, thì câu lệnh:

```
hien(&d) ;
```

(câu lệnh gần cuối trong hàm main) sẽ gọi tới C::xuat() , ph- ơng thức này đ- ợc kế thừa trong lớp D (vì D dẫn xuất từ C).

§ 4. SỰ LINH HOẠT CỦA PH□ ƠNG THỨC ẢO TRONG PHÁT TRIỂN NÂNG CẤP CH□ ƠNG TRÌNH

Ví dụ về các lớp TS và TS2 trong §2 đã chỉ ra sự hạn chế của ph- ơng thức tĩnh trong việc sử dụng tính thừa kế để nâng cấp, phát triển ch- ơng trình. Trong §2 cũng đã chỉ ra lớp TS2 ch- a đáp ứng đ- ợc yêu cầu nêu ra là in địa chỉ của thí sinh. Giải pháp cho vấn đề này rất đơn giản: Thay các ph- ơng thức tĩnh in() bằng cách dùng chúng nh- các ph- ơng thức ảo. Ch- ơng trình khi đó sẽ nh- sau:

```
//CT6-03B
// Sự linh hoạt của ph- ơng thức ảo
// 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;
            }
          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 ph- ơng thức ảo nên
              //có thể gọi đến  TS::in() hoặc TS2::in()
            }
    } ;
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(stdprn,"\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 ch- ơng trình này, chúng ta nhận thấy: Dữ liệu thí sinh in ra đã có địa chỉ. Điều này có thể giải thích nh- sau:

Xét câu lệnh (thứ 2 từ d- ới lên trong hàm main):

```
t[i].xem_in() ;
```

Câu lệnh này gọi tới ph- ơng thức xem_in của lớp TS2 (vì t[i] là đối t- ợng của lớp TS2). Nh- ng lớp TS2 không định nghĩa ph- ơng thức xem_in, nên ph- ơng thức TS::xem_in() sẽ đ- ợc gọi tới. Hãy theo rõi ph- ơng thức này:

```
void xem_in()
  {
    int ch;
    cout << "\nHo ten: " << ht ;
    cout << "\nCo in khong? - C/K" ;
    ch = toupper(getch());
    this->in(); // Vì in() là ph- ơng thức ảo nên
                //có thể gọi đến  TS::in() hoặc TS2::in()
  }
```

Các lệnh đầu của ph- ơng thức sẽ in họ tên thí sinh. Nếu chọn Có (bấm phím C), thì câu lệnh:

```
this->in() ;
```

sẽ đ- ợc thực hiện. Địa chỉ của t[i] (là đối t- ợng của lớp TS2) đ- ợc truyền cho con trỏ this (của lớp cơ sở TS). Vì in() là ph- ơng thức ảo và vì this đang trỏ tới đối t- ợng t[i] của lớp TS2, nên câu lệnh này gọi tới ph- ơng thức TS2::in(). Trong ph- ơng thức TS2::in() có in địa chỉ của thí sinh.

Nh- vậy việc sử dụng các ph- ơng thức tĩnh in() (trong các lớp TS và TS2) đã không đáp ứng đ- ợc yêu cầu phát triển ch- ơng trình. Có một giải pháp rất đơn giản là: Định nghĩa các ph- ơng thức in() trong các lớp TS và TS2 nh- các ph- ơng thức ảo (virtual).

§ 5. LỚP CƠ SỞ TRỪU T□ ỢNG

5.1. Lớp cơ sở trừu t□ợng

Một lớp cơ sở trừu t- ợng là một lớp chỉ đ- ợc dùng làm cơ sở cho các lớp khác. Không hề có đối t- ợng nào của một lớp trừu t- ợng đ- ợc tạo ra cả, bởi vì nó chỉ đ- ợc dùng để định nghĩa một số khái niệm tổng quát, chung cho các lớp khác. Một ví dụ về lớp trừu t- ợng là lớp CON_VAT (con vật), nó sẽ dùng làm cơ sở để xây dựng các lớp con vật cụ thể nh- lớp CON_CHO (con chó), CON_MEO (con mèo),... (xem ví dụ bên d- ới)

Trong C++ , thuật ngữ "Lớp trừu tượng" đặc biệt áp dụng cho các lớp có chứa các phương thức ảo thuần tuý. Ph- ơng thức ảo thuần tuý là một ph- ơng thức ảo mà nội dung của nó không có gì. Cách thức định nghĩa một ph- ơng thức ảo thuần tuý nh- sau:

```
virtual  void  tên_ph- ơng_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 t- ợng. Các ph- ơng thức nhap và xuat đ- ợc khai báo là các lớp ảo thuần tuý (bằng cách gán số 0 cho chúng thay cho việc cài đặt các ph- ơng thức này). Ph- ơng thức chuong() là một ph- ơng thức bình th- ờng và sẽ phải có một định nghĩa ở đâu đó cho ph- ơng thức này.

Không có đối t- ợng nào của một lớp trừu t- ợng lại có thể đ- ợc phát sinh. Tuy nhiên các con trỏ và các biến tham chiếu đến các đối t- ợng của lớp trừu t- ợng thì vẫn hợp lệ. Bất kỳ lớp nào dẫn xuất từ một lớp cớ sở trừu t- ợng phải định nghĩa lại tất cả các ph- ơng thức thuần ảo mà nó thừa h- ởng, hoặc bằng các ph- ơng thức ảo thuần tuý, 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 h- ớng đối t- ợng, ta vẫn có thể có một lớp trừu t- ợng mà không nhất thiết phải chứa đựng những ph- ơng thức thuần tuý ảo.

Một cách tổng quát mà nói thì bất kỳ lớp nào mà nó chỉ đ- ợc 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 t- ợng là xem có dùng lớp đó để khai báo các đối t- ợng hay không? . Nếu không thì đó là lớp cơ sở trừu t- ợng.

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 ch- ơng 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.

Ch- ơng trình đ- ợc tổ chức nh- sau:

+ Tr- ớc tiên định nghĩa lớp CON_VAT là lớp cơ sở ảo. Lớp này có một thuộc tính là tên con vật và một ph- ơng thức ảo dùng để x- ng tên.

+ Hai lớp là CON_MEO và CON_CHO đ- ợc 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 t- ợng kiểu CON_MEO hoặc CON_CHO.

Lớp sẽ có 3 ph- ơng thức để thực hiện 3 chức năng nêu trên của ch- ơng trình. Nội dung ch- ơng trình nh- 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 t- ợng. Tuy nhiên theo quan điểm của C++ thì lớp này ch- a phải là lớp cơ sở trừu t- ợng, vì trong lớp không có các ph- ơng thức thuần tuý ảo. Ph- ơng thức xung_ten:

```
virtual void xung_ten()
    {
    }
```

là ph- ơng thức ảo, đ- ợc định nghĩa đầy đủ , mặc dù thân của nó là rỗng.

Do vậy khai báo:

```
CON_VAT   cv("Con vat chung");
```

vẫn đ- ợc C++ chấp nhận.

Bây giờ nếu định nghĩa lại ph- ơng thức xung_ten nh- sau:

```
virtual void xung_ten() = 0 ;
```

thì nó trở thành ph- ơng thức thuần ảo và C++ sẽ quan niệm lớp CON_VAT là lớp trừu t- ợng. Khi đó câu lệnh khai báo:

```
CON_VAT   cv("Con vat chung");
```

sẽ bị C++ bắt lỗi với thông báo:

Cannot create instance of abstruct class 'CON_VAT'

§ 6. SỬ DỤNG T□ ƠNG ỨNG BỘI VÀ PH□ ƠNG THỨC ẢO

6.1. Chiến l□ợc sử dụng t□ơng ứng bội

T- ơng ứng bội cho phép xét các vấn đề khác nhau, các đối t- ợng khác nhau, các ph- ơng pháp khác nhau, các cách giải quyết khác nhau theo cùng một l- ợc đồ chung.

Các b- ớc áp dụng t- ơng ứng bội có thể tổng kết lại nh- sau:

1. Xây dựng lớp cơ sở trừu t- ợng bao gồm những thuộc tính chung nhất của các thực thể cần quản lý. Đ- a vào các ph- ơng thức ảo hay thuần ảo dùng để xây dựng các nhóm ph- ơng thức ảo cho các lớp dẫn xuất sau này. Mỗi nhóm ph- ơng 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 ảo. Số mức dẫn xuất là không hạn chế. Các lớp dẫn xuất sẽ mô tả các đối t- ợng cụ thể cần quản lý.

3. Xây dựng các ph- ơng thức ảo trong các dẫn xuất. Các ph- ơng thức này tạo thành các nhóm ph- ơng thức ảo trong sơ đồ các lớp có quan hệ thừa kế.

4. Xây dựng lớp quản lý các đối t- ợng. Dữ liệu của lớp này là một dẫy con trỏ của lớp cơ sở trừu t- ợng ban đầu. Các con trỏ này có thể chứa địa chỉ đối t- ợng của các lớp dẫn xuất. Do vậy có thể dùng các con trỏ này để thực hiện các thao tác trên các đối t- ợng của bất kỳ lớp dẫn xuất nào.

6.2. Ví dụ

Ch- ơng trình quản lý các con vật trong §5 là một ví dụ về cách sử dụng t- ơng ứng bội. D- ới 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 đ- ợc các bức tranh khác nhau. Ch- ơng trình d- ới đây sẽ cho hiện tất cả các bức tranh khác nhau. Ch- ơng trình đ- ợc tổ chức theo các b- ớc nêu trong 6.1:

+ Lớp cơ sở trừu t- ợng là lớp HINH (hình) gồm một thuộc tính mau (mầu) và một ph- ơng thức ảo thuần tuý:

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 t- ợng: DTHANG, HTRON, CHUNHAT và VUONG. Sử dụng ph- ơng thức draw gọi từ 4 phần tử mảng nói trên sẽ nhận đ- ợc một bức tranh. Bằng cách hoán vị các phần tử này, sẽ nhận đ- ợc 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;
            }
          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 bk1, int m):HINH(m)
        {
          bk = bk1;
        }
      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();
                            }
            }
          DTHANG 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 t- ơng ứng bội để tổ chức thực hiện các thuật toán khác nhau trên cùng một bài toán nh- sau:

+ Lớp cơ sở trừu t- ợng sẽ chứa dữ liệu bài toán và một ph- ơng thức ảo.

+ Mỗi lớp dẫn xuất ứng với một thuật toán cụ thể. Ph- ơng 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 t- ợng của lớp dẫn xuất. Sau đó dùng các phần tử mảng con trỏ để gọi tới các ph- ơng 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 hoạ 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). Ch- ơng trình gồm 4 lớp:

+ Lớp cơ sở trừu t- ợng:

```
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.

- Ph- ơng thức hoan_vi(i,j) dùng để hoán vị các phần tử a[i] và a[j]. Ph- ơng thức này đ- ợc dùng trong 3 lớp dẫn xuất bên d- ới.

- Ph- ơng 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 ph- ơng thức ảo:

```
    virtual void sapxep(int *a1, long n) ;
```

để thực hiện hiện việc sắp xếp theo theo một thuật toán cụ thể.

+ Trong hàm main() sẽ tạo ra một dãy 30000 số nguyên một cách ngẫu nhiên, sau đó lần l- ợt sử dụng 3 thuật toán sắp xếp để so sánh. Kết quả nh- 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 ch- ơng trình nh- 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 tg = a[i];
              a[i] = a[j];
              a[j] = tg;
            }
      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();
        gettime(&t1);
        s[k]->sapxep(a,n);
        gettime(&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();
}
```

CH□ ƠNG 7

CÁC DÒNG TIN (STREAM)

C đã cung cấp một th- viện các hàm nhập xuất nh- printf, scanf, gets, getch(), puts, puch(), fprintf, fscanf, fopen, fwite, fread,... . Các hàm này làm việc khá hiệu quả nh- ng không thích ứng với cách tổ chức ch- ơng trình h- ớng đối t- ợng.

C++ sử dụng khái niệm dòng tin (stream) và đ- a ra các lớp dòng tin để tổ chức việc nhập xuất. Dòng tin có thể xem nh- 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 đ- a các byte từ bộ nhớ ra dòng tin (khi đó gọi là dong 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

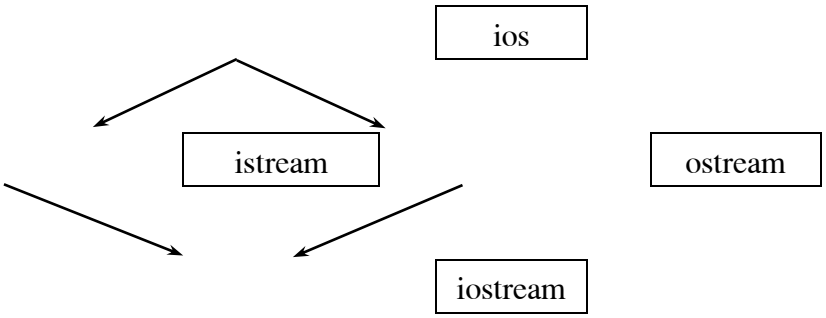
Có 4 lớp quan trọng cần nhớ là:

+ 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 nh- sau:



Lớp ios

+ Thuộc tính của lớp: Trong lớp ios định nghĩa các thuộc tính đ- ợc sử dụng làm các cờ định dạng cho việc nhập xuất và các cờ kiểm tra lỗi (xem bên d- ới).

+ Các ph- ơng thức: Lớp ios cung cấp một số ph- ơng thức phục vụ việc định dạng dữ liệu nhập xuất, kiểm tra lỗi (xem bên d- ới).

Lớp istream

Lớp này cung cấp toán tử nhập >> và nhiều ph- ơng thức nhập khác (xem bên d- ới) nh- các ph- ơng thức: get, getline, read, ignore, peek, seekg, tellg,...

Lớp ostream

Lớp này cung cấp toán tử xuất << và nhiều ph- ơng thức xuất khác (xem bên d- ới) nh- các ph- ơng thức: put, write, flush, seekp, tellp,...

Lớp iostream

Lớp này thừa kế các ph- ơng thức nhập xuất của các lớp istream và ostream.

§ 2. DÒNG CIN VÀ TOÁN TỬ NHẬP

Dòng cin là một đối t- ợng kiểu istream đã định nghĩa trong C++ . Đó là dòng vào (input) chuẩn gắn với bàn phím (t- ơng tự nh- stdin của C). Các thao tác nhập trên dòng cin đồng nghĩa với nhập dữ liệu từ bàn phím.

Do cin là một đối t- ợng của lớp istream nên với cin chung ta có thể sử dụng toán tử nhập >> và các ph- ơng thức nhập của các lớp ios và istream.

Cách dùng toán tử nhập để đọc dữ liệu từ dòng cin nh- 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 nh- sau:

cin >> Tham_số_1 >> Tham_số_2 >> ... >> Tham_số_k ;

Cách thức nhập nh- sau: Bỏ qua các ký tự trắng (dấu cách, dấu tab, dấu chuyển dòng) đứng tr- ớc nếu có và sau đó đọc vào các ký tự t- ơng ứng với kiểu yêu cầu. Cụ thể đối với từng kiểu nh- sau:

Khi nhập số nguyên sẽ bỏ qua các ký tự trắng đứng tr- ớc 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 t- ơng tự: Bỏ qua các khoảng trắng đứng tr- ớc 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 tr- ớc nếu có, sau đó nhận một ký tự khác ký tự trắng. Ví dụ nếu gõ <space><space>XY thì ký tự X đ- ợc nhận và con trỏ nhập dừng tại ký tự Y.

Phép nhập một dãy ký tự: Bỏ qua các khoảng trắng đứng tr- ớc 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 ch- ơng 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 nh- sau:

```
n=123
x=3.14
ch='Z'
ten="HONG"
que = "HAI"
```

Con trỏ nhập sẽ dừng tại ký tự <space> tr- ớc từ PHONG. Các ký tự còn lại sẽ đ- ợc nhận trong các câu lệnh nhập tiếp theo.

Ví dụ 2: Xét đoạn ch- ơng 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 KÝ TỰ VÀ CHUỖI KÝ 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 ph- ơng thức sau (định nghĩa trong lớp istream):

cin.get cin.getline cin.ignore

3.1. Ph- ơng thức get có 3 dạng (thực chất có 3 ph- ơng 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 hoạ 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 đ- ợc tham chiếu bởi ch.

Chú ý:

+ Cách thức đọc của cin.get dạng 2 cũng giống nh- dạng 1

+ Do cin.get() dạng 2 trả về tham chiếu tới cin, nên có thể sử dụng các ph- ơng 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 đ- a vào vùng nhớ do str trỏ tới. Quá trình đọc kết thúc khi xẩy 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' đ- ợc bổ sung vào dãy ký tự nhận đ- ợc

+ ký tự giới hạn vẫn còn lại trên dòng nhập để dành cho các lệnh nhập tiếp theo.

Chú ý:

+ Cũng giống nh- get() dạng 2, có thể viết các ph- ơng thức get() dạng 3 nối đuôi nhau trên một dòng lệnh.

+ Ký tự <Enter> còn lại trên dòng nhập có thể làm trôi ph- ơng thức get() dạng 3. Ví dụ xét đoạn ch- ơng 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 ch- ơng 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 ph- ơng thức get() dạng 1 hoặc dạng 2 để lấy ra ký tự <Enter> trên dòng nhập tr- ớc khi dùng get (dạng 3).

+ Dùng ph- ơng thức ignore để lấy ra một số ký tự không cần thiết trên dòng nhập tr- ớc khi dùng get dạng 3. Ph- ơng thức này viết nh- sau:

```
cin.ignore(n) ; // Lấy ra (loại ra hay bỏ qua) n ký tự trên
                // dòng nhập.
```

Nh- vậy để có thể nhập đ- ợc cả quê quán và cơ quan, cần sửa lại đoạn ch- ơng trình trên nh- 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. Ph- ơng thức getline

T- ơng tự nh- get dạng 3, có thể dùng getline để nhập một dãy ký tự từ bàn phím. Ph- ơng thức này đ- ợc mô tả nh- sau:

```
istream& cin.getline(char *str, int n, char delim = '\n');
```

Ph- ơng thức đầu tiên làm việc nh- get dạng 3, sau đó nó loại <Enter> ra khỏi dòng nhập (ký tự <Enter> không đ- a vào dãy ký tự nhận đ- ợc). Nh- 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 ch- ơng trình nhập họ tên, quê quán và cơ quan bên trên có thể viết nh- 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 nh- get() dạng 2 và get() dạng 3, có thể viết các ph- ơng thức getline() nối đuôi nhau trên một dòng lệnh. Ví dụ đoạn ch- ơng trình trên có thể viết lại nh- 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. Ph- ơng thức ignore

Ph- ơng thức ignore dùng để bỏ qua (loại bỏ) một số ký tự trên dòng nhập. Trong nhiều tr- ờng hợp, đây là việc làm cần thiết để không làm ảnh h- ởng đến các phép nhập tiếp theo.

Ph- ơng thức ignore đ- ợc mô tả nh- sau:

```
istream& cin.ignore(int n=1);
```

Ph- ơng thức sẽ bỏ qua (loại bỏ) n ký tự trên dòng nhập.

3.4. Nhập đồng thời giá trị số và ký tự

Nh- 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 d- ới. 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 tr- ớc khi thực hiện việc nhập ký tự hoặc chuỗi ký tự.

3.5. Ví dụ: Ch- ơng trình d- ới đây sử dụng lớp TSINH (Thí sinh) với 2 ph- ơng thức xuat và nhap.

```
//CT7_04.CPP
// Nhập dữ liêu 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 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 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;
  }
```

§ 4. DÒNG COUT VÀ TOÁN TỬ XUẤT

4.1. Dòng cout

Dòng cout là một đối t- ợng kiểu ostream đã định nghĩa trong C++. Đó là dòng xuất (output) chuẩn gắn với màn hình (t- ơng tự nh- stdout của C). Các thao tác xuất trên dòng cout đồng nghĩa với xuất dữ liệu ra màn hình.

Do cout là một đối t- ợng của lớp ostream nên với cout chung ta có thể sử dụng toán tử xuất << và các ph- ơng 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 nh- sau:

```
cout << Tham_số ;
```

Trong đó Tham_số biểu thị một giá trị cần xuất ra màn hình. Giá trị sẽ đ- ợc biến đổi thành một dãy ký tự tr- ớc khi đ- a ra dòng xuất. Kiểu của Tham_số có thể nh- 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 nh- sau:

```
cout << Tham_số_1 << Tham_số_2 << ... << Tham_số_k ;
```

Chú ý: Toán tử xuất đ- ợc định nghĩa chồng (trùng tên) với toán tử dịch trái và nó cùng có mức độ - u tiên nh- toán tử dịch trái. Xem phụ lục 1 chúng ta thấy toán tử xuất có thứ tự - u 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 nh- 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 nh- 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 ch- a có khả năng định dạng, mà cần sử dụng các công cụ sau:

+ Các ph- ơng thức định dạng

+ Các các cờ định dạng

+ Các hàm và bộ phận định dạng

Mục sau sẽ trình bầy cách định dạng giá trị xuất.

§ 5. CÁC PH□ ƠNG 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: Nh- đã nói ở trên, C++ sẽ biến đổi giá trị cần xuất thành một chuỗi ký tự rồi đ- a 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 đinh 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 ph- ơng 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 d- thừa. Các vị trí d- thừa sẽ đ- ợc 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 ph- ơng 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 ph- ơng 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 - ớc 8
cout.fill('0') ;    // Ký tự độn là số 0
cout << x ;
```

thì kết quả in ra là:

```
0034.46
```

5.2. Các ph- ơng thức định dạng

1. Ph- ơng thức

int cout.width()

cho biết độ rộng quy định hiện tại.

2. Ph- ơng thức

int cout.width(int n)

Thiết lập độ rộng quy định mới là n và trả về độ rộng quy định tr- ớc 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. Ph- ơng thức

int cout.precision()

Cho biết độ chính xác hiện tại (đang áp dụng để xuất các giá trị thức).

4. Ph- ơng thức

int cout.precision(int n)

Thiết lập độ chính xác sẽ áp dụng là n và cho biết độ chính xác tr- ớc đó. Độ chính xác đ- ợc thiết lập sẽ có hiệu lực cho tới khi gặp một câu lệnh thiết lập độ chính xác mới.

5. Ph- ơng thức

char cout.fill()

Cho biết ký tự độn hiện tại đang đ- ợc áp dụng.

6. Ph- ơng thức

char cout.fill(char ch)

Quy định ký tự độn mới sẽ đ- ợc dùng là ch và cho biết ký tự độn đang dùng tr- ớc đó. Ký tự độn đ- ợc thiết lập sẽ có hiệu lực cho tới khi gặp một câu lệnh chọn ký tự độn mới.

Ví dụ xét ch- ơng trình:

```
//CT7_06.CPP
// Cac phuong thuc dinh dang
#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, ch- ơng 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 bầy các ph- ơng 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 tr- ớc, ví dụ:

```
***35
**-89
```

Chú ý: Mặc định cờ ios::right bật.

Cờ ios::internal: Cờ ios:internal có tác dụng giống nh- cờ ios::right chỉ khác là dấu (nếu có) in đầu tiên, ví dụ:

```
***35
```

-**89

Ch- ơng trình sau minh hoạ cách dùng các cờ định vị:

```
//CT7_06.CPP
// Cac phuong thuc dinh dang
// Co dinh vi
#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 ch- ơng trình in ra 6 dòng nh- 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 đ- ợc in d- ới dạng cơ số 10

+ Khi ios::oct bật : Số nguyên đ- ợc in d- ới dạng cơ số 8

+ Khi ios::hex bật : Số nguyên đ- ợc in d- ới 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 d- ới dạng thập phân, số chữ số phần phân (sau dấu chấm) đ- ợc tính bằng độ chính xác n nh- ng khi in thì bỏ đi các chữ số 0 ở cuối.

Ví dụ nếu độ chính xác n = 4 thì:

Số thực -87.1500 đ- ợc in: -87.15

Số thực 23.45425 đ- ợc in: 23.4543

Số thực 678.0 đ- ợc in: 678

+ Khi ios::fixed bật và cờ ios::showpoint bật thì số thực in ra d- ới dạng thập phân, số chữ số phần phân (sau dấu chấm) đ- ợc in ra đúng bằng độ chính xác n.

Ví dụ nếu độ chính xác n = 4 thì:

Số thực -87.1500 đ- ợc in: -87.1500

Số thực 23.45425 đ- ợc in: 23.4543

Số thực 678.0 đ- ợc in: 678.0000

+ Khi ios::scientific bật và cờ ios::showpoint tắt thì số thực in ra d- ới dạng mũ (dạng khoa học). Số chữ số phần phân (sau dấu chấm) của phần định trị đ- ợc tính bằng độ chính xác n nh- ng khi in thì bỏ đi các chữ số 0 ở cuối.

Ví dụ nếu độ chính xác n = 4 thì:

Số thực -87.1500 đ- ợc in: -8.715e+01

Số thực 23.45425 đ- ợc in: 2.3454e+01

Số thực 678.0 đ- ợc in: 6.78e+02

+ Khi ios::scientific bật và cờ ios::showpoint bật thì số thực in ra d- ới dạng mũ. Số chữ số phần phân (sau dấu chấm) của phần định trị đ- ợc in đúng bằng độ chính xác n.

Ví dụ nếu độ chính xác n = 4 thì:

Số thực -87.1500 đ- ợc in: -8.7150e+01

Số thực 23.45425 đ- ợc in: 2.3454e+01

Số thực 678.0 đ- ợc 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 đ- ợc in tr- ớc số d- ơng.

+ Nếu cờ ios::showpos bật thì dấu cộng đ- ợc in tr- ớc số d- ơng.

Cờ ios::showbase

+ Nếu cờ ios::showbase bật thì số nguyên hệ 8 đ- ợc in bắt đầu bằng ký tự 0 và số nguyên hệ 16 đ- ợc 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 tr- ớc số nguyên hệ 8 và không in 0x tr- ớc 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 (nh- A, B, C, ...) đ- ợc in d- ới dạng chữ hoa.

+ Nếu cờ ios::uppercase tắt (mặc định) thì các chữ số hệ 16 (nh- A, B, C, ...) đ- ợc in d- ới dạng chữ th- ờng.

6.3. Các ph- ơng thức bật tắt cờ

Các ph- ơng thức này định nghĩa trong lớp ios.

+ Ph- ơng 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 th- ờng giá trị f đ- ợc xác định bằng cách tổ hợp các cờ trình bầy trong mục 6.1.

Ví dụ câu lệnh:

cout.setf(ios::showpoint | ios::scientific) ;

sẽ bật các cờ ios::showpoint và ios::scientific.

+ Ph- ơng 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 th- ờng giá trị f đ- ợc xác định bằng cách tổ hợp các cờ trình bầy trong mục 6.1.

Ví dụ câu lệnh:

cout.unsetf(ios::showpoint | ios::scientific) ;

sẽ tắt các cờ ios::showpoint và ios::scientific.

+ Ph- ơng thức

long cout.flags(long f) ;

có tác dụng giống nh- cout.setf(long). Ví dụ câu lệnh:

cout.flags(ios::showpoint | ios::scientific) ;

sẽ bật các cờ ios::showpoint và ios::scientific.

+ Ph- ơng 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    // nh- cờ ios::dec
oct    // nh- cờ ios::oct
hex    // nh- cờ ios::hex
endl   // xuất ký tự '\n' (chuyển dòng)
flush  // đẩy dữ liệu ra thiết bị xuất
```

Chúng có tác dụng nh- cờ định dạng nh- ng đ- ợc viết nối đuôi trong toán tử xuất nên tiện sử dụng hơn.

Ví dụ xét ch- ơng trình đơn giản sau:

```
//CT7_08.CPP
// Bo phan dinh dang
#include <iostream.h>
#include <conio.h>
```

```
void main()
  {
    clrscr();
    cout.setf(ios::showbase)
    cout << "ABC" << endl << hex << 40 << " " << 41;
    getch();
  }
```

Ch- ơng 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)              // nh- cout.width(int n)
setpecision(int n)       // nh- cout.pecision(int n)
setfill(char ch)         // nh- cout. fill(char ch)
setiosflags(long l)      // nh- cout.setf(long f)
resetiosflags(long l)    // nh- cout.unsetf(long f)
```

Các hàm định dạng có tác dụng nh- các ph- ơng thức định dạng nh- ng đ- ợc viết nối đuôi trong toán tử xuất nên tiện sử dụng hơn.

Chú ý 1: Các hàm định dạng (cũng nh- các bộ phận định dạng) cần viết trong các toán tử xuất. Một hàm định dạng đứng một mình nh- 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 ch- ơng trình câu lệnh:

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

Ví dụ có thể thay ph- ơng thức

```
cout.setf(ios::showbase) ;
```

trong ch- ơng trình của mục 7.1 bằng hàm

```
cout << setiosflags(ios::showbase);
```

(chú ý hàm phải viết trong toán tử xuất)

Nh- vậy ch- ơng trình trong 7.1 có thể viết lại theo các ph- ơng án sau:

Ph□ơng 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();
  }
```

Ph□ơng án 2:

```
#include <iostream.h>
#include <iomanip.h>
#include <conio.h>
void main()
  {
```

```
    clrscr();
    cout << "ABC" << endl << setiosflags(ios::showbase)
    << hex << 40 << " " << 41;
    getch();
  }
```

D- ới 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 nh- sau:

```
    23
    0x17
    **23
    00023
```

7.3. Ví dụ: Ch- ơng trình d- ới đây minh hoạ cách dùng các hàm định dạng và ph- ơng thức định dạng để in danh sách thí sinh d- ới dạng bảng với các yêu cầu sau: Số báo danh in 4 ký tự (chèn thêm số 0 vào tr- ớc ví dụ 0003), tổng điểm in với đúng một chữ số phần phân.

```
//CT7_08.CPP
// Bo phan dinh dang
// Ham dinh dang
// Co dinh dang
#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 t- ợng của các lớp Stream) đã định nghĩa tr- ớc, đ- ợc cài đặt khi ch- ơng trình khởi động. Hai trong số đó đã nói ở trên là:

cin dòng input chuẩn gắn với bàn phím, giống nh- stdin của C.

cout dòng output chuẩn gắn với màn hình, giống nh- stdout của C.

Hai dòng tin chuẩn khác là:

cerr dòng output lỗi chuẩn gắn với màn hình, giống nh- stderr của C.

clog giống cerr nh- ng có thêm bộ đệm.

Chú ý 1: Có thể dùng các dòng cerr và clog để xuất ra màn hình nh- đã dùng đối với cout.

Chú ý 2: Vì clog có thêm bộ đệm, nên dữ liệu đ- ợc đ- a vào bộ đệm. Khi đầy bộ đệm thì đ- a dữ liệu từ bộ đệm ra dòng clog. Vì vậy tr- ớc khi kết thúc xuất cần dùng ph- ơng thức:

```
clog.flush();
```

để đẩy dữ liệu từ bộ đệm ra clog.

Ch- ơng trình sau minh hoạ 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 ch- ơng 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();
  }
```

§ 9. XUẤT RA MÁY IN

Trong số 4 dòng tin chuẩn không dòng nào gắn với máy in. Nh- vậy không thể dùng các dòng này để xuất dữ liệu ra máy in. Để xuất dữ liệu ra máy in (cũng nh- 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 t- ợng 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 t- ợng kiểu ofstream hay gọi là tên dòng xuất do chúng ta tự đặt.

+ fd (file descriptor) là chỉ số tập tin. Chỉ số tập 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 đ- ợc 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 tr- ớc hết chuyển vào bộ đệm, khi đầy bộ đệm thì dữ liệu sẽ đ- ợc 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 ph- ơng thức flush hoặc bộ phận định dạng flush. Cách viết nh- sau:

```
prn.flush();  // Ph- ơng thức
prn << flush ; // Bộ phận định dạng
```

Các câu lệnh sau sẽ xuất dữ liệu ra prn (máy in) và ý nghĩa của chúng nh- 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 d- ới đây cũng xuất dữ liệu ra máy in nh- ng 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 d- ới 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: Tr- ớc khi kết thúc ch- ơng trình, dữ liệu từ bộ đệm sẽ đ- ợc tự động đẩy ra máy in.

Ch- ơng trình minh hoạ: Ch- ơng trình d- ới đây t- ơng tự nh- ch- ơng trình trong mục 7.3 (chỉ sửa đổi ph- ơng thức xuất) nh- ng thay việc xuất ra màn hình bằng xuất ra máy in.

```
//CT7_08B.CPP
// Xuat ra may in
// Bo phan dinh dang
// Ham dinh dang
#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::leftlios::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

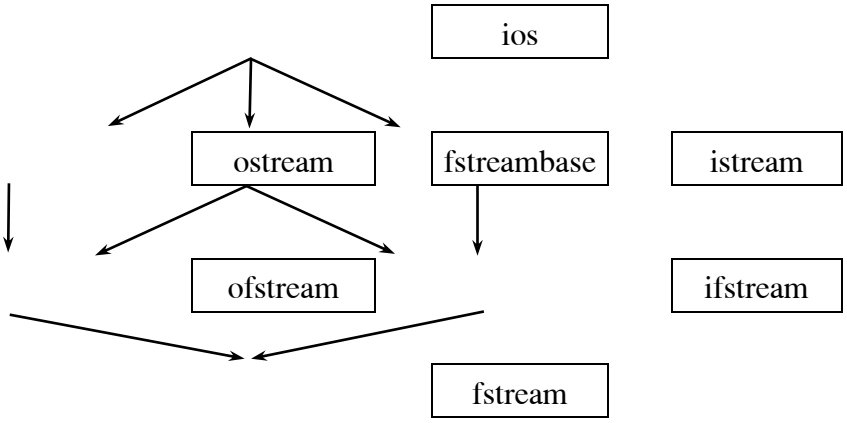
Nh- đã nói ở trên, C++ cung cấp 4 dòng tin chuẩn để làm việc với bàn phím và màn hình. Muốn nhập xuất lên tệp chúng ta cần tạo các dòng tin mới (khai báo các đối t- ợng Stream) và gắn chúng với một tệp cụ thể. C++ cung cấp 3 lớp stream để làm điều này, đó là các lớp:

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 nh- sau:



10.2. Ghi dữ liệu lên tệp

Thủ tục ghi dữ liệu lên tệp nh- 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 nh- thể xuất dữ liệu ra dòng xuất chuẩn cout.

10.3. Đọc dữ liệu từ tệp

Thủ tục đọc dữ liệu từ tệp nh- 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 nh- 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 nh- 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 nh- thể nhập dữ liệu từ dòng nhập chuẩn cin.

3. Thực hiện xuất dữ liệu ra dòng nhập-xuất vừa tạo nh- thể xuất dữ liệu ra dòng xuất chuẩn cout.

Nhận xét: Nh- vậy:

1. Việc xuất dữ liệu ra máy in hoặc lên tệp đ- ợc thực hiện hoàn toàn giống nh- xuất dữ liệu ra dòng xuất chuẩn cout (màn hình).

2. Việc đọc dữ liệu từ tệp đ- ợc thực hiện hoàn toàn giống nh- 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 ph- ơng thức của các lớp ios và ostream. Nó cũng thừa kế ph- ơng thức:

close

của lớp fstreambase. Ngoài ra lớp ofstream có thêm các hàm tạo và các ph- ơng thức sau:

1. Hàm tạo:

ofstream() ; // Không đối

dùng để tạo một đối t- ợng ofstream (dòng xuất), ch- a gắn với tệp.

2. Hàm tạo:

ofstream(const char *fn, int mode = ios::out,
int prot = filebuf::openprot);

dùng để tạo một đối t- ợng ofstream, mở tệp có tên fn để ghi và gắn đối t- ợng vừa tạo với tệp đ- ợc 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 xoá

ios::app ghi bổ sung vào cuối tệp

ios::ate chuyển con trỏ tệp tới cuối tệp sau khi mở tệp

ios::trunc xoá nội dung của tệp nếu nó tồn tại

ios::nocreate nếu tệp ch- a 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 đ- ợc gán một giá trị mặc định.

3. Hàm tạo:

ofstream(int fd);

dùng để tạo một đối t- ợng ofstream và gắn 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:

ofstream(int fd, char *buf, int n);

dùng để tạo một đối t- ợng ofstream , gắn nó với một tệp có chỉ số fd đang mở và sử dùng một vùng nhớ n byte do buf trỏ tới làm bộ đệm.

5. Ph- ơng 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 t- ợng ofstream. Các tham số của ph- ơng thức có cùng ý nghĩa nh- 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 ph- ơng thức để xuất dữ liệu ra dòng xuất vừa tạo nh- 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 ph- ơng 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 ph- ơng 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 t- ợng ofstream) để xuất dữ liệu lên nhiều tệp khác nhau.

11.3. Ví dụ

Ch- ơng trình 1: Ch- ơng trình d- ới đây sẽ nhập danh sách n thí sinh. Thông tin thí sinh gồm: Họ tên, tỉnh hoặc thành phố c- trú, số báo danh, các điểm toán lý hoá. Dữ liệu thí sinh đ- ợc 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 nh- 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 phần phân). Ch- ơng trình sử dụng lớp TS (Thí sinh) có 3 ph- ơng 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.

Ch- ơng trình 2 ngay bên d- ới cũng giải quyết cùng bài toán nêu trên nh- ng sử dụng cách ghi tệp thứ 2 (dùng hàm tạo 1 và ph- ơng thức open)

Một điều đáng nói ở đây là việc nhập một chuỗi ký tự (nh- họ tên và tên tỉnh) bằng các ph- ơng thức get hoặc getline ch- a đ- ợc thuận tiện, vì 2 lý do sau: thứ nhất là các ph- ơng thức này có thể bị ký tự chuyển dòng (còn

sót trên cin) làm trôi. Thứ hai là các ph- ơng 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 h- ởng đến các phép nhập tiếp theo. Để khắc phục các nh- ợc điểm trên, chúng ta đ- a vào 2 ch- ơng trình trên hàm getstr để nhập chuỗi ký tự từ bàn phím.

```
//CT7_10.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)  // 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]==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(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();
  }
```

Ch- ơng 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::outlios::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 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 TỆP

12.1. Lớp ifstream

Để đọc dữ liệu từ tệp chúng ta sử dụng lớp ifstream. Lớp ifstream thừa kế các ph- ơng thức của các lớp ios và istream. Nó cũng thừa kế ph- ơng thức:

close

của lớp fstreambase. Ngoài ra lớp ifstream có thêm các hàm tạo và các ph- ơng thức sau:

1. Hàm tạo:

ifstream() ; // Không đối

dùng để tạo một đối t- ợng ifstream (dòng nhập), ch- a gắn với tệp.

2. Hàm tạo:

ifstream(const char *fn, int mode = ios::in,
int prot = filebuf::openprot);

dùng để tạo một đối t- ợng ifstream, mở tệp có tên fn để đọc và gắn đối t- ợng vừa tạo với tệp đ- ợc mở.

+ Tham số fn cho biết tên tệp.

+ 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 tệp tới cuối tệp sau khi mở tệp

+ Tham số thứ ba prot quy định cấp bảo vệ của dòng tin, tham số này có thể bỏ qua vì nó đã đ- ợc gán một giá trị mặc định.

3. Hàm tạo:

ifstream(int fd);

dùng để tạo một đối t- ợng ifstream và gắn 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:

ifstream(int fd, char *buf, int n);

dùng để tạo một đối t- ợng ifstream , gắn nó với một tệp có chỉ số fd đang mở và sử dùng một vùng nhớ n byte do buf trỏ tới làm bộ đệm.

5. Ph- ơng thức:

void open(const char *fn, int mode = ios::in,
int prot = filebuf::openprot);

dùng để mở tệp có tên fn để đọc và gắn nó với đối t- ợng ifstream. Các tham số của ph- ơng thức có cùng ý nghĩa nh- 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 ph- ơng thức để nhập dữ liệu từ dòng nhập vừa tạo nh- thể nhập dữ liệu từ cin (xem các mục trên)

+ *Cách 2:* Dùng hàm tạo 1 để xây dựng một dòng nhập. Sau đó dùng ph- ơng thức open để mở một tệp cụ thể và cho gắn với dòng nhập vừa xây dựng. Khi không cần làm việc với tệp này nữa, chúng ta có thể dùng ph- ơng 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 t- ợng 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 ph- ơng 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 ch- ơng trình:

```
ifstream fin("DSTS");
  if (fin.bad())
    {
      cout << "\nTep DSTS không tồn tai";
      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 ph- ơng thức eof sẽ cho giá trị khác không. Ch- ơng trình d- ới đây dùng ph- ơng thức eof để xác định độ dài (số byte) của tệp TC.EXE (chú ý cần dùng kiểu đọc nhị phân):

```
//CT7_14.CPP
// Do dai tep
#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()) ++dd;
    cout << "\n Do dai TC.EXE: " << dd;
    getch();
  }
```

12.4. Ví dụ

Ch- ơng trình d- ới đây sẽ:

+ Đọc danh sách thí sinh từ tệp DS1.DL do ch- ơng 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

Ch- ơng trình sử dụng lớp TS (Thí sinh) có 4 ph- ơng thức:

```
void xuat();
void sapxep();
void ghitep(char *ttep);
void doctep(char *ttep);
//CT7_12.CPP
// Doc tep
#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 << "\nHo 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.ghitep("DS3.DL");
    t.doctep("DS3.DL");
    t.xuat();
    cout << "\n Hoan thanh";
    getch();
  }
```

§ 13. ĐỌC GHI ĐỒNG THỜI TRÊN TỆP

13.1. Lớp fstream

Để đọc ghi đồng thời trên tệp, chúng ta sử dụng lớp fstream. Lớp fstream thừa kế các ph- ơng thức của các lớp ofstream và ifstream. Ngoài ra lớp fstream có các hàm tạo và ph- ơng thức sau:

1. Hàm tạo:

fstream() ; // Không đối

dùng để tạo một đối t- ợng fstream (dòng nhập-xuất), ch- a gắn với tệp.

2. Hàm tạo:

fstream(const char *fn, int mode,

int prot = filebuf::openprot);

dùng để tạo một đối t- ợng fstream, mở tệp có tên fn và gắn đối t- ợng vừa tạo với tệp đ- ợc mở.

+ Tham số fn cho biết tên tệp.

+ 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 tệp, nếu tệp đã có thì nó bị xoá
ios::in đọc tệp
ios::app ghi bổ sung vào cuối tệp
ios::ate chuyển con trỏ tệp về cuối sau khi mở
ios::trunc xoá nội dung của tệp nếu nó tồn tạI
ios::nocreate nếu tệp ch- a có thì không làm gì (bỏ qua)
ios::noreplace nếu tệp 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 đ- ợc gán một giá trị mặc định.

3. Hàm tạo:

fstream(int fd);

dùng để tạo một đối t- ợng fstream và gắn 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 t- ợng fstream , gắn nó với một tệp có chỉ số fd đang mở và sử dùng một vùng nhớ n byte do buf trỏ tới làm bộ đệm.

5. Ph- ơng thức:

void open(const char *fn, int mode,

int prot = filebuf::openprot);

dùng để mở tệp có tên fn và gắn nó với đối t- ợng fstream. Các tham số của ph- ơng thức có cùng ý nghĩa nh- 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 gắn tệp với dòng nhập-xuất. Sau đó dùng toán tử nhập >> , toán tử xuất >> và các ph- ơng thức nhập, xuất để nhập, xuất dữ liệu ra dùng nhập-xuất vừa tạo (nh- 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 ph- ơng thức open để mở một tệp cụ thể (để đọc và ghi) và cho gắn với dòng nhập-xuất vừa xây dựng. Khi không cần làm việc với tệp này nữa, chúng ta có thể dùng ph- ơng thức close để chấm dứt mọi ràng buộc giữa dòng nhập-xuất và tệp. Sau đó có thể gắn dòng nhập-xuất với tệp khác. Theo cách này, có thể dùng một dòng nhập-xuất (đối t- ợng 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 ph- ơng thức sau (của lớp ostream) :

1. Ph- ơng 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. Ph- ơng 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. Ph- ơng thức

long teelp() ;

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 ph- ơng thức sau (của lớp istream):

4. Ph- ơng 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. Ph- ơng 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 vị trí hiện tại của con trỏ tệp

6. Ph- ơng thức

long teelg() ;

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 ph- ơng thức nêu trên.

13.4. Ví dụ

Ví dụ 1. Trong §12 đã viết ch- ơng trình xác định độ dài của tệp TC.EXE. D- ới đây là một ph- ơng án khác đơn giản hơn:

```
fstream  f("TC.EXE");
f.seekg(0,ios::end);
do_dai = f.teelg();
```

Ví dụ 2. Ch- ơng trình d- ới đây sẽ nhập danh sách n thí sinh từ bàn phím và ghi lên tệp. Sau đó đ- a 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 c- 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::outlios::inlios::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 << "\nHo 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 chẩy mà th- ờng xẩy ra nhiều điều trục trặc, chẳng hạn:

1. Mở một tệp để đọc nh- ng tệp không tồn tại.
2. Đọc dữ liệu nh- ng con trỏ tệp đã ở cuối tệp
3. Ghi dữ liệu nh- ng hết không gian đĩa (đĩa đầy).
4. Tạo tệp nh- ng đia hỏng, hoặc đĩa cấm ghi hoặc đĩa đầy.
5. Dùng tên tệp không hợp lệ
6. Định thực hiện một thao tác nh- ng tệp lại không đ- ợc mở ở mode phù hợp để thực hiện thao tác đó.

Tóm lại khi làm việc với tệp th- ờng gặp nhiều lỗi khác nhau, nếu không biết cách phát hiện xử lý thì ch- ơng trình sẽ dẫn đến rối loạn hoặc cho kết quả sai. Trong lớp ios của C++ có nhiều ph- ơng thức cho phép phát hiện lỗi khi làm việc với tệp. Đó là:

1. Ph- ơng thức

   int eof() ;

Nếu con trỏ tệp đã ở cuối tệp mà lại thực hiện một lệnh đọc dữ liệu thì ph- ơng thức eof() trả về giá trị khác không, trái lại ph- ơng thức có giá trị bằng 0.

2. Ph- ơng thức

   int fail() ;

Ph- ơng thức fail() trả về giá trị khác không nếu lần nhập xuất cuối cùng có lỗi, trái lại ph- ơng thức có giá trị bằng 0.

3. Ph- ơng thức

   int bad() ;

Ph- ơng thức bad() trả về giá trị khác không khi một phép nhập xuất không hợp lệ hoặc có lỗi mà ch- a phát hiện đ- ợc, trái lại ph- ơng thức có giá trị bằng 0.

4. Ph- ơng thức

   int good() ;

Ph- ơng thức good() trả về giá trị khác không nếu mọi việc đều tốt đẹp (không có lỗi nào xẩy ra). Khi có một lỗi nào đó thì ph- ơng thức có giá trị bằng 0.

5. Ph- ơng 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 ch- ơng 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 đ- ợc tệp hay không. Để làm điều đó, chúng ta có thể dùng đoạn ch- ơng 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 đ- ợc tệp << ten_tep ;
   exit(1) ;
  }
```

Ví dụ 3. Để xác định độ dài của tệp, có thể dùng ph- ơng 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 vùng lặp.

Thuật toán trên đ- ợc thể hiện bằng đoạn ch- ơng 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 ph- ơng 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 ph- ơng thức:

```
ostream & put(char) ;
```

+ Để đọc một ký tự từ tệp có thể dùng ph- ơng 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 ch- ơng 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 (ph- ơng thức eof() = 0) thì ghi lên tệp đích và lại tiếp tục đọc ký tự tiếp theo.

+ Nếu đọc không thành công (ph- ơng 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. Ch- ơng trình sao tệp d- ới 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 fnguon, fdich;
    cout << "\nTen tep nguon: " ; cin >> tep_nguon;
    cout << "\nTen tep dich: " ; cin >> tep_dich;
    fnguon.open(tep_nguon,ios::in | ios::binary);
    fdich.open(tep_dich,ios::out | ios::binary);
    if (fnguon.bad() || fdich.bad() )
      {
        cout << "\n Loi mo tep nguon hoac dich " ;
        getch();
        exit(1);
      }
    while(fnguon.get(ch),!fnguon.eof())
    fdich.put(ch) ;
    fnguon.close();
    fdich.close();
    cout << "\nHoan thanh" ;
    getch();
  }
```

15.3. Đọc, ghi một dãy ký tự theo kiểu nhị phân

+ Ph- ơng thức:

ostream & write(char *buf, int n) ;

sẽ xuất n ký tự (byte) chứa trong buf ra dòng xuất.

+ Ph- ơng thức:

istream & read(char *buf, int n) ;

sẽ nhập n ký tự (byte) từ dòng nhập và chứa vào buf.

+ Ph- ơng thức

int gcount

cho biết số ký tự thực sự đọc đ- ợc trong ph- ơng thức read.

Chú ý: Các ph- ơng thức write, read chỉ làm việc một cách chính xác trong kiểu nhập-xuất nhị phân.

D- ới đây là ch- ơng trình sao tệp sử dụng các ph- ơng thức write, read và gcount.

```
//CT7_16.CPP
// Sao tep dung write, read va gcount
#include <iostream.h>
#include <fstream.h>
#include <conio.h>
#include <stdlib.h>
void main()
  {
    clrscr();
    char tep_nguon[40], tep_dich[40] ;
    char buf[5000];
    int n;
    fstream fnguon, fdich;
    cout << "\nTen tep nguon: " ; cin >> tep_nguon;
    cout << "\nTen tep dich: " ; cin >> tep_dich;
    fnguon.open(tep_nguon,ios::in | ios::binary);
    fdich.open(tep_dich,ios::out | ios::binary);
    if (fnguon.bad() || fdich.bad() )
      {
        cout << "\n Loi mo tep nguon hoac dich " ;
        getch();
        exit(1);
      }
    while(fnguon.read(buf,5000),(n=fnguon.gcount()))
      fdich.write(buf,n) ;
    fnguon.close();
    fdich.close();
    cout << "\nHoan thanh" ;
    getch();
  }
```

§ 16. ĐỌC GHI ĐỒNG THỜI THEO KIỂU NHỊ PHÂN

Ch- ơng trình d- ới đây minh hoạ cách đọc ghi đồng thời trên tệp theo kiểu nhị phân. Ch- ơng trình sử dụng các ph- ơng thức write, read, các ph- ơng thức di chuyển con trỏ tệp và các ph- ơng thức kiểm tra lỗi. Ch- ơng 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 T□ ỢNG TRÊN TỆP

Trong các mục trên đã trình bầy 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 bầy cách xây dựng các toán tử dùng để đọc ghi các đối t- ợng của một lớp bất kỳ do ng- ời dùng định nghĩa.

Giả sử chúng ta muốn sử dụng các toán tử nhập xuất để đọc ghi các đối t- ợng của lớp TS. Khi đó ta đ- a vào các hàm bạn toán tử nhập xuất nh- 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 nh- nhị phân.

Ví dụ 1: Ghi theo kiểu văn bản

Ch- ơng trình d- ới đây minh hoạ cách xây dựng và sử dụng các toán tử nhập xuất đối t- ợng trên màn hình, bàn phím và tệp. Ch- ơng trình đ- a vào lớp TS (Thí sinh) và các hàm toán tử cho phép nhập xuất các đối t- ợng TS trên màn hình, bàn phím và tệp. Ch- ơng 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 t- ợng t1, t2, t3.

+ Ghi nội dung của 3 biến đối t- ợng t1, t2, t3 lên tệp TS.DL

+ Đọc các đối t- ợng từ tệp TS.DL và chứa vào 3 biến t4, t5, t6

+ In các biến đối t- ợng t4, t5, t6 ra màn hình

+ Chuyển con trỏ về đầu tệp, dùng chu trình while để lần l- ợt đọc các đối t- ợng từ tệp và in ra màn hình. Dùng ph- ơng thức eof để kiểm tra xem đã đọc hết dữ liệu hay ch- a.

```
//CT7_17.CPP
// Cac toan tu doc ghi doi tuong tren  Tep
#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 fstream& 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

Ch- ơng trình d- ới đây cũng có các chức năng nh- ch- ơng trình trong ví dụ 1 bên trên, nh- ng cách ghi đọc tệp theo kiểu nhị phân.

```
//CT7_19.CPP
// Cac toan tu doc ghi doi tuong tren Tep
```

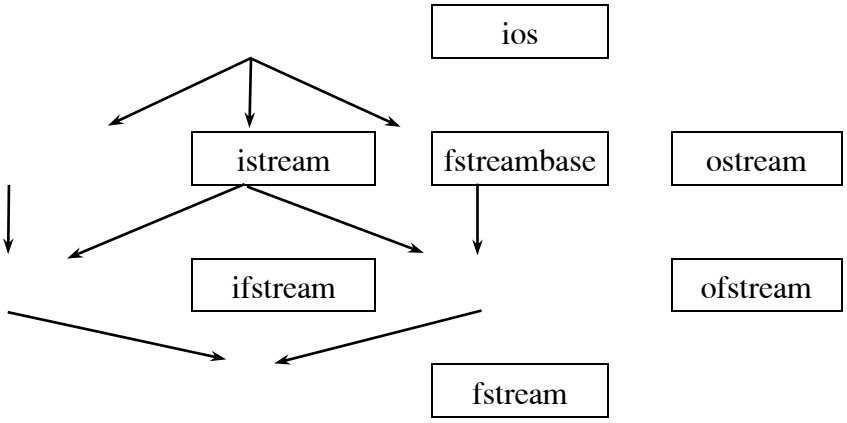
```
// Kieu nhi phan
#include <iostream.h>
#include <iomanip.h>
#include <fstream.h>
#include <conio.h>
#include <stdlib.h>
#include <ctype.h>
class TS
    {
      private:
          char ht[25];
          float td;
          static int size;
      public:
          friend ostream& operator<<(ostream& 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.DL",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 ph- ơng 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 ph- ơng 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 ph- ơng 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 ph- ơng thức của lớp fstreambase

void close()

18.6. Các ph- ơng 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 ph- ơng 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 ph- ơng 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)

CH□ ƠNG 8

ĐỒ HỌA

Trong ch- ơng này sẽ giới thiệu các hàm để vẽ các đ- ờng và hình cơ bản nh- đ- ờng tròn, cung elip, hình quạt, đ- ờng gẫy khúc, hình đa giác, đ- ờng thẳng, đ- ờng chữ nhật, hình chữ nhật, hình hộp chữ nhật, ... Ngoài ra còn đề cập tới các vấn đề rất lý thú khác nh- : xử lý văn bản trên màn hình đồ họa, cửa sổ và kỹ thuật tạo ảnh di động. Các hàm đồ họa đ- ợc khai báo trong 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 tr- ớc đến nay chúng ta chủ yếu làm việc với kiểu văn bản. Nghĩa là màn hình đ- ợc 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 đ- ợc phần cứng của máy PC ấn định tr- ớc và ta không thể nào thay đổi đ- ợc kích th- ớc, kiểu chữ.

Ở màn hình đồ họa, ta có thể xử lý đến từng chấm điểm (pixel) trên màn hình và do vậy muốn vẽ bất kỳ thứ gì cũng đ- ợc. Sự bài trí và số pixel trên màn hình đ- ợc 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 8-1 cho thấy các kiểu đồ họa và các tệp tin điều khiển chúng.

Ngoài các tệp có đuôi BGI chứa ch- ơng trình điều khiển đồ họa, TURBO C còn cung cấp các tệp tin đuôi CHR 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

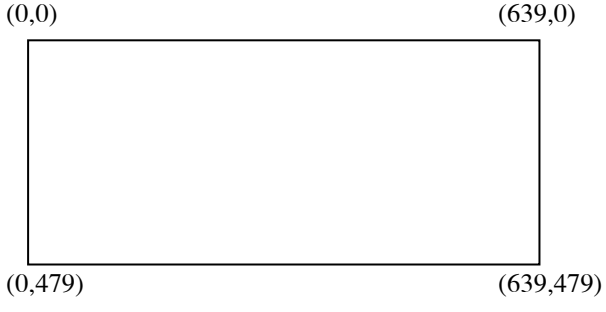
TRIP.CHR

Bảng 8-1. 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	IBMCGA, MCGA và các máy t- ơng thích
EGAVGA.BGI	IBM EGA, VGA và các máy t- ơng thích
HERC.BGI	Hercules monochrome và các máy t- ơng thích
IBM8514.BGI	IBM 8514 và các máy t- ơng thích
PC3270.BGI	IBM 3270 PC

Màn hình đồ họa gồm nhiều điểm ảnh đ- ợc sắp xếp trên các đ- ờng thẳng ngang và dọc. Điều này đúng cho tất cả các kiểu màn hình đồ họa của máy tính. Khác biệt chủ yếu giữa chúng là kích th- ớc và số các điểm ảnh. Trong kiểu CGA (độ phân giải thấp), điểm ảnh có kích th- ớc lớn, chiều ngang có 320 điểm ảnh, còn theo chiều dọc 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 l- ợng 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 nh- sau :



Hình 8.1. 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 d- ới phải 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 ch- ơng trình đồ họa th- ờng gồm các phần sau:

- Khởi động hệ thống đồ họa.
- Xác định mầu nền (mầu màn hình), mầu đ- ờng vẽ, mầu tô và kiểu (mẫu) tô.
- Vẽ, tô mầu các hình mà ta mong muốn.
- Các thao tác đồ họa khác nh- 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 mốt đồ họa sẽ sử dụng trong ch- ơng trình. Để làm điều này ta dùng hàm:

void initgraph(int *graphdriver, int *graphmode,char *driverpath);

trong đó: driverpath là đ- ờng dẫn của th- mục chứa các tệp tin điều khiển đồ họa, graphdriver, graphmode cho biết màn hình và mốt đồ họa sẽ sử dụng trong ch- ơng trình. Bảng 8-2 cho thấy 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 th- mục C: \TC, khi đó ta có thể khởi động hệ thống đồ họa nh- sau:

```
#include "graphics.h"
main()
  {
    int mh=EGA, mode= EGALO;
    initgraph(&mh, &mode, "C:\TC");
    . . .
  }
```

Bảng 8-2. 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)	EGAMONOHi (0)	640 x 350
VGA (9)	VGALO (0)	640 x 200
	VGAMED (1)	640 x 350
	VGAHI (2)	640 x 480

HERCMONO (7)	HERCMONOHI	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 mầu
	IBM8514HI (1)	1024 x 768, 256 mầu

Chú ý 1. Bảng 8-2 cho các tên 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 t- ơng ứng của chúng. Chẳng hạn các phép gán trong ví dụ 1 có thể viết theo một cách khác t- ơng đ- ơng nh- sau:

```
mh=3;
mode=0;
```

Chú ý 2. Bảng 8.2 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 640x 350, 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 đ- ợc phát hiện, giá trị số của nó đ- ợc gán cho biến graphdriver.

- Mode đồ họa ở độ phân giải cao nhất ứng với màn hình đang sử dụng cũng đ- ợc phát hiện và giá trị số của nó đ- ợc gán cho biến graphmode.

Nh- vậy việc dùng hằng số DETECT chẳng những có thể khởi động đ- ợc 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. Ch- ơng trình d- ới đây xác định kiểu màn hình đang sử dụng:

```
#include "graphics.h"
#include "stdio.h"
main()
  {
    int mh=0, mode= 0;
    initgraph(&mh, &mode, "");
    printf("\n Giá trị số của màn hình là: %d", mh);
    closegraph();
  }
```

Nếu ch- ơng 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 (nh- trong ví dụ 2) thì ch- ơng trình dịch sẽ tìm các tệp điều khiển đồ họa trên th- mục chủ.

§ 3. LỖI ĐỒ HỌA

Khi khởi động hệ thống đồ họa nếu máy không tìm thấy các ch- ơng trình điều khiển đồ họa thì sẽ phát sinh lỗi đồ họa và việc khởi động coi nh- không thành. Lỗi đồ họa còn phát sinh khi dùng các hàm đồ hoạ. Trong

mọi tr- ờng hợp, hàm graphresult cho biết có lỗi hay không lỗi và đó là lỗi gì. Bảng 8-3 cho các mã lỗi mà hàm này phát hiện đ- ợc. Ta có thể dùng hàm grapherrormsg với mã lỗi do hàm graphresult trả về để biết đ- ợc đó là lỗi gì, ví dụ:

```
int maloi;
maloi = graphresult();
printf("\nLỗi đồ họa là: %d", grapherrormsg(maloi));
```

Bảng 8-3. Các mã lỗi của Graphresult

Hằng	Trị	Lỗi phát hiện
grOk	0	Không có lỗi
grNoInitGraph	-1	Ch- a 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	V- ợt vùng RAM trong Scan fill
grNoFloodMem	-7	V- ợt vùng RAM trong flood fill
grFontNoFound	-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. MẦU VÀ MẪU

1. Để chọn mầu nền ta sử dụng hàm

```
void setbkcolor(int color);
```

2. Để chọn mầu đ□ờng vẽ ta dùng hàm

```
void setcolor(int color);
```

3. Để chọn mẫu (kiểu) tô và mầu tô ta dùng hàm

```
void setfillstyle(int pattern, int color);
```

Trong cả 3 tr- ờng hợp color xác định mã của mầu. Các giá trị khả dĩ của color cho trong bảng 8-4, pattern xác định mã của mẫu tô (xem bảng 8-5).

Mẫu tô và mầu tô sẽ đ- ợc sử dụng trong các hàm pieslice, fillpoly, bar, bar3d và floodfill (xem §5 d- ới đây).

4. Chọn giải mầu

Để thay đổi giải mầu đã đ- ợc định nghĩa trong bảng 8.4 ta dùng hàm

```
void setpalette(int colornum, int color);
```

Ví dụ câu lệnh

```
setpalette(0, LIGHTCYAN);
```

biến mầu đầu tiên trong bảng mầu thành xanh lơ nhạt. Các mầu khác không bị ảnh h- ởng.

Bảng 8-4. Các giá trị khả dĩ của color

Tên hằng	Giá trị số	Mầu 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
LIHGTGRAY	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 mầu hiện hành ta dùng hàm

void getpalette (struct palettetype *palette);

ở đây palettetype là kiểu đã định nghĩa tr- ớc nh- sau:

```
#define MAXCOLORS 15
struct palettetype
    {
      unsigned char size;
      unsigned char colors[MAXCOLORS+1];
    };
```

ở đây: size là số l- ợng mầu trong palette, colors là mảng chứa mầu với chỉ số mảng chạy từ 0 đến size - 1

Bảng 8-5. Các giá trị khả dĩ của pattern

Tên hằng	Giá trị số	Mô tả kiểu tô
EMPTY_FILL	0	Tô bằng mầu nền
SOLID_FILL	1	Tô bằng đ- ờng 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 đ- ờng gạch bóng nhạt
XHATCH_FILL	8	Tô bằng đ- ờng gạch bóng chữ thập
INTERLEAVE_FILL	9	Tô bằng đ- ờng đứt quãng
WIDE_DOT_FILL	10	Tô bằng dấu chấm th- a
CLOSE_DOT_FILL	11	Tô bằng dấu chấm mau

6. Hàm getcolor trả về mầu đã xác định tr- ớc đó bằng hàm setcolor.

7. Hàm getbkcolor trả về mầu đã xác định tr- ớc đó bằng hàm setbkcolor.

8. Hàm getmaxcolor trả về mã mầu cực đại thuộc giải mầu hiện đang có hiệu lực. Trên 256 K EGA, hàm getmaxcolor luôn cho giá trị 15.

§ 5. VẼ VÀ TÔ MẦU

Có thể chia các đ- ờng và hình thành bốn nhóm chính:

- Đ- ờng tròn và ellipse
- Đ- ờng gấp khúc và hình đa giác
- Đ- ờng thẳng
- Hình chữ nhật

A. Đ□ờng tròn và hình tròn

Nhóm này gồm cung tròn, đ- ờng 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 d- ới đây, góc tính theo độ và có giá trị từ 0 đến 360.

2. Đ□ờng tròn. Để vẽ một đ- ờng 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 đ- ờng 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. Ch- ơng trình d- ới đây sẽ vẽ: một cung tròn ở góc phần t- thứ nhất, một cung ellipse ở góc phần t- thứ ba, một đ- ờng tròn và một hình quạt quét từ 90 đến 360 độ.

```
#include <graphics.h>
main()
  {
    int mh, mode;
    // Khởi động đồ họa, màn hình EGA, mode EGALO
```

```
    mh=EGA;
    mode=EGALO;
    initgraph(&mh, &mode,"");
    // Mầu nền Green, mầu đ- ờng vẽ
    //White, mầu tô Red, kiểu tô SlashFill
    setbkcolor (GREEN);
    setcolor (WHITE);
    setfillstyle (SLASH_FILL, RED);
    // Vẽ: một cung tròn ở góc phần t-  thứ nhất,
    // một cung Ellipse ở góc phần t-  thứ ba,
    // một đ- ờng 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);
    // Kết thúc chế độ đồ họa
    closegraph();
  }
```

B. Đ□ờng gấp khúc và đa giác

5. Muốn vẽ một đ□ờng gấp khúc đi qua n điểm: (x1,y1), ... , (xn,yn) thì tr- ớc hết ta phải đ- a 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 đ- ợc một đ- ờng gấp khúc khép kín.

6. Giả sử a là mảng đã nói trong điểm 5, khi đó lời gọi hàm

```
    fillpoly(n, a);
```

sẽ vẽ và tô mầu một đa giác có đỉnh là các điểm

(x1, y1), ... ,(xn, yn).

Ví dụ 2. Ch- ơng trình d- ới đây sẽ vẽ một đ- ờng gấp khúc và hai hình tam giác.

```
#include <graphics.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, "");
    // Mầu nền CYAN, mầu đ- ờng vẽ
    // YELLOW, mầu tô MAGENTA, mẫu tô SolidFill
    setbkcolor (CYAN); Setcolor (YELLOW);
    setfillstyle (SOLID_FILL, MAGENTA);
    drawpoly (3, poly1); // Đ- ờng gấp khúc
```

```
    fillpoly (3, poly2); // Hình đa giác
    fillpoly(4, poly3); // Hình đa giác
    closegraph();
  }
```

C. Đ□ờng thẳng

7. Hàm

void line(int x1,int y1,int x2,int y2);

vẽ đ- ờng thẳng nối hai điểm (x1, y1) và (x2, y2) nh- ng không làm thay đổi vị trí con chạy.

8. Hàm

void lineto(int x,int y);

vẽ đ- ờng thẳng từ điểm hiện tại tới điểm (x, y) và chuyển con chạy đến điểm (x, y).

9. Hàm

void linerel(int dx,int dy);

vẽ một đ- ờng thẳng từ vị trí hiện tại (x, y) của con chạy đến điểm (x + dx,y + dy). Con chạy đ- ợc di chuyển đến vị trí mới.

10. Hàm

void moveto(int x,int y);

sẽ di chuyển con chạy tới vị trí (x, y).

Ví dụ 3. Ch- ơng trình d- ới đây tạo lên một đ- ờng gấp khúc bằng các đoạn thẳng. Đ- ờng gấp khúc đi qua các đỉnh: (20, 20), (620, 20), (620, 180), (20, 180) và (320, 100).

```
#include <graphics.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);
    closegraph();
  }
```

D. Hình chữ nhật

11. Hàm

void rectangle(int x1,int y1,int x2,int y2);

sẽ vẽ một đ- ờng 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 d- ới bên phải là (x2,y2).

12. Hàm

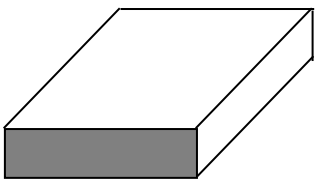
void bar(int x1,int y1,int x2,int y2);

sẽ vẽ và tô mầu một hình chữ nhật. Các giá trị x1,y1,x2 và y2 có ý nghĩa nh- đã nói trong điểm 11.

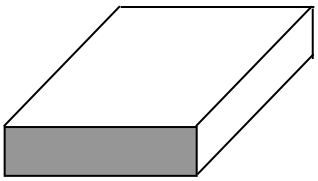
13. Hàm

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 (nh- đã nói trong điểm 12). Hình chữ nhật này đ- ợc tô mầu. Tham số depth ấn định số điểm ảnh trên bề sâu của khối 3 chiều. Tham số top có thể nhận trị 1 (TOPON) hay 0 (TOPOFF) và khối 3 chiều sẽ có nắp hay không nắp (xem hình vẽ).



TOPON



TOPOFF

Ví dụ 4. Ch- ơng trình d- ới đây sẽ vẽ một đ- ờng chữ nhật, một hình chữ nhật và một khối hộp chữ nhật có nắp.

```
#include <graphics.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);
    closegraph();
  }
```

§ 6. CHỌN KIỂU Đ□ ỜNG

1. Hàm

void setlinestyle(int linestyle,int pattern,int thickness);

tác động đến nét vẽ của các thủ tục line, lineto, rectange, drawpoly, circle,... Hàm này cho phép ta ấn định 3 yếu tố của đ- ờng thẳng là dạng, bề dầy và mẫu tự tạo.

+ Dạng đ- ờng do tham số linestyle khống chế. Sau đây là các giá trị khả dĩ của linestyle và dạng đ- ờng thẳng t- ơng ứng.

SOLID_LINE = 0	Nét liền
DOTTED_LINE = 1	Nét chấm
CENTER_LINE = 2	Nét chấm gạch
DASHED_LINE = 3	Nét gạch
USERBIT_LINE = 4	Mẫu tự tạo

+ Bề dầy do tham số thickness khống chế. Giá trị này có thể là:

NORM_WIDTH = 1	Bề dầy bình th- ờng
THICK_WIDTH = 3	Bề dầy gấp ba

+ Mẫu tự tạo: Nếu tham số thứ nhất là USERBIT_LINE thì ta có thể tạo ra mẫu đ- ờng thẳng bằng tham số pattern. Ví dụ xét đoạn ch- ơng trình:

```
int pattern= 0x1010;
```

```
setlinestyle(USERBIT_LINE, pattern, NORM_WIDTH);
line(0,0,100,200);
```

Giá trị của pattern trong hệ 16 là 0x1010 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 đ- ợc định nghĩa tr- ớc nh- sau:

```
struct linesettingstype
  {
    int linestyle;
    unsigned int upattern;
    int thickness;
  };
```

Ví dụ 1. Ch- ơng trình d- ới đây minh họa cách dùng các hàm setlinestyle và getlinesettings để vẽ đ- ờng thẳng.

```
// kiểu đ- ờng
#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);
    // L- u 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

```
void setwritemode( int writemode);
```

sẽ thiết lập kiểu thể hiện đ- ờng thẳng cho các hàm line, drawpoly, linerel, lineto, rectangle. Kiểu thể hiện do tham số writemode khống chế:

- Nếu writemode bằng COPY_PUT = 0, thì đ- ờng thẳng đ- ợc viết đè lên dòng đang có trên màn hình.

- Nếu writemode bằng XOR_PUT = 1, thì mầu của đ- ờng thẳng định vẽ sẽ kết hợp với mầu của từng chấm điểm của đ- ờng hiện có trên màn hình theo phép toán XOR (ch- ơng 3, §3) để tạo lên một đ- ờng thẳng mới.

Một ứng dụng của XOR_PUT là: Khi thiết lập kiểu writemode bằng XOR_PUT rồi vẽ lại đ- ờng thẳng cùng mầu thì sẽ xóa đ- ờng thẳng cũ và khôi phục trạng thái của màn hình.

Ch- ơng trình d- ới 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);
    tt: rectangle(x1,y1,x2,y2); // Vẽ hình chữ nhật
    if ( (x1+1)<(x2-1) && (y1+1)<(y2-1) )
      {
        rectangle(x1,y1,x2,y2); // xóa hình chữ nhật
        x1=x1+1; y1=y1+1; co hình chữ nhật
        x2=x2-1; y2=y2-1;
        goto tt;
      }
    setwritemode(COPY_PUT); // Trở về overwrite mode
    closegraph();
  }
```

§ 7. CỬA SỔ (VIEWPORT)

1. Viewport là một vùng chữ nhật trên màn hình đồ họa tựa nh- 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 d- ới bên phải. Bốn giá trị này phải thỏa mãn:

```
0 <= x1 <= x2
0 <= y1 <= 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 đ- ợc định nghĩa nh- 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 đ□a 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 d□ơng

Nhờ sử dụng Viewport có thể viết các ch- ơng trình đồ họa theo tọa độ âm d- ơng. Muốn vậy ta thiết lập viewport sao cho tâm tuyệt đối của 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 ch- ơng trình thực hiện công việc trên

```
int xc, yc;
xc= getmaxx()/2; yc= getmaxy()/2;
setviewport(xc, yc, getmaxx(), getmaxy(), 0);
```

Nh- thế màn hình sẽ đ- ợc chia làm 4 phần với tọa độ âm d- ơng nh- sau:

Phần t- trái trên: x âm, y âm

Phần t- trái d- ới: x âm, y d- ơng

Phần t- phải trên: x d- ơng, y âm

Phần t- phải d- ới: x d- ơng, y d- ơng

Ch- ơng trình d- ới đây vẽ đồ thị hàm sin(x) trong hệ trục tọa độ âm d- ơng. Hoành độ x lấy các giá trị từ -4*PI đến 4*PI. Trong ch- ơng 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()/,getmaxy()/2,
```

```
    getmaxx(),getmaxy(), 0);
    // Kẻ 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();
  }
```

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 ch- ơng trình d- ới đây. Ta sẽ đ- ợc đồ thị từ các đoạn thẳng rất ngắn ghép lại.

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();
  }
```

§ 8. TÔ ĐIỂM, TÔ MIỀN

1. Hàm

```
void putpixel(int x, int y, int color);
```

sẽ tô điểm (x,y) theo mầu xác định bởi color.

2. Hàm

```
unsigned getpixel(int x, int y);
```

sẽ trả về số hiệu mầu của điểm ảnh ở vị trí (x,y). Chú ý: nếu điểm này ch- a đ- ợc tô mầu bởi các hàm vẽ hoặc putpixel (mà chỉ mới đ- ợc tạo mầu nền bởi setbkcolor) thì hàm cho giá trị bằng 0. Vì vậy có thể dùng hàm này theo mẫu d- ới đây để xác định các nét vẽ trên màn hình đồ hoạ và vẽ ra giấy.

```
if (getpixel(x,y)!=0)
  {
    // Điểm (x,y) đ- ợc vẽ , đặt một chấm điểm ra giấy
  }
```

3. Tô miền

Để tô mầu 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 mầu.

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 đ- ờng (cong hoặc gấp khúc) khép kín mà mã mầu của nó bằng giá trị của border thì:

+ Miền giới hạn bởi đ- ờng kín sẽ đ- ợc tô mầu nếu điểm gieo (x,y) nằm bên trong miền này.

+ Nếu điểm gieo (x,y) nằm bên ngoài thì phần màn hình bên ngoài miền đóng nói trên đ- ợc tô màu.

b) Khi trên màn hình không có một đ- ờng nào nh- vậy, thì cả màn hình đ- ợc tô màu.

Ví dụ 1. Ch- ơng trình d- ới đây sẽ vẽ một đ- ờng tròn đỏ trên màn hình xanh. Tọa độ (x,y) của điểm gieo đ- ợc nạp vào từ bàn phím. Tùy thuộc vào giá trị cụ thể của x,y, ch- ơng trình sẽ tô mầu 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(" Toa do diem gieo x,y ");
    scanf("%d%d",&x,&y); flooddfill(x,y,RED);
  }
```

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 ch- ơng 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(seed);
     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

1. Hiển thị văn bản trên màn hình đồ hoạ

Hàm

void outtext (char *s);

sẽ hiện chuỗi ký tự (do s trỏ tới) tại vị trí hiện tại của con trỏ.

Hàm

void outtextxy(int x,int y,char *s);

sẽ hiện chuỗi ký tự (do s trỏ tới) tại ví 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 mốt đồ họa vẫn cho phép dùng hàm nhập dữ liệu scanf và các hàm bắt phím getch, kbhit.

2. Fonts

Nh- đã nói ở trên: Các Fonts nằm trong các tệp tin .CHR trên đĩa. Các Font này cho các kích th- ớc và kiểu chữ khác nhau sẽ hiện thị trên màn hình đồ hoạ bằng outtext hay 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)

Với direction là một trong hai hằng số:

HORIZ_DIR = 0

VERT_DIR = 1

Nếu direction là HORIZ_DIR, văn bản sẽ hiển thị theo h- ớng nằm ngang từ trái sang phải. Nếu direction là VERT_DIR, văn bản sẽ hiển thị theo chiều đứng từ d- ới lên trên.

Đối charsize là hệ số phóng to ký tự và có giá trị trong khoảng từ 1 đến 10.

- Nếu charsize = 1, ký tự đ- ợc thể hiện trong hình chữ nhật 8*8 pixel.
- Nếu charsize = 2, ký tự đ- ợc thể hiện trong hình chữ nhật 16*16 pixel.

. . .

- Nếu charsize = 10, ký tự đ- ợc thể hiện trong hình chữ nhật 80*80 pixel.

Cuối cùng là tham số font để chọn kiểu chữ và nhận một trong các hằng 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 ");
```

3. Vị trí hiển thị

Hàm settextjustify cho phép ấn định nơi hiển thị văn bản của outtext theo quan hệ với vị trí hiện tại của con chạy hay của outtextxy theo quan hệ với toạ độ (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 (Văn bản xuất hiện bên phải con chạy)

CENTER_TEXT = 1 (Chỉnh tâm văn bản theo vị trí con chạy)

RIGHT_TEXT = 2 (Văn bản xuất hiện bên trái con chạy)

Tham số Vert có thể là một trong các hằng số sau:

BOTTOM_TEXT = 0 (Văn bản xuất hiện phía trên con chạy)

CENTER_TEXT = 1 (Chỉnh tâm văn bản theo vị trí con chạy)

TOP_TEXT = 2 (Văn bản xuất hiện phía d- ới con chạy)

Ví dụ 3:

```
settextjustify(1,1);
outtextxy(100,100,"ABC");
```

Kết quả là điểm (100,100) sẽ nằm giữa chữ B.

4. Bề rộng và bề cao của văn bản

Hàm

```
void textheight (char *s);
```

trả về chiều cao (theo pixel) của chuỗi do s trỏ tới. Ví dụ nếu ký tự có kích th- ớc 8*8 thì

```
textheight ("H") = 8
```

Ví dụ 4: Đoạn ch- ơng trình d- ới đây sẽ cho hiện 5 dòng chữ.

```
#include <graphics.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

```
void textwidth(char *s);
```

sẽ dựa vào chiều dài của chuỗi, kích th- ớc Font chữ, hệ số khuyếch đại chữ để trả về bề rộng (theo pixel) của chuỗi do s trỏ tới.

Ví dụ 5: Trong ch- ơng trình d- ới 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

unsigned imagesize(int x1,int y1,int x2,int y2)

trả về số byte cần thiết để l- u trữ ảnh trong phạm vi hình chữ nhật (x1,y1,x2,y2).

2. Hàm

#include <alloc.h>

void *malloc(unsigned n);

trả về con trỏ trỏ tới một vùng nhớ n byte mới đ- ợc cấp phát.

3. Hàm

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 đ- ợc xác định bằng hàm imagesize.

4. Hàm

void putimage(int x,int y,void *bitmap,int copymode);

dùng để sao ảnh l- u 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 xi.
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 ng- ợc (phép NOT) với ảnh trong bitmap.

Nhận xét: Nếu dùng mode XOR_PUT để chép hình, rồi lặp lại đúng câu lệnh đó thì hình sẽ bị xoá và màn hình trở lại nh- cũ. Kỹ thuật này dùng để tạo lên các hình ảnh chuyển động.

Ví dụ 1: Ch- ơng trình d- ới đây minh hoạ cách dùng imagesize, malloc, getimage và putimage.

```
#include <alloc.h>
#include <graphics.h>
main()
  {
    int mh=0,mode=0;
    char *p;
    unsigend 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 đ- ợc cấp phát
    getimage (10,20,30,40,p);
    getch();
    cleardevice();
    putimage (100,100,p,COPY_PUT);
    getch();
```

```
    closegraph();
  }
```

5. Tảo ảnh di động

Nguyên tắc tạo ảnh di động giống nh- 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 nh- ng tại các vị trí khác nhau.

Cách 2: L- u ảnh vào một vùng nhớ rối đ- a ảnh ra màn hình tại các vị trí khác nhau.

B) Xóa ảnh

Cách 1: Dùng hàm cleardevice

Cách 2: Dùng hàm putimage (mode XOR_PUT) để xếp chồng lên ảnh cần xoá.

Cách 3: L- u trạng thái màn hình vào một chỗ nào đó. Vẽ một hình ảnh. Đ- a trạng thái cũ màn hình ra xếp đè lên ảnh vừa vẽ.

Kỹ thuật tạo ảnh di động đ- ợc minh hoạ trong các ch- ơng trình của §11.

§ 11. MỘT SỐ CH□ ƠNG TRÌNH ĐỒ HOẠ

Ch□ơng trình 1: Đầu tiên vẽ bầu trời đầu sao. Sau đó từng chùm pháo hoa đ- ợc 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 ch- ơng 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()));
      }
    // L- u 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
      {
        // Đ- a 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);
        //Xoá 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();
  }
```

Ch□ơng trình 2: Vẽ đồng hồ có 3 kim giờ, phút và giây. Đồng hồ chạy đúng theo giờ hệ thống. Muốn kết thúc ch- ơng trình bấm Enter.

```
// 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 dầy và mầu
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 giay
void ke_giay(unsigned giay)
  {
    int ddo;
    ddo = (90 - 6*giay);
    ke(ddo,rgiay,1,mgiay);
  }
// Hàm ke kim phut
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;
    gettime(&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;
    gettime(&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;
    gettime(&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);
  settextjustify(1,1); setcolor(MAGENTA);
  outtextxy(x0,y0+120,"IBM-JIMIKO");
  // Ghi chữ số
  settextstyle(3,0,3); settextjustify(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);
  // Ve kim gio,phut,giay
  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 tầu vũ trụ bay trên bầu trời đầy sao theo quỹ đạo ellipse. Trong khi tầu chuyển động thì các ngôi sao thay nhau nhấp nháy

```
// Tầu 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 ch- ơng trình
void tau_cd(void); // tầu chuyển động
void nhap_nhay_bt(void); // nhấp nháy bầu trời
void main(void); // hàm main
// Khai báo các biến mảng ngoài
int a,b,x,y,x0,y0,mh=0,mode=0,n,i;
float goc,xt,yt;
char *p;
int xx[1001],yy[1001];
// Hàm main
void main(void)
  {
    initgraph(&mh,&mode,"");
    if (graphresult()!=0) exit(1);
    // Vẽ tầu 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);
    // L- u ảnh của tầu 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 l- u 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 tầu
    goc= 2*M_PI + M_PI/2;
    x0= (getmaxx() - 42)/2;
    y0= (getmaxy() - 25)/2;
    a=x0; b=y0;
    // chu trình tầu 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 tầu 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 tầu 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 mầu 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 HOẠ

Hàm in_anh d- ới đây sẽ in ảnh trong miền chữ nhật (xt, yt, xd, yd) của màn hình đồ hoạ 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 th- ờng chon dd=1. Nếu muốn in rõ hơn ta chọn dd bằng 2 hay 3.

Trong hàm in_anh có dùng hàm tao_mau, nó đ- ợc mô tả nh- sau:

```
int tao_mau(int k,int x,int y);
```

Hàm này sẽ dò trên k chấm điểm theo chiều dọc bắt đầu từ toạ độ (x,y) trên màn hình để biết xem chấm điểm nào đã tô mầu. Hàm sẽ trả về một giá trị nguyên tạo bởi các bit 1 (ứng với điểm đã tô mầu) và 0 (ứng với điểm ch- a tô mầu).

Hàm in_anh sẽ dùng hàm tao_mau để duyệt trên miền chữ nhật (xt,yt,xd,yd). Mỗi lần duyệt sẽ nhận đ- ợc một mẫu các chấm điểm (giá trị nguyên) và mẫu này đ- ợc in ra giấy.

D- ới đây là nội dung của 2 hàm nói trên.

```
// in ảnh
#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(stdprn,"%c%c%c%c%c%c",13,27,'*',
                                                0,n1,n2); //LQ
                for (j=0;j < scot;++j)
                  {
                    if (kbhit())//bat phim
                      {
                        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,'@');
}
```

CH□ ƠNG 9

TRUY NHẬP TRỰC TIẾP VÀO BỘ NHỚ

Trong ch- ơng này trình bầy 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 nh- 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 nh- 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 nh- 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 nh- 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 nh- 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 đ- a 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 mầu hiển thị gọi là byte thuộc tính. Các bit của byte thuộc tính:

B7B6B5B4B3B2B1B0

đ- ợc 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 đ- ợc một số nguyên từ 0 đến 7 và biểu thị 8 mầu nền của ký tự.

+ Nhóm 3 gồm các bit B3, B2, B1 và B0. Các bit này chứa đ- ợc một số nguyên từ 0 đến 15 và biểu thị 16 mầu của ký tự.

2.2. Trang màn hình

Mỗi trang màn hình gồm 80x25 ký tự, do đó cần 80x25x2=4000 byte bộ nhớ. Thực tế mỗi trang màn hình đ- ợc phân bố 4096 = 0x1000 byte. Nh- vậy 4 trang màn hình đ- ợc phân bố nh- 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 đ- ợc 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 hoạ

Ví dụ sau dùng hàm pokeb để đ- a 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();
    //Hien thi 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();
    //Hien thi trang 0
```

```
    v.h.ah = 5 ; v.h.al = 0 ;
    int86(0x10,&v,&r);
    getch();
    //Hien thi 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_chỉ_thực)
unsigned FP_OFF(địa_chỉ_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 HOẠ

Ch□ơng trình 1. Ch- ơng trình minh hoạ cách truy nhập trực tiếp vào bộ nhớ màn hình có địa chỉ đầu là 0xB800:0. Ch- ơng 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 mầu có toạ độ góc trên-trái là (dongt, cott) và góc d- ới-phải là (dongd,cotd). Mầu 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 đ- a ra màn hình một dãy ký tự (chứa trong dãy) tại dòng dong, từ cột cotd đến cotc. Mầu nền cho bởi m_nen, mầu 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).

```
/*
ch- ơng trình minh hoạ cách truy nhập trực tiếp vào bộ
nhớ của màn hình
*/
#include "dos.h"
```

```
#include "conio.h"
void duarmh(char *day, int dong,I nt cotd, int cotc,I nt m_nen,  int m_chu);
void cuaso(int dongt,int cott,int dongd,int cotd,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;
    }
}
```

Ch□ơng trình 2. Biết địa chỉ của các thủ tục xử lý ngắt đ- ợc l- u trữ trong bộ nhớ từ địa chỉ 0000:0000 đến 0000:0x0400. Ch- ơng 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 đ- ợc tính từ 0, nh- ng n đ- ợc 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);
    }
  }
```

Ch□ơng trình 3. Ch- ơng trình minh hoạ 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 . Ch- ơng trình yêu cầu nhập mật khẩu. Nếu chọn đúng (bấm ABCD và Enter) thì ch- ơng 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 f;
    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 ;
```

```
    }
  if (sai_mat_khau)
    {
      cout << "\nSai mat khau, Khoi dong lai may";
      getch();
      khoi_dong_may();
    }
  else
    {
      cout << "\nDung mat khau, tiep tuc chuong trinh";
      getch();
    }
}
```

Chương trình 4. Chương trình minh hoạ cách dùng biến con trỏ để lấy dữ liệu về thời gian hệ thống chứa trong 4 byte bắt đầu từ địa chỉ 0:0x46C . Chương trình cũng minh hoạ cách truy nhập trực tiếp bộ nhớ màn hình văn bản (địa chỉ đầu là 0xB800:0) và cách bắt phím tổng quát. Chương trình sẽ in ra màn hình các chữ cái một cách ngẫu nhiên. Khi bấm phím F1 chương trình tạm dừng để thông báo thời gian. Để kết thúc chương trình bấm phím ESC.

```
#include <dos.h>
#include <conio.h>
#include <stdlib.h>
#define VT 132  // vi tri thong bao
//Dia chi vung nho man hinh
char far *p_mh = (char far*)MK_FP(0xB800,0) ;
//Dia chi 4 byte chua thoi gian
unsigned long far *t_time=(unsigned long far*)MK_FP(0,0x46C);
char buf_time[]={'T',47,'I',47,'M',47,'E',47,':', 47, 32, 47, 32, 47,
                 32, 47, 32, 47, 32, 47, 32, 47, 32, 47, 32, 47, 32, 47};
char buf_luu[28];
void thong_bao_thoi_gian()
  {
    //Luu trang thai man hinh
    for (int i=0; i<28; ++i)
    buf_luu[i]=p_mh[i];
    // Xac dinh gio, phut, giay
    int gio =  (int)(*t_time/65543) ;
    unsigned long  du =   *t_time%65543 ;
    int phut = (int)(du/1092);
    du = du%1092;
    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);
      putch(random(26)+65);
      delay(400);
    }
  }
```

MỤC LỤC

Trang

PHỤ LỤC 1

THỨ TỰ □U TIÊN CỦA CÁC PHÉP TOÁN

Các phép toán đ- ợc chia thành 16 nhóm. Các phép toán trong cùng nhóm có mực độ - u tiên nh- 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 hoàm (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 th- ớc 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 d- (Remainder - modulus)

4. Nhóm bốn

.* Gọi gián tiếp tới thành phần từ một biến đối t- ợng

->* Gọi gián tiếp tới thành phần từ một con trỏ đối t- ợng

5. Nhóm năm

+ Cộng (Binary plus)

- Trừ (Binary minus)

6. Nhóm sáu

<< Dịch trái (Shift left)

>> Dịch phải (Shift right)

7. Nhóm bẩy

< 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 m□ời

^ Phép hoặc loại trừ theo bit (Bitwise XOR)

11. Nhóm m□ời một

| Phép hoặc theo bit (Bitwise OR)

12. Nhóm m□ời hai

&& Phép và logic (Logical AND)

13. Nhóm m□ời ba

&& Phép hoặc logic (Logical OR)

14. Nhóm m□ời bốn

?: Toán tử điều kiện (a ? x : y means "if a then x, else y")

15. Nhóm m□ời 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 d- (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 m□ời 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 t- ợng

:: 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

PHỤ LỤC 3

BẢNG MÃ ASCII VÀ MÃ QUYÉT

1. Bảng mã ASCII

Bộ ký tự ASCII gồm 256 ký tự đ- ợc phân bố nh- sau:

+ 32 ký tự đầu tiên là các ký tự điều khiển không in đ- ợc nh- 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 nh- 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 th- ờng từ a đến z

L□u ý: Chữ th- ờng có mã ASCII lớn hơn 32 so với chữ hoa t- ơng 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 hoạ.

Bảng sau cho mã ASCII của 128 ký tự đầu tiên. Để nhận đ- ợc các ký tự đồ hoạ (có mã từ 128 đến 255) có thể dùng ch- ơng trình sau:

```
// In các ký tự đồ hoạ 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
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
MÃ (Số TT)	KÝ TỰ	MÃ (Số TT)	KÝ TỰ	MÃ (Số TT)	KÝ TỰ
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
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 đ- ợc gán một con số, gọi là mã scan, từ 1 đến 83. IBM PC AT đùng 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 đ- ợc nhấn thì bộ xử lý của bàn phím gửi cho CPU mã scan t- ơng ứng, khi nó đ- ợc nhả thì mã scan cộng thêm 80 hex sẽ đ- ợc 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 d- ới	trừ, gạch d- ới
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
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
Hex	**Thập phân**	**Phím của PC**	**Phím của PC-AT**
26	38	L	K
27	39	Chấm phẩy, :	L
28	40	Nháy	Chấm phẩy,:
29	41	Tidle	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	Phảy	N
34	52	Chấm	M
35	53	/,?	Phảy
36	54	Shift phải	Chấm
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
Hex	**Thập phân**	**Phím của PC**	**Phím của PC-AT**
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	Dấu 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
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
Hex	**Thập phân**	**Phím của PC**	**Phím của PC-AT**
66	102		Mũi tên phải
67	103		PgDn
68	104		Del
69	105		Sys
6A	106		*, PrtScr
6B	107		-
6C	108		+

PHỤ LỤC 4

HÀM VỚI ĐỐI SỐ BẤT ĐỊNH TRONG C

Trong các giáo trình C th- ờng chỉ h- ớng 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 nh- 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 l- ợng lẫn kiểu cách. Ví dụ trong câu lệnh:

```
printf("\n Tổng = %d " , 3+4+5) ;
```

có 2 tham số, nh- ng trong câu lệnh:

```
printf("\n Hà Nội" ) ;
```

chỉ có một tham số.

Nh- vậy cần phân biệt các khái niệm sau:

- Đối số cố định đ- ợc 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 đ- ợc khai báo bởi ba dấu chấm: bất định cả về số l- ợng và kiểu

- Tham số bất định (ứng với đối bất định) là một danh sách giá trị với số l- ợng và kiểu tuỳ ý (không xác định)

Trong phụ lục này sẽ trình bầy cách xây dựng các hàm với đối số bất định. Công cụ chủ yếu đ- ợc 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 đ- ợc 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
```

(Để 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 đ- ợc 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 nh- 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 tuỳ theo kiểu giá trị mà ta phải dùng phép chuyển kiểu thích hợp đối với con trỏ. Ví dụ sau minh hoạ 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;  // Đ- a số nguyên 12 vào danh sách
((int*)p)++ ;       // Chuyển sang phần tử tiếp theo
*((float*)p)  = 3.14; // Đ- a 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 bất định bao giờ cũng đặt sau cùng và đ- ợc khai báo bằng dấu ba chấm. Ví dụ ví dụ hàm

```
void f(int n, char *s, ...) ;
```

có 2 đối cố định là n, s và đối bất định.

+ Để nhận đ- ợc các tham số bất định trong lời gọi hàm ta cần l- u ý các điểm sau:

- Các tham số bất định chứa trong một danh sách. Để nhận đ- ợc 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 l- ợng 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 đ- ợc địa chỉ đầu danh sách, biết đ- ợc số l- ợng và kiểu của mỗi tham số , thì dễ dàng nhận đ- ợc giá trị các tham số để sử dụng trong thân hàm.

Ví dụ sau đây minh hoạ 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 nh- 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[1];
      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("ifsssffii", 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 ng- ời nghĩ hàm khai báo nh- sau

```
void  f();
```

là hàm không đối trong C. Trong C++ thì hiểu nh- thế là đúng, còn trong C thì đó là hàm có đối bất định (hàm không đối trong C khai báo nh- sau: f(void)). Do không có đối cố định nào cho biết về số l- ợng 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 không thuận tiện cho ng- ời dùng vì phải khai báo đúng tên biến toàn bộ và phải khởi gán giá trị cho nó tr- ớc khi gọi hàm. Ví dụ trình bầy một hàm chỉ có đối bất định dùng để tính max và min của các giá trị thực. Các tham số bất định đ- ợc đ- a vào theo trình tự sau: Địa chỉ chứa max, địa chỉ chứa min, các giá trị nguyên cần tính max, min. Ch- ơng 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;
     }
 }
```

PHỤ LỤC 5

TÓM TẮT CÁC HÀM CỦA TURBO C THEO THỨ TỰ ABC

1. _chmod	<io.h>	42. exp	<math.h>
2. _close	<io.h>	43. fabs	<math.h>
3. _creat	<io.h>	44. fclose	<stdio.h>
4. _open	<io.h>	45. fcloseall	<stdio.h>
5. abort	<process.h>	46. fcvt	<ctype.h>
6. abs	<stdlib.h>	47. feof	<stdio.h>
7. acos	<math.h>	48. ferror	<stdio.h>
8. arc	<graphics.h>	49. fflush	<stdio.h>
9. asin	<math.h>	50. fflushall	<stdio.h>
10. atan	<math.h>	51. fgetc	<stdio.h>
11. atan2	<math.h>	52. fgets	<stdio.h>
12. atof	<ctype.h>	53. fillpopy	<graphics.h>
13. atoi	<ctype.h>	54. findfirst	<dir.h>
14. atol	<ctype.h>	55. findnext	<dir.h>
15. bar	<graphics.h>	56. floodfill	<graphics.h>
16. bar3d	<graphics.h>	57. floor	<math.h>
17. cabs	<math.h>	58. fmode	<math.h>
18. calloc	<alloc.h>	59. fopen	<stdio.h>
19. ceil	<math.h>	60. FP_OFF	<dos.h>
20. chdir	<dir.h>	61. FP_SEG	<dos.h>
21. chmod	<io.h>	62. fprintf	<stdio.h>
22. circle	<graphics.h>	63. fprintf	<stdio.h>
23. cleardevive	<graphics.h>	64. fputc	<stdio.h>
24. clearviewport	<graphics.h>	65. fputs	<stdio.h>
25. close	<io.h>	66. fread	<stdio.h>
26. clreol	<conio.h>	67. free	<alloc.h>
27. clrscr	<conio.h>	68. fscanf	<stdio.h>
28. coreleft	<alloc.h>	69. fseek	<stdio.h>
29. cos	<math.h>	70. fteel	<stdio.h>
30. cosh	<math.h>	71. fwrite	<stdio.h>
31. cprintf	<conio.h>	72. gcvt	<ctype.h>
32. creat	<io.h>	73. geninterrupt	<dos.h>
33. cscanf	<conio.h>	74. getbkcolor	<graphics.h>
34. delay	<dos.h>	75. getc	<stdio.h>
35. delline	<conio.h>	76. getch	<conio.h>
36. disable	<dos.h>	77. getchar	<stdio.h>
37. drawpoly	<graphics.h>	78. getche	<conio.h>
38. ecvt	<ctype.h>	79. getcolor	<graphics.h>
39. ellipse	<graphics.h>	80. getcwd	<dir.h>
40. enable	<dos.h>	81. getdate	<time.h>
41. exit	<process.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. segread <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. strcspn <string.h>
196. strdup <string.h>
197. stricmp <string.h>
198. strlen <string.h>
199. strlwr <string.h>
200. strncat <string.h>
201. strncmp <string.h>
202. strncpy <string.h>
203. strnicmp <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>

PHỤ LỤC 6

PHÂN TÍCH, THIẾT KẾ VÀ LẬP TRÌNH H□ỚNG ĐỐI T□ỢNG

§ 1. PHÂN TÍCH H□ỚNG ĐỐI T□ỢNG

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 tr- ớc 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 ng- ời thiết kế Kim Tự Tháp có thể đ- ợc coi nh- những nhà thiết kế hệ thống, những kiến trúc s- đại tài, còn những ng- ời tổ chức vận chuyển nguyên vật liệu, huy động nhân công xây dựng Kim Tự Tháp, theo một nghĩa nào đó, chính là những ng- ời phân tích hệ thống. Từ giữa thế kỷ tr- ớc, các nhà t- bản, các doanh nghiệp muốn có lợi nhuận cao đã phải tiến hành nghiên cứu ph- ơng 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 ph- ơng 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, nh- ng giá phần mềm lại tăng. Nhất là phí tổn cho bảo trì để hệ thống đáp ứng đ- ợc nh- cầu của ng- ời 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 ph- ơng 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 ng- ời sử dụng, nắm đ- ợc các chức năng và cách phân rã hệ thống vật lý thành các đơn thể (module). Theo ph- ơng pháp truyền thống thì điều đó th- ờng đ- ợc thực hiện theo cách tiếp cận trên-xuống (top-down), sử dụng ph- ơng pháp phân tích có cấu trúc. Phân tích h- ớng đối t- ợng cho phép mô tả hệ thống gần với thế giới thực hơn, xác định rõ các đối t- ợng, trừu t- ợng hoá các yêu cầu để trên cơ sở đó xây dựng đ- ợc cấu trúc của hệ thống. Ph- ơng pháp h- ớng đối t- ợng giải quyết đ- ợc hố ngăn cách giữa phân tích và thiết kế hệ thống.

Trong mục này chúng ta đề cập đến các b- ớc cần thực hiện trong phân tích h- ớng đối t- ợng (PTHĐT). Thông qua ví dụ về phân tích hệ thống th- viện, chúng ta hình dung rõ hơn công việc xây dựng các đặc tả yêu cầu, mô tả đối t- ợng và cách xác định mối quan hệ giữa các lớp đối t- ợng trong hệ thống.

1.2. Các b□ớc thực hiện trong phân tích h□ớng đối t□ợng

Để xây dựng một hệ thống phần mềm thì phải giải quyết ba vấn đề chính sau:

+ Dữ liệu, lớp các đối t- ợng và cấu trúc của chúng.

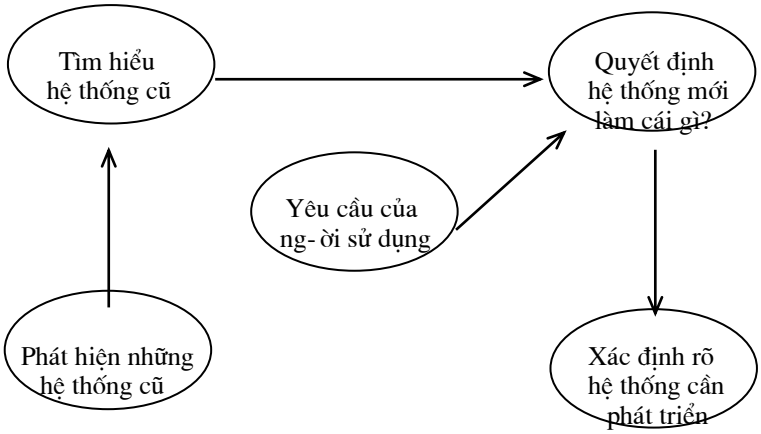
+ Những hành vi thể hiện các chức năng cục bộ, những quá trình trong hệ thống.

+ 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 th- ờng xuyên thay đổi. Do vậy phải phân tích kỹ bài toán, lựa chọn ph- ơng 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.

Ng- ời phân tích hệ thống là ng- ời 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 đ- ợc những ng- ời đầu t- , thiết kế và những ng- ời sử dụng hệ thống.

Nhiệm vụ của phân tích hệ thống là phải trả lời cho đ- ợc câu hỏi "Hệ thống làm cái gì?" và "Tại sao?". Để xác định đ- ợc bài toán và trả lời đ- ợc những câu hỏi nêu trên thì ng- ời 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 ch- a phải là hệ thống tin học hoá. Trên cơ sở nghiên cứu những hệ thống cũ, xác định rõ yêu cầu của ng- ời sử dụng để quyết định xem hệ thống cần xây dựng sẽ làm cái gì và hoạt động nh- thế nào. Quá trình đó đ- ợc mô tả nh- ở hình 1-1.



Hình 1-1. Mức độ bao quát thế giới thực

Trong các ph- ơng pháp truyền thống thì mô hình dòng dữ liệu đ- ợc mô tả thông qua sơ đồ dòng dữ liệu. Các quá trình trong hệ thống đ- ợc xác định thông qua việc phân rã chức năng top-down. Sơ đồ biến đổi trạng thái đ- ợc sử dụng để mô tả sự biến đổi thông tin và dòng điều khiển trong hệ thống. Ph- ơng pháp h- ớng đối t- ợng kết hợp hai ph- ơng diện dữ liệu với quá trình, gộp chung hành vi cục bộ với dữ liệu trong một đơn vị cấu trúc. Ph- ơng pháp phân tích h- ớng đối t- ợng cung cấp cho chúng ta công cụ đơn giản nh- ng đủ mạnh để xác định các đối t- ợng và xây dựng các đơn nguyên của hệ thống cần phát triển. Phân tích h- ớng đối t- ợng bao gồm các b- ớc sau:

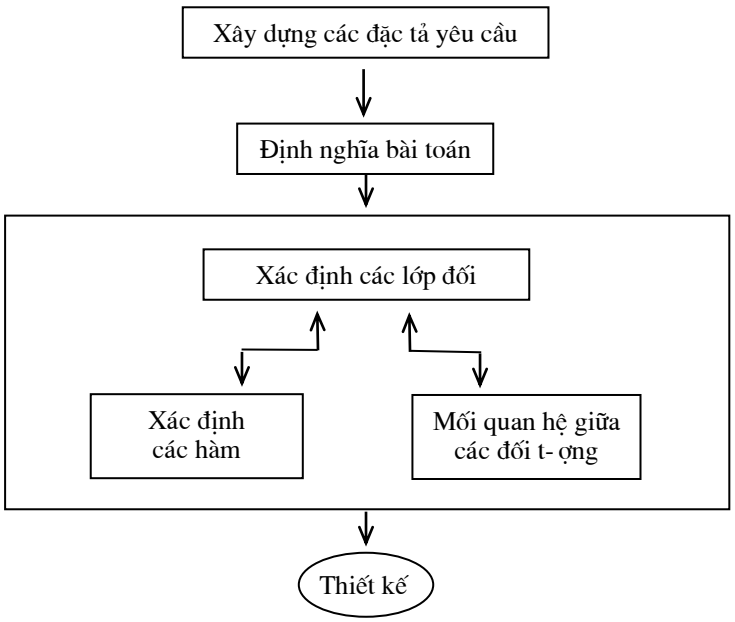
+ Tìm hiểu bài toán.

+ Xác định rõ các đặc tả yêu cầu của ng- ời sử dụng, của hệ thống phần mềm.

+ Xác định các đối t- ợng và các thuộc tính của chúng.

+ Xác định các hàm mà các đối t- ợng sẽ phải thực hiện (hành vi của các đối t- ợng).

+ Xác định mối quan hệ t- ơng tác giữa các đối t- ợng, các thông báo và sự truyền thông báo giữa các đối t- ợng.



Hình 1-2. Phân tích h- ớng đối t- ợng

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. Ng- ời phân tích phải gặp gỡ, trao đổi với những ng- ời đầu t- , những ng- ời sử dụng để biết rõ về chức năng, nhiệm vụ của hệ thống cần phát triển. Đồng thời ng- ời phân tích phải tìm hiểu, phát hiện những hệ thống cũ đã hoặc đang giải quyết những vấn đề t- ơng tự nh- những vấn dề mà hệ thống cần xử lý. Dựa vào những kinh nghiệm, kết quả phân tích những hệ thống cũ, những công việ mà hàng ngàyphải 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 s- phầm mềm để đảm bảo đ- a ra đ- ợc lời giải tin học (hệ thống thực hiện đ- ợc 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 s- 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 ng- ời 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 b- ớc 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 đ- ợc danh sách các yêu cầu của ng- ời sử dụng. Rõ ràng là ở đây cần có sự trao đổi, hiểu biết giữa ng- ời sử dụng và ng- ời phát triển hệ thống về những điều mà họ mong muốn. Dựa trên những yêu cầu của ng- ời sử dụng, ng- ời phát triển đ- a ra các đặc tả cho hệ thống. Ng- ời xây dựng hệ thống phải trả lời đ- ợc 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 đ- ợc mối quan hệ giữa đầu vào/ra (input/output), nghĩa là xác định đ- ợc 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 trắc nghiệm hệ thống để kiểm tra xem những nhiệm vụ đặt ra có đ- ợc hoàn thành hay không.

1.2.3. Xác định các đối t□ợng

Thông th- ờng các đối t- ợng sẽ đ- ợc xác định thông qua các thực thể trong thế giới thực và đ- ợc trừu t- ợng hoá thành các đối t- ợng trừu t- ợng. Để xác định các đối t- ợng 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, đ- ợc chuyển đến cho một hay nhiều quá trình xử lý và ng- ợc 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 đ- ợc gửi tới các kho dữ liệu. Trong sơ đồ dòng dữ liệu, một quá trình sẽ đ- ợc thực hiện khi có đủ các thông tin đầu vào (theo các đ- ờng có mũi tên dẫn đến quá trình đó).

Trong sơ đồ, hình tròn hoặc ellipse đ- ợc 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 nh- : "Xử lý đơn hàng", "Ghi nhận nguồn hàng" v.v... Ví dụ:

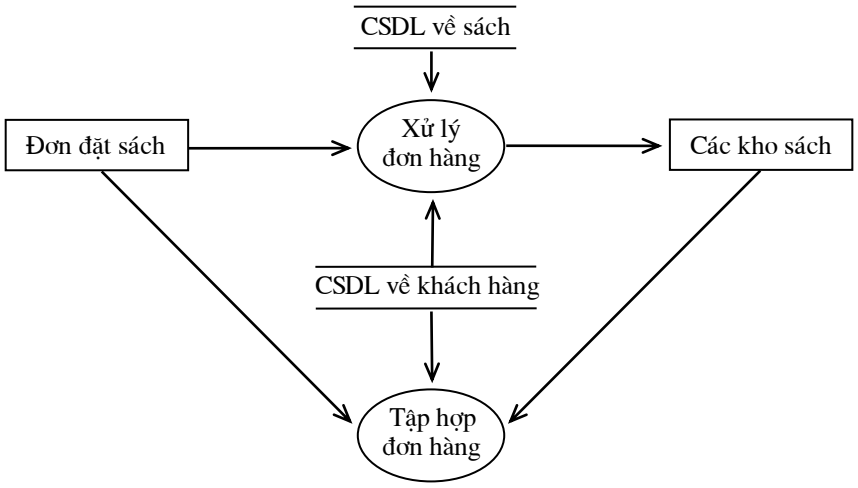
Chức năng quan trọng của quá trình là xử lý dữ liệu, biến đổi thông tin. Dòng dữ liệu đ- ợc biểu diễn bằng đ- ờng thẳng có mũi tên làm nhiệm vụ chuyển tải thông tin vào hoặc ra khỏi một quá trình. Mũi tên chỉ h- ớng của dòng thông tin. L- u ý là ở đây chỉ nói tới sự vận chuyển thông tin logic chứ không phải thông tin ở dạng vật lý. Dòng dữ liệu đ- ợc gắn với một tên nh- ng không nhất thiết phải là duy nhất. Các dòng dữ liệu, và tên đ- ợc gắn cho nó phải chỉ ra đ- ợc thông tin logic t- ơng ứng cho một quá trình. Trong sơ đồ dòng dữ liệu, các dữ liệu đ- ợc biểu diễn bằng hình chữ nhật có chứa tên của thông tin đ- ợcc cất giữ. Tên gắn với dữ liệu phải là danh từ. Ví dụ:

biểu diễn cho những thông tin về khách hàng đ- ợc 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 đ- ợc 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 đ- ợc biểu diễn trong các hình chữ nhật và các kho dữ liệu đ- ợc biểu diễn với tên gọi đặt trong hai đ- ờng thẳng song song. Kho dữ liệu biểu diễn cho một l- ợng lớn thông tin cần phải l- u trữ trong một thời gian dài, th- ờng là trong các tệp dữ liệu để cho nhiều ng- ời 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 t- ợng khác nhau. Quá trình "Xử lý đơn hàng", "Tập hợp đơn hàng" ở hình 1-4 đ- ợc làm mịn từ quá trình "Xử lý đơn hàng" ở hình 1-3 và có thể tiếp tục đ- ợc làm mịn thêm, mô tả những quá trình nh- 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

Ph□ơng 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 ng- ời phân tích là phân tích bài toán để xây dựng sơ đồ chức năng của hệ thống. Theo ph- ơng 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 đ- ợc đề cập kỹ trong cuốn "Phân tích, thiết kế và cài đặt hệ thông tin quản lý, Viện Tin học"). Các chức năng trong sơ đồ chức năng sẽ đ- ợc chuyển t- ơng 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 th- ờng đ- ợc sử dụng ở giai đoạn đầu của quá trình phân tích và đ- ợc dùng để vạch phạm vi hoạt động của hệ thống. Thông th- ờng sơ đồ ngữ cảnh đ- ợc xây dựng d- ới dạng tựa nh- sơ đồ chức năng, bao gồm một nút chính biểu diễn cho nhiệm vụ trung tâm của hệ thống, và toả 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 khả trong cuốn "Phân tích, thiết kế và cài đặt hệ thông 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 t- ợng. Trong sơ đồ dòng dữ liệu, những ô hình chữ nhật, ô có hai đ- ờng thẳng song song biểu diễn cho dữ liệu, kho dữ liệu có thể đ- ợc xem nh- là các đối t- ợng. L- u ý rằng không có sự t- ơng ứng 1-1 giữa những nút biểu diễn cho dữ liệu, kho dữ liệu trong sơ đồ dòng dữ liệu với các đối t- ợng. Một đối t- ợng có thể là đại diện của một hay nhiều nút dữ liệu, kho dữ liệu trong sơ đồ dòng dữ liệu tuỳ thuộc vào ngữ cảnh của vấn đề mà nó mô tả. Ví dụ trong hình 1-4. chúng ta sẽ có ba đối t- ợng: 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 t- ợng SACH vì cùng quản lý những thông tin về sách; đối t- ợng DON_HANG đ- ợc xác định từ nút "Đơn đặt sách" còn KHACH_HANG đ- ợc 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, ch- ơng, phần, tuỳ thuộc vào mức độ phức tạp của bài toán. Trong đó các đối t- ợng th- ờng đ- ợc mô tả bằng các danh từ. Danh từ th- ờng đ- ợc phân loại thành danh từ

riêng, danh từ chung, và các danh từ trừu t- ợng hoặc danh từ chỉ đại l- ợng.

Điều quan trọng cần l- u 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 t- ợng hoặc danh từ chỉ đại l- ợng trong ngữ cảnh khác. Cũng cần l- u ý là không phải tất cả các danh từ đều đ- ợc dùng để biểu diễn cho những đối t- ợng cần thiết cho hệ thống của chúng ta.

Bảng 1-1. Bảnh 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 t- ợng xác định	Nguyễn An, IBM, BBC
Danh từ trừu t- ợng hoặc đại l- ợng	Xác định chất l- ợng, đại l- ợng hoặc hoạt động ứng với danh từ	Thu nhập, l- ơng, giao thông

Tóm lại, chúng ta có thể sử dụng một trong hai công cụ trên để xác định danh sách các đối t- ợng của bài toán ứng dụng và sau đó tiếp tục:

1. Xác định những đối t- ợng chỉ nằm trong không gian bài toán, không gian lời giải, và những đối t- ợng nằm trong không gian bài toán nh- ng nằm ngoài giới hạn của hệ thống phần mềm.

2. Xây dựng các thuộc tính cho các đối t- ợng của không gian lời giải.

Sau khi đã xác định đ- ợc các đối t- ợng thì nhiệm vụ tiếp theo là xác định những thuộc tính mô tả các tính chất của từng lớp đối t- ợng. Ng- ời 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 t- ợng:

1. Từ những kinh nghiệm, tri thức của ng- ời 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 ng- ời 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 đ- ợc sử dụng th- ờng xuyên trong lĩnh vực đang nghiên cứu để chọn lọc ra những thuộc tính cho lớp các đối t- ợng 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 nh- 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 đ- ợc tiếp tục xem xét, bổ sung cho đầy đủ trong giai đoạn thiết kế.

Cần l- u ý là phải cân nhắc để đ- a ra đ- ợc những thuộc tính chung nhất, với những tên gọi đặc tr- ng cho từng lớp đối t- ợng.

1.2.4. Xác định các hàm

Để mô tả đầy đủ, chính xác các đối t- ợng 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 th- ờng, trong các câu mô tả thì động từ đ- ợc dùng để chỉ một hành động, sự xuất hiện, phân loại hay cấu thành của các đối t- ợng.

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 đ- ợc các hàm, còn động từ chỉ sự xuất hiện, so sánh giúp chúng ta xây dựng đ- ợc cấu trúc phân loại. Động từ sở hữu giúp cho việc xác định những cấu trúc cấu thành của các đối t- ợng. Cách thứ hai là dựa vào sơ đồ dòng dữ liệu để xác định các hàm, các chức năng đ- ợc biểu diễn bằng các hình tròn hoặc ellipse. Ví dụ, để mô tả cho hành vi của đối t- ợng trong lớp KHACH_HANG chúng ta phải xây dựng các hàm xử lý những thuộc tính đã xác định ở trên nh- 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 t ợng

B- ớc tiếp theo là xác định mối quan hệ giữa các đối t- ợng, nghĩa là sự trao đổi thông tin giữa chúng. Nh- 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 t- ợng KHACH_HANG sẽ phải gửi một thông báo (đơn đặt hàng) cho đối t- ợng DON_HANG. T- ơng tự, DON_HANG lại có quan hệ với SACH vì những cuốn sách sẽ đ- ợc bán cho khách hàng khi nhận đ- ợc các đơn đặt hàng. Quan hệ giữa các lớp đối t- ợng có thể có những kiểu khác nhau và đ- ợc phân thành ba kiểu sau:

1. Quan hệ một - một
2. Quan hệ một - nhiều
3. Quan hệ nhiều - nhiều

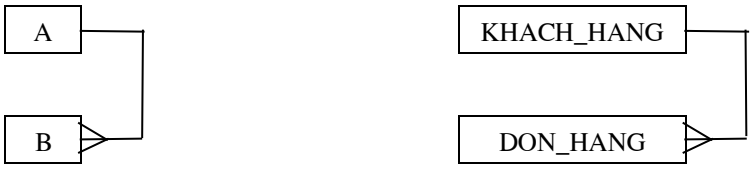
Quan hệ một - một: Hai lớp có quan hệ 1-1 nếu với mỗi đối t- ợng của lớp này có liên quan t- ơng ứng một đối t- ợng ở lớp kia và ng- ợc lại. Ví dụ: Hai lớp PHIEU_GHI và MAT_HANG có quan hệ 1-1. Mỗi phiếu ghi trong lớp PHIEU_GHI sẽ mô tả đúng một mặt hàng đ- ợc quản lý trong lớp MAT_HANG. Quan hệ này đ- ợc biểu diễn nh- 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 t- ợng trong lớp A có quan hệ với một hay nhiều đối t- ợng trong lớp B.
- Mỗi đối t- ợng trong lớp B có quan hệ với một đối t- ợng của lớp A.

Quan hệ một - nhiều đ- ợc biểu diễn nh- sau:



Hình 1-6. Quan hệ một - nhiều

Lớp KHACH_HANG có quan hệ một - nhiều với lớp DON_HANG vì một khách hàng có thể đặt nhiều đơn

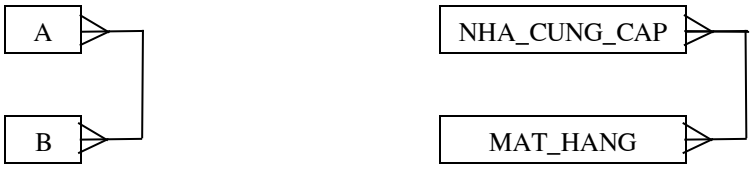
hàng khác nhau.

Quan hệ nhiều - nhiều: Hai lớp A và B có quan hệ nhều - nhiều nếu:

- Mỗi đối t- ợng A có sự t- ơng ứng với nhiều đối t- ợng trong B.

- Ng- ợc lại, mỗi đối t- ợng trong B cũng có t- ơng ứng với nhiều đối t- ợng trong A.

Quan hệ nhiều - nhiều đ- ợc biểu diễn nh- 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 ng- ợc lại, một mặt hàng cũng có thể đ- ợc sản xuất ở nhiều nơi.

Mô hình dữ liệu và những quan hệ giữa các đối t- ợng đ- ợc sử dụng không chỉ nh- một công cụ để phân tích, thiết kế mà còn nh- một ph- ơng 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 thông tin quản lý, Viện Tin học".

1.3. Ví dụ

1.3.1. Phát biểu bài toán

Tại khoa Tin học của một tr- ờng đại học có khoảng vài trăm cuốn sách để cho các cán bộ nhân viên trong khoa m- ợn. 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 ng- ời m- ợn và ai m- ợn.

Đây là ví dụ đơn giản, nh- ng cũng khá điển hình trong quá trình phân tích, thiết kế h- ớng đối t- ợng. Chúng ta cũng sẽ phân tích khả năng mở rộng, khả năng bảo trì hệ thống h- ớng đối t- ợng theo những yêu cầu mới cần mở rộng và phát triển hệ thống nhằm đáp ứng nhu cầu của ng- ời 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 nh- FoxPro, Access v.v... Ở đây chúng ta muốn thông qua ví dụ này làm rõ hơn những công việc, các b- ớc cần thực hiện trong quá trình phân tích h- ớng đối t- ợng.

1.3.2. Phân tích hệ thống

Bài toán nêu ở trên t- ơng đối rõ ràng. Yêu cầu xây dựng hệ thống phần mềm để quản lý các cuốn sách. Phân tích h- ớng đối t- ợng là việc lặp lại nhiều lần việc phân tích bài toán để xác định các đối t- ợng và xây dựng các đặc tả bài toán. Nh- ng phân tích h- ớng đối t- ợng còn có yếu tố tổng hợp. Việc thực hiện trừu t- ợng hoá những yêu cầu của ng- ời sử dụng và xác định rõ đ- ợc các đối t- ợng 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 đ- ợc để ở đâu đó, trong phòng làm việc hoặc đã cho ai m- ợn.

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 nh- hệ thống th- 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 t- ợng là: SACH và BAN_DOC. Trong các hệ thống th- viện, một cuốn sách có thể đ- ợc xác định thông các thuộc tính nh- : mã số th- 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 nh- 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 t- ợng trong lớp SACH. Lan Anh là tên một độc giả, là một đối t- ợng 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 m- ợn cuốn Peter Norton.

+ Hoang Trung đã m- ợn những cuốn sách nào?

+ Ai m- ợn cuốn sách Peter Hendenson?

+ Lan Anh trả cuốn Peter Norton.

+ Một cuốn sách mới đ- ợc bổ sung.

Đây chính là danh sách các yêu cầu của hệ thống.

Sau khi đã xác định đ- ợc các yêu cầu của bài toán và lớp các đối t- ợng, chúng ta thực hiện b- ớc tiếp theo là xác định các thuộc tính, hàm và mối quan hệ giữa các lớp đối t- ợng.

Sách là đối t- ợng đã đ- ợc xác định và đ- ợc biểu diễn nh- sau:

SACH
Peter Norton
...

Trong đó, SACH là tên gọi lớp tất cả các cuốn sách có trong th- viện. Peter Norton là tên một cuốn sách, một đối t- ợng cụ thể. T- ơng tự đối t- ợng bạn đọc sẽ đ- ợc mô tả nh- sau:

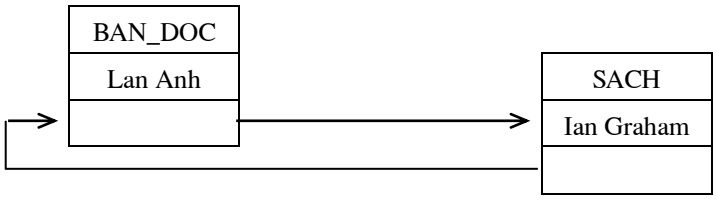
BAN_DOC
Lan Anh
...

Trong đó, BAN_DOC là lớp các độc giả và Lan Anh là một bạn đọc, một đối t- ợng trong lớp đó.

Trong hệ thống th- viện, những cuốn sách và độc giả sẽ đ- ợc mô tả t- ơng ứng là các đối t- ợng SACH và BAN_DOC. Bằng nhiều cách khác nhau, chúng ta phân tích và xác định đ- ợc các thuộc tính, các hàm cho hai lớp SACH, BAN_DOC.

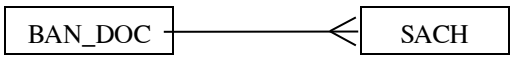
```
class SACH
   {
    Attribute //Thuộc tính
       Tac_gia      : Tác giả cuốn sách,
       Ten_sach     : Tên gọi hoặc tiêu đề của cuốn sách
       Xuat_ban     : Nhà, năm xuất bản
       Noi_giu      : Sách đã cho ai m- ợn hay có tại th- viện
    Function  //Hàm
       Nhap_sach() : Nhập các thông tin về cuốn sách vào
                     th- viện
       Cho_muon() : Xác định là sách đã cho m- ợn
       Hoan_tra()  : Sách đã đ- ợc trả lại th- viện
       Display()    : Hiện các thông tin về cuốn sách
   }
class BAN_DOC
   {
    Attribute //Thuộc tính
       Ho_ten       : Họ và tên ng- ời m- ợn sách,
       Dia_chi      : Địa chỉ, điện thoại của bạn đọc
       Ten_sach     : Tên những cuốn sách đã m- ợn
    Function  //Hàm
       Nhan_HT() : Nhập họ tên, địa chỉ của một bạn đọc
       Muon()       : Nhập thêm những cuốn sách mới m- ợn
       Tra()        : Trả sách cho th- 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 m- ợn cuốn Peter Norton có thể đ- ợc mô tả nh- đồ thị sau:



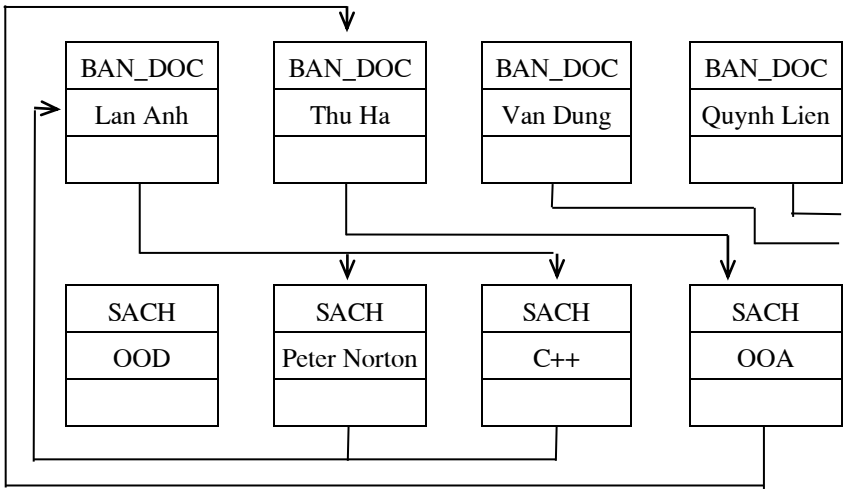
Hình 1-8. Lan Anh m- ợn cuốn sách Peter Norton

Một cuốn sách có thể cho nhiều nhất một ng- ời m- ợn, ng- ợc lại một ng- ời có thể m- ợn 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, mối quan hệ giữa hai lớp đối t- ợng đ- ợc thể hiện chi tiết hơn nh- hình 1-10.



Hình 1-10. Thể hiện mối quan hệ giữa hai lớp đối t- ợng SACH và BAN_DOC

Sơ đồ trên mô tả sự trao đổi thông tin giữa các lớp đối t- ợng. Lan Anh m- ợn hai cuốn Peter Norton và C++; Hoang Trung m- ợn cuốn OOA, cuốn sách OOD vẫn ch- a có ai m- ợn v.v...

Hệ thống của chúng ta luôn hoạt động bởi vì:

+ Th- ờng xuyên có ng- ời m- ợn sách.

+ Một cuốn sách đ- ợc trả lại hoặc đ- ợc 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 nh- ng đáp ứng đ- ợc yêu cầu đặt ra là quản lý đ- ợc sách và dễ dàng sửa đổi, bổ sung khi cần thiết.

§ 2. THIẾT KẾ H Ớ NG ĐỐI T Ợ NG

2.1. Giới thiệu chung

Mục này sẽ mô tả ph- ơng pháp thiết kế phần mềm dựa trên các đối t- ợng. Ph- ơng pháp h- ớng đối t- ợng (HĐT) nhằm che dấu thông tin ở mức tối đa và vì vậy hỗ trợ cho việc thiết kế những hệ thống với những cặp bộ giữa các thành phần là cực tiểu nh- ng mức độ cố kết hệ thống lại cao hơn cách tiếp cận chức năng. Chúng ta sẽ tập trung nghiên cứu các b- ớc cần thực hiện trong thiết kế h- ớng đối t- ợng 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 đ- ợc nhiều nhất l- ợng 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 h- ớng đối t- ợng (TKHĐT) là ph- ơng pháp thiết kế đ- ợc thực hiện theo nguyên lý che giấu thông tin. Khác với cách tiếp cận truyền thống (h- ớng chức năng) là nó xem hệ thống phần mềm (HTPM) là tập hợp các đối t- ợng t- ơng tác với nhau. Mỗi đối t- ợng làm việc với trạng thái (dữ liệu)

riêng của mình. Đối t- ợng, khái niệm cơ sở đã đ- ợc đề cập nhiều ở các phần tr- ớc 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 t- ợng. Một đối t- ợng không đ- ợc quyền truy nhập trực tiếp hoặc làm thay đổi trạng thái của đối t- ợng khác. Điều này dẫn đến là các đối t- ợng chỉ có thể trao đổi với nhau bằng các thông báo. Thông báo sẽ kích hoạt các hàm của đối t- ợng nhận thông tin t- ơng ứng. Hoạt động của cơ chế truyền thông báo giữa các đối t- ợng là dị bộ (không đồng bộ) vì vậy ch- ơng trình đ- ợc thiết kế theo h- ớng đối t- ợng có thể đ- ợc thực hiện song song hoặc tuần tự tuỳ theo ph- ơng 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 h- ớng đối t- ợng là ph- ơng 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 đ- ợc cài đặt bằng ngôn ngữ ch- a có đặc tính đó nh- Turbo C, hoặc Pascal, nh- ng tốt nhất là nên sử dụng những ngôn ngữ h- ớng đối t- ợng để cài đặt những thiết kế đ- ợc thực hiện theo cách tiếp cận h- ớng đối. Nhiều ngôn ngữ lập trình h- ớng đối t- ợng nh- Eiffel, Object Pascal, Smalltalk, C++ có những đặc tính h- ớng đối t- ợng hỗ trợ cho việc mô tả và thực hiện cài đặt trực tiếp những thiết kế h- ớng đối t- ợng hiệu quả hơn.

Tóm lại thiết kế h- ớng đối t- ợng có những - u điểm chính sau:

- Loại bỏ đ- ợc những miền dữ liệu dùng chung thông qua cơ chế trao đổi thông tin giữa các đối t- ợng bằng các thông báo.

- Các đối t- ợng đ- ợc 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 t- ợng chỉ sảy ra bên trong của đối t- ợng đó, không ảnh h- ởng đến các đối t- ợng khác. Mọi sự thay đổi trong thiết kế, trong hệ thống phần mềm chỉ sảy ra cục bộ đối với một số đối t- ợng liên quan. Điều này đảm bảo hệ thống có tính dễ mở rộng và dễ thích nghi, đáp ứng đ- ợc nhiều tính chất quan trọng của sản phẩm phần mềm.

Các đối t- ợng có thể đ- ợc 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 b□ớc thực hiện trong thiết kế h□ớng đối t□ợng

Nhiện vụ của thiết kế h- ớng đối t- ợng là xác định các đối t- ợng 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 d- ới - lên (bottom - up). Điều quan trọng là phải tạo ra đ- ợc cấu trúc phân cấp, xác định đ- ợc các lớp đối t- ợng trừu t- ợng và giảm thiểu đ- ợc sự trao đổi giữa các đối t- ợng. Ở đây chúng ta cũng đề cập đến khả năng sử dụng lại trong thiết kế, phân loại các đối t- ợng thành những hệ thống con trong cấu trúc phân cấp.

Cách tiếp cận TKHĐT gồm các b- ớc sau:

1. Xác định các lớp và các đối t- ợng, các thành phần cơ bản của lời giải.
2. Xây dựng các đặc tả cho các đối t- ợng, 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 ch- ơng trình chính.

Xác định các đối t- ợng trong không gian lời giải

Khi phân tích văn bản mô tả bài toán và các yêu cầu của ng- ời sử dụng, chúng ta xác định đ- ợc các thực thể, những đối t- ợng trong không gian bài toán. B- ớc tiếp theo là phân tích kỹ các đối t- ợng, xác định các thuộc tính và các hàm đặc tả cho từng đối t- ợng. Đồng thời xác định thêm những đối t- ợng mới xuất hiện trong không gian lời giải. Khi xây dựng các đặc tả cho đối t- ợng, chúng ta phải xác định đ- ợc các thuộc tính, dữ liệu mô tả trạng thái của đối t- ợng và các hàm mô tả hành vi của đối t- ợng. Thuộc tính là miền dữ liệu riêng của lớp đối t- ợng, 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 t- ợng trong cùng lớp và vùng dùng chung, cho phép những đối t- ợng trong các lớp có quan hệ với nhau đ- ợc quyền sử dụng. Các hàm (nhiều sách còn gọi là thủ tục, dịch vụ, ph- ơng thức) có thể dùng chung cho một số đối t- ợng. Quá trình xác định các hàm mô tả đối t- ợng (còn đ- ợc gọi là hàm thành phần của lớp) đ- ợc thực hiện nh- sau:

1. Nếu một hàm chỉ cần thiết cho một đối t- ợng thì hàm này chỉ hoạt động trong đối t- ợng yêu cầu.

2. Nếu có hai hoặc nhiều hơn đối t- ợng cần yêu cầu về một hàm thì cần phải xác định vùng hoạt động riêng của hàm trong các đối t- ợng đó.

3. Nếu có một hàm cần nhiều hơn một kiểu đối t- ợng (liên quan đến hai hoặc nhiều hơn các lớp đối t- ợng) 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 đ- ợc danh sách các hàm mô tả hành vi của các đối t- ợng. Đồng thời chúng ta cũng loại bỏ đ- ợc những d- thừa, những thành phần phụ không cần thiết trong cấu trúc và trong các đối t- ợng.

Sự phụ thuộc giữa các lớp

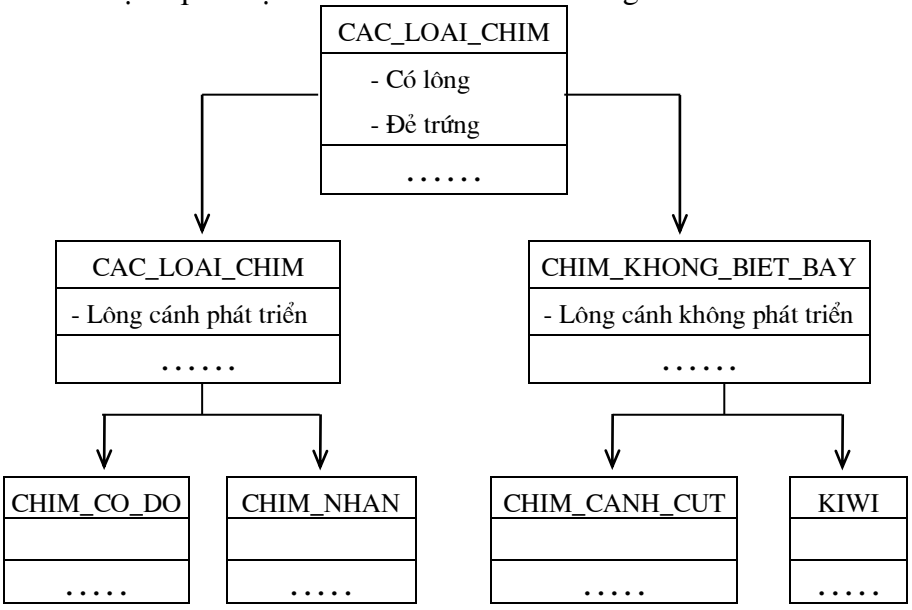
Mục tiêu của thiết kế là xây dựng cấu trúc phân cấp cho hệ thống. Do vậy, nhiệm vụ tiếp theo của chúng ta là xác định mối quan hệ giữa các lớp đối t- ợng cấu thành hệ thống. Lớp là tập hợp các đối t- ợng có chung một số thuộc tính, một số hàm vừa đủ để phân biệt với những lớp khác. Đối t- ợng là thể hiện của lớp. Trong thiết kế, khái niệm lớp đối t- ợng và đối t- ợng là hầu nh- không phân biệt, các lớp biểu diễn cho các đối t- ợng trong không gian lời giải. Để xây dựng đ- ợc 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 ph- ơng pháp h- ớng đối t- ợng là xây dựng các lớp đối t- ợng 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 t- ợng và khả năng sử dụng một số đặc tính kế thừa từ những lớp tr- ớc. Một lớp có thể sử dụng lại một số thuộc tính, hàm của một hay nhiều lớp đã đ- ợc định nghĩa tr- ớc. Lớp đ- ợc định nghĩa tr- ớc có những tính chất chung để cho những lớp khác có thể kế thừa đ- ợc gọi là lớp cơ sở và lớp kế thừa lớp cơ sở đ- ợc 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 đ- ợc 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 đ- ợc khai báo sử dụng chung mới đ- ợc 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 nh- có lông, đẻ chứng. Trong số các loài chim thì chúng ta cá thể phân làm hai loại: loại chin không bay đ- ợc 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 chứng 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 nh- trong hình 2-1.



Hình 2-1. Quan hệ kế thừa

Quan hệ thành phần: Đối t- ợng 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 t- ợng của những lớp khác.

Ví dụ: Một lớp A có thể sử dụng các lớp B và C theo các cách nh- 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**

Mối quan hệ của các lớp đóng vai trò quan trọng trong thiết kế ch- ơng trình sau này.

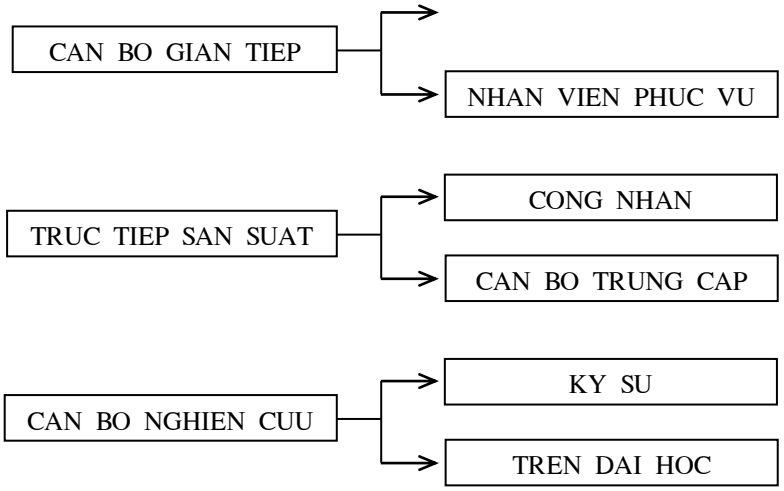
Tổ chức phân cấp các lớp (theo nghuyên lý tổng quát hoá)

Ở trên chúng ta đã nghiên cứu mối quan hệ mà chủ yếu là quan hệ kế thừa của các lớp đối t- ợng. 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 đ- ợc thiết kế tr- ớc.

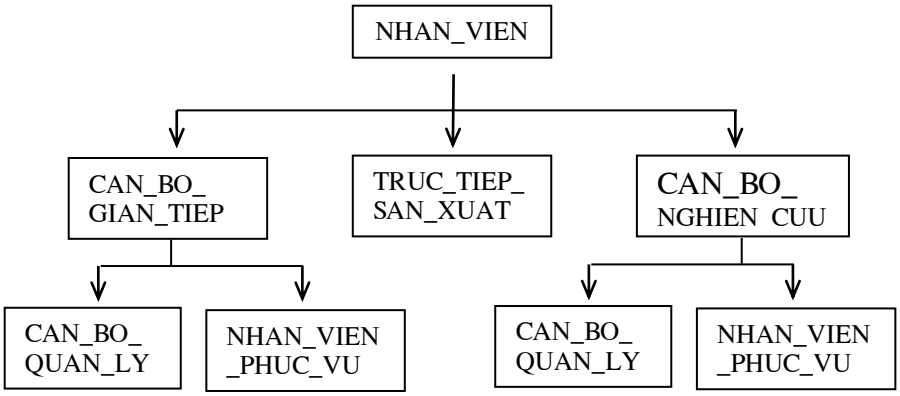
Tổ chức phân cấp các lớp là tập trung phân tích từng nhóm lớp có liên quan để xác định đ- ợc những thuộc tính, hàm chung nhất của cả nhóm và sau đó kết hợp chúng lại để tạo ra lớp mới. Lớp mới đ- ợc gọi là lớp trừu t- ợng và cũng là lớp cơ sở để cho các lớp trong cùng nhóm kế thừa. Lớp trừu t- ợng có thể có hoặc không có thể hiện là đối t- ợng trong không gian bài toán. Nó đ- ợc tạo ra thuần tuý bằng cách gộp những thuộc tính chung lại ở nhiều mức trừu t- ợng khác nhau cho đến khi cảm thấy chắc chắn không còn một lớp nào mới đ- ợc tạo ra nữa.

Ví dụ: Sau khi phân tích kỹ bài toán quản lý nhân sự của một xí nghiệp chúng ta có đ- ợc các lớp đối t- ợng: 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, [illegible] ông việc phục vụ, quản lý xí nghiệp để tạo ra một lớp mới là CAN_BO_GIAN_TIEP. T- ơng 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 đ- ợc thể hiện nh- sau:

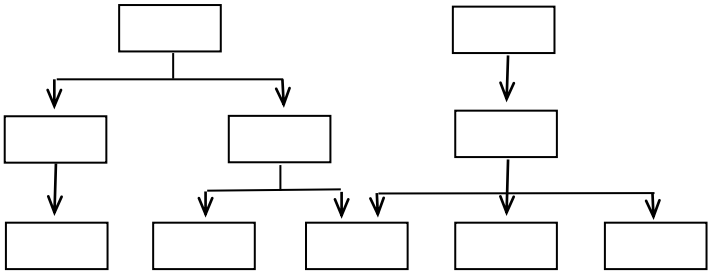


Các lớp mới đ- ợc xây dựng: CAN_BO_GIAN_TIEP, TRUC_TIEP_SAN_XUAT, CAN_BO_NGHIEN_CUU lại có những thuộc tính chung là cán bộ, nhân viên trong cùng một xí nghiệp nên có thể gộp những đặc tính chung lại để tạo ra một lớp trừu t- ợng mới là NHAN_VIEN. Những gì đã mô tả trong các lớp cơ sở thì không cần nêu lại trong các lớp dẫn xuất. Sau khi phân tích kỹ mối quan hệ giữa các đối t- ợng để thiết kế lớp, chúng ta sẽ đ- ợc cấu trúc phân cấp các lớp của hệ thống quản lý nhân sự nh- sau (theo nguyên lý tổng quát hoá).



Hình 2-2. Cấu trúc phân cấp các lớp (dạng cây)

Cấu trúc mà chúng ta thu đ- ợc ở hình 2-2 có dạng cấu trúc cây. Tuy nhiên trong thực tế có nhiều hệ thống trong đó các lớp trừu t- ợng 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) nh- hình 2-3.



Hình 2-3. Cấu trúc phân cấp dạng rừng cây

Thiết kế các lớp

Trong b- ớc phân tích, chúng ta đã xác định đ- ợc các lớp với các thuộc tính và tập tối thiểu các hàm chính thực hiện trên các thuộc tính mô tả đối t- ợng của lớp đó. Để xây dựng đ- ợc 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 t- ợng. Ngoài những hàm thể hiện đặc tính cơ bản của đối t- ợng 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 t- ợng đ- ợc tạo lập nh- thế nào?
   + Một đối t- ợng đ- ợc huỷ bỏ nh- thế nào?
2. Những hàm thực hiện cài đặt lớp
   + Những phép toán nào đ- ợc thực hiện trên dữ liệu kiểu lớp?
3. Những hàm truy nhập vào lớp
   + Làm thế nào chúng ta nhận đ- ợc thông tin về các biến nội bộ của một lớp.
4. Hàm xử lý lỗi
   + Làm thế nào xử lý đ- ợc các lỗi xuất hiện khi thao tác với các đối t- ợng.

Để thiết kế các lớp chúng ta cần phải biết rõ cách thức trao đổi thông tin giữa các đối t- ợng, 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 đ- ợc các 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 đ- ợc phân biệt trong quá trình thực hiện bởi từng đối t- ợng cụ thể.

3. Những th- viện các lớp nào đ- ợc sử dụng để thiết kế lớp.

Kết quả thiết kế lớp sẽ ảnh h- ởng rất lớp đến chất l- ợng 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 t- ợng của một lớp không đ- ợc gửi thông báo trực tiếp cho đối t- ợng của lớp khác.

3. Hàm đ- ợc khai báo là chung (public) chỉ khi nó đ- ợc sử dụng chung cho nhiều đối t- ợng 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. T- ơng tác giữa các lớp phải luôn luôn t- ờng minh.

7. Lớp dẫn xuất là một tr- ờng hợp của lớp cơ sở, đ- ợc 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 t- ợng của hệ thống.

Thông th- ờng khi thiết kế các lớp, hàm và ch- ơng trình chính, chúng ta nên sử dụng ngôn ngữ lập trình sẽ đ- ợc chọn để cài đặt (tốt nhất là chọn ngôn ngữ lập trình h- ớng đối t- ợng nh- C++) để mô tả.

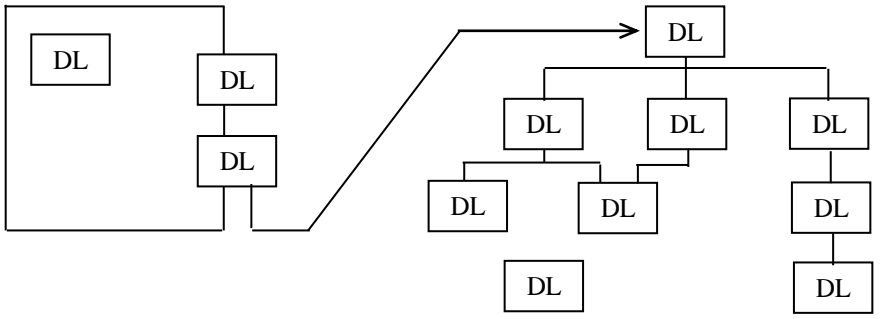
Thiết kế hàm thành phần

Đến giai đoạn này chúng ta đã xây dựng đ- ợc:

1. Các đối t- ợng và lớp
2. Các thành phần dữ liệu
3. Giao diện giữa các đối t- ợng
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 t- ợng. Các hàm này cũng giống nh- 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 t- ợng 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 nh- 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, nh- C, C++ đã đ- ợc xây dựng để hỗ trợ cho ph- ơng pháp mô tả thiết kế và lập trình có cấu trúc. Chúng ta có thể cài đặt các đơn thể đ- ợc cấu thành từ những cấu trúc tuần tự, tuyển chọn và vòng lặp. Có thể thiết kế các hàm thành phần theo ph- ơng 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 đ- ợc tổ hợp từ một trong ba cấu trúc cơ bản: cấu trúc tuần tự, tuyển chọn và vòng lặp.

Thiết kế ch- ơng trình chính

B- ớc cuối cùng trong khâu thiết kế hệ thống là xây dựng ch- ơng trình chính, giống nh- ch- ơng trình main() trong ngôn ngữ C++. Hệ thống đ- ợc bắt đầu và kết thúc tại ch- ơng trình chính. Do vậy nhiệm vụ của ch- ơng trình chính là:

1. Nhập dữ liệu từ ng- ời sử dụng.
2. Tạo ra các đối t- ợng theo định nghĩa các lớp.
3. Tổ chức thực hiện trao đổi thông tin giữa các đối t- ợng.
4. L- u trữ kết quả xử lý hoặc hiện lên màn hình, máy in, thiết bị ngoại vi theo yêu cầu ng- ời sử dụng.

Mọi hoạt động, xử lý trong quá trình thực hiện ch- ơng trình đều là kết quả của sự trao đổi, t- ơng tác giữa các đối t- ợng. Vì vậy nhiệm vụ chủ yếu của thiết kế ch- ơng trình là xác định thứ tự logic của quá trình trao đổi thông tin giữa các đối t- ợng trong hệ thống.

Ch- ơng trình chính liên quan trực tiếp đến ng- ời sử dụng. Vì vậy trong thiết kế chúng ta cũng cần đề cập đến thiết kế giao diện thân thiện với ng- ời sử dụng.

Thiết kế giao diện ng- ời máy

Có nhiều kiểu thiết kế giao diện đã đ- ợc tạo ra nhằm phục vụ cho ng- ời 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 ng- ời sử dụng, những ng- ời tham gia trực tiếp đối thoại với máy tính. Nhìn chung, các hệ giao diện với ng- ời sử dụng đều cần phải có những tính chất sau:

+ Dễ sử dụng: Giao diện thân thiện, dễ sử dụng ngay cả với những ng- ời sử dụng không có kinh nghiệm.

+ Dễ học: Các lệnh, thao tác hệ thống đ- ợc xây dựng theo những qui định chung, dễ tiếp thu và dễ nhớ.

+ Tốc độ thao tác nhanh, hợp lý: Các b- ớc thao tác, ấn nút trên bàn phím, con chuột nhanh gọn, tiện lợi cho ng- ời sử dụng. Thời gian thực hiện và trả lời trên máy tính nhanh và chính xác.

+ Đảm bảo an toàn: Kiểm soát đ- ợc các tình huống, những thao tác cố tình hay vô ý của ng- ời sử dụng đều đ- ợc xử lý tốt.

Dễ phát triển: Hệ thống có tính mở, có khả năng thay đổi, bổ sung theo yêu cầu của ng- ời sử dụng.

D- ới đây chúng ta sẽ đề cập đến một số kiểu thiết kế giao diện: dạng hỏi đáp, thực đơn và biểu t- ợng. Bạn đọc nào quan tâm sâu về thiết kế giao diện với ng- ời 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 ng- ời thiết. Bảng đối thoại th- ờng 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 tr- ờng hợp, có đầy đủ chú thích, h- ớng dẫn trợ giúp ng- ời sử dụng. Cách thiết kế này phù hợp với những ng- ời 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 th- ờng bảng thực đơn nên tổ chức thành cấu trúc phân cấp. Mỗi mục trong bảng thực đơn chính lại đ- ợc tổ chức thành bảng thực đơn con gồm một số mục để lựa chọn và đ- ợc thể hiện ở dạng màn hình cửa sổ (Window).

Thiết kế biểu t- ợng: Biểu t- ợng đ- ợc sử dụng để giới thiệu các chức năng của hệ thống trên màn hình. Mỗi chức năng đ- ợc biểu diễn bằng một biểu t- ợng (hình vẽ t- ơng ứng) sao cho dễ nhớ và dễ hình dung nhất.

Ở trên chúng ta đã nêu tất cả các b- ớc của quá trình thiết kế h- ớng đối t- ợng. Phần còn lại của ch- ơng 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 ch- ơng 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 đ- ợc hai đối t- ợng là: SINH_VIEN và MON_HOC. T- ơng ứng với các đối t- ợng này trong thiết kế là các lớp SINH_VIEN và MON_HOC. Vì đối t- ợng là thể hiện của lớp, nên hai lớp SINH_VIEN và MON_HOC xác định hai loại đối t- ợng cơ bản trong hệ thống theo dõi kết quả học tập mà giáo viên cần phải phát triển.

B- ớc tiếp theo trong thiết kế là mô tả chi tiết các lớp đối t- ợng và mối quan hệ giữa chúng. Theo yêu cầu của giáo viên (ng- ời sử dụng) thì mỗi đối t- ợng trong lớp SINH_VIEN đ- ợc 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 t- ợng 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 t- ợng. Cuối cùng chúng ta có danh sách các thuộc tính mô tả lớp SINH_VIEN và MON_HOC nh- 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 đ- ợc các đối t- ợng, lớp và danh sách các thuộc tính mô tả đối t- ợng, b- ớc tiếp theo là xác định các hàm thành phần của các lớp. Hàm thành phần mô tả hành vi của đối t- ợng bao gồm:

+ Truy nhập vào thành phần dữ liệu của đối t- ợng

+ Cho phép cập nhật, thay đổi dữ liệu thành phần

+ Kiểm tra dữ liệu theo nhiều cách khá 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 nh- ng cùng liên quan đến một loại dữ liệu. Ng- ời thiết kế các lớp muốn xây dựng các class đáp ứng đ- ợc mọi nhu cầu của ng- ời lập trình (khách hàng). Quan hệ giữa ng- ời phát triển với ng- ời lập trình, cũng giống nh- quan hệ chủ hàng - khách hàng, dẫn đến tr- ờng hợp là chủ hàng mong sao có nhiều ng- ời sử dụng nhất, nghĩa là lớp sẽ đ- ợc xây dựng với một số l- ợng lớn các hàm thành phần. Nh- ng 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 đ- ợc những hàm thành phần cơ bản, đặc tr- ng 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 t- ợng sinh viên: SINH_VIEN()

b. Hàm huỷ bỏ đối t- ợng 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 t- ợng 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 t- ợng của một lớp th- ờng có cùng tên của lớp đối t- ợng đó và đ- ợc gọi là cấu tử, còn hàm huỷ bỏ một đối t- ợng cũng cùng tên nh- ng có dấu "~" đứng tr- ớc đ- ợc gọi là huỷ tử. Hai hàm này sẽ đ- ợc gọi thực hiện một cách tự động khi cần thiết. Hàm cấu tử đ- ợc gọi khi cần định nghĩa, tạo lập một đối t- ợng, còn hàm huỷ tử đ- ợc gọi khi ch- ơng trình thoát ra khỏi phạm vi mà đối t- ợng đó đ- ợc định nghĩa. Một lớp có thể có nhiều hàm cấu tử và chúng đ- ợc phân biệt bởi danh sách và kiểu của các tham biến. Nh- ng chỉ có duy nhất một hàm huỷ tử 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 ng- ời sử dụng.

Theo cách tiếp cận truyền thống thì chúng ta nghĩ ngay đến việc dùng cấu trúc mảng một chiều để l- u trữ nhiệt độ mà ng- ời 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 đ- ợc thiết kế dựa theo thuật toán sau:

1. Lần l- ợt xem xét các phòng I=1 . . 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. Ng- ợc lại nếu nhiệt độ trong phòng làm việc đã đạt đ- ợc nh- nhiệt độ ng- ời 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 h- ớng đối t- ợng. Nhiệm vụ tr- ớc tiên của chúng ta là xác định các đối t- ợng thành phần cơ sở của hệ thống điều hoà nhiệt độ. Phân tích bài toán chúng ta thấy ở đây có hai lớp đối t- ợng:

1. Đối t- ợng PHONG
2. Đối t- ợng DIEU_HOA

Trong mỗi phòng: nhiệt độ hiện thời đ- ợc 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 tuỳ thuộc vào thông báo của PHONG: nhiệt độ trong phòng có cần phải thay đổi hay không.

Sau khi xác định đ- ợc các lớp, các thuộc tính mô tả trạng thái của các đối t- ợng và mối quan hệ giữa hai lớp đối t- ợng chúng ta cần xác định các hàm thành phần của chúng.

Đối với lớp PHONG chúng ta có:

1. Hàm khởi tạo đối t- ợng: PHONG()
2. Hàm xoá bỏ đối t- ợng: ~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 t- ợng: DIEU_HOA()
2. Hàm xoá bỏ đối t- ợng: ~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 t- ơng ứng 1-1 giữa số phòng và máy điều hoà nh- điều kiện ban đầu bài toán đặt ra. Trong tr- ờng hợp cần thay đổi, ví dụ nh- đối với hội tr- ờng 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 nh- trên thì thiết kế theo cách tiếp cận thứ nhất phải thay đổi ít nhiều, ch- ơng trình gần nh- phải viết lại toàn bộ vì chúng ta đã giả thiết rằng mỗi phòng có một máy điều hoà. Nh- ng cách tiếp cận thứ hai (thiết kế h- ớng đối t- ợng) thì chỉ cần sửa đổi ở những đối t- ợng liên quan. Để sửa đổi hệ thống theo yêu cầu thì hàm DONG_MO() của đối t- ợng 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 t- ợng PHONG dùng chung một DIEU_HOA cần đ- ợc thiết kế lại sao cho xác định đ- ợc 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 t□ợng thuỷ văn

Nội dung bài toán: Hệ thống gồm nhiều trạm đo số liệu và thu nhận thông tin dữ liệu về khí t- ợng thuỷ 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, h- ớng gió; áp suất và l- u l- ợng m- a.

Một trong những - u điểm của cách tiếp cận h- ớng đối t- ợng là có thể tiến hành thiết kế ngay khi ch- a 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 ch- a cho biết về tần số thu thập dữ liệu cũng nh- cách xử lý dữ liệu, nh- ng chúng ta có thể tiến hành ngay xây dựng thiết kế cho hệ thống.

Tr- ớc tiên chúng ta cần xác định những thành phần cơ bản của hệ thống, nghĩa là các thực thể sẽ t- ơng ứng với khái niệm đối t- ợng của hệ thống phần mềm mà chúng ta cần xây dựng. Nói chung, việc xác định các đối t- ợng đ- ợc thực hiện theo cách làm mịn dần trong quá trình thiết kế. Hệ thống mà chúng ta xây dựng ở đây là hệ khí t- ợng thuỷ văn sẽ có cả đối t- ợng "cứng" lẫn đối t- ợng "mềm". Đối t- ợng "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 t- ợng này đ- ợc nhúng vào hệ thống phần mềm để điều khiển các phần cứng t- ơng ứng. Ng- ợc lại, đối t- ợng "mềm" là những đối t- ợng chỉ t- ơng tác với các đối t- ợng khác trong hệ thống, nghĩa là giúp các đối t- ợng "cứng" trao đổi thông tin với nhau. Phân tích kỹ bài toán chúng ta có thể xác định đ- ợc các đối t- ợng "cứng" nh- sau:

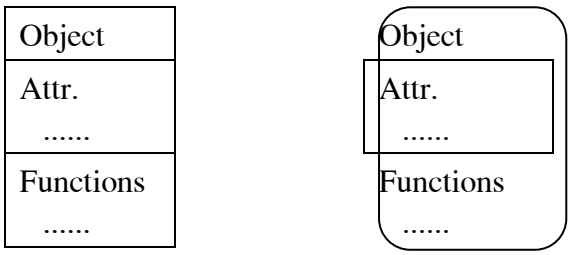
+ Trạm khí t- ợng thuỷ văn
+ Máy đo nhiệt độ khí quyển
+ Máy đo nhiệt độ lòng đất
+ Máy đo sức gió
+ Máy đo h- ớng gió
+ Máy đo áp suất khi quyển
+ Máy đo l- u l- ợng m- a
+ Đồng hồ (xác định thời gian thu nhận dữ liệu và truyền thông báo)
+ Modem (truyền và nhận thông báo từ máy tính trung tâm)

Nhiệm vụ của từng đối t- ợng đ- ợc mô tả nh- trong Bảng 2.1.

Bảng 3.1. Xác định công việc của các đối t- ợng "cứng"

Đối t□ợng	**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 h- ớng gió, tính theo độ đo góc
May_DLA	Evaluate	Đo áp suất của không khí
May_DLM	Evaluate	Xác định l- ợng n- ớc m- a kể từ lúc Reset lại
	Reset	Đặt lại máy đo, xoá 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 trng tâm

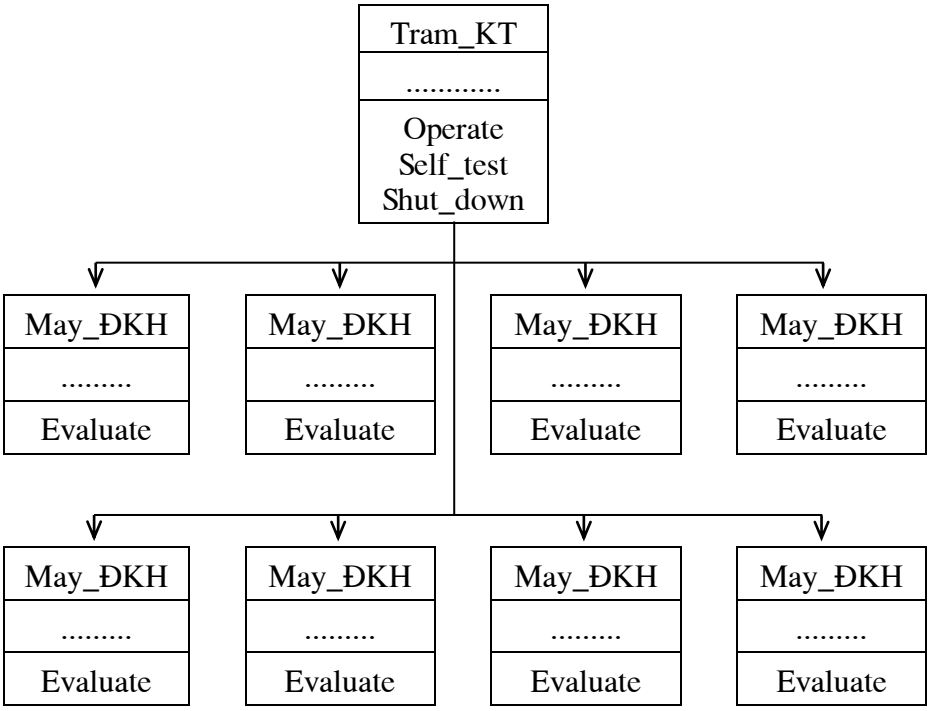
Để dễ theo dõi, chúng ta nên sử dụng sơ đồ khối để mô tả thiết kế. Ở trên chúng ta đã xác định là trong hệ thống sẽ có hai loại đối t- ợng: cứng và mềm. Những đối t- ợng đó sẽ đ- ợc mô tả nh- sau:



a) Đối t- ợng cứng b) Đối t- ợng mềm

Các đối t- ợng "cứng" có quan hệ với nhau trong hệ thống nh- trong hình 2-6. Đối t- ợng TRAM_KT thu nhận các thông tin từ các đối t- ợng 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 l- u ý ở đây là những đối t- ợng "cứng" không thể trao đổi thông tin trực tiếp với nhau mà phải thông qua các đối t- ợng "mềm".

B- ớc tiếp theo là xác định các đối t- ợng "mềm". Để làm đ- ợc 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 ng- ời sử dụng hệ thống khí t- ợng thuỷ văn để xác định bài toán, chúng ta có thêm những mô tả nh- sau:



Hình 2-6. Cấu trúc phân cấp của phần cứng

"Tất cả các tham số về thời tiết, ngoại trừ l- ợng m- a sẽ đ- ợc 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 đ- ợc xử lý để tính giá trị trung bình, giá trị cực đại, cực tiểu và trong 1 giờ đó, l- ợng m- a cũng đ- ợc ghi nhận. Các h- ớng gió đ- ợc đo theo độ đo góc α *(*$\alpha < 15^0$*) cũng sẽ đ- ợc ghi nhận. Tất cả những thông số đó sẽ đ- ợc tính toán sơ bộ và đ- ợc 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 t- ợng "mềm" sau:

+ SUC_ GIO Xác định tốc độ của gió
+ H_GIO Xác định h- ớng gió
+ ND_KH Đo nhiệt độ không khí
+ ND_LD Đo nhiệt độ lòng đất
+ L_MUA Đo l- u l- ợng n- ớc m- a
+ A_SUAT Đo áp suất khí quyển

Những đối t- ợng này cung cấp thông tin cho một đối t- ợng làm nhiệm vụ tổng hợp dữ liệu là:

+ TH_DL Tạo ra các bản ghi thông tin từ các đối t- ợng con, kiển tra, truyền và nhận thông báo.

Chức năng và nhiệm vụ của các đối t- ợng trên đ- ợc mô tả chi tiết trong Bảng 2-2.

Bảng 2-2. Mô tả công việc của các đối t- ợng mềm

Đối t□ợng	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 h- ớng gió
	Variances	Danh sách h- ớng gió lớn hơn α
	Mean	Xác định độ đo trung bình
DN_KH	Collect	Ghi nhận nhiệt độ không khí theo chu kỳ
	Max	Xác định giá trị cực đại của nhiệt độ
	Min	Xác định giá trị cực tiểu của nhiệt độ
	Mean	Xác định giá trị trung bình của nhiệt độ
DN_LD	Collect	Ghi nhận nhiệt độ của lòng đất
	Max	Xác định giá trị cực đại
	Min	Xác định giá trị cực tiểu
	Mean	Xác định độ đo trung bình
L_MUA	Collect	Ghi nhận l- ợng n- ớc m- a
	Max	Xác định giá trị cực đại
	Min	Xác định giá trị cực tiểu
	Mean	Xác định độ đo trung bình
A_SUAT	Collect	Đo đo áp suất không khí
	Max	Xác định giá trị cực đại
	Min	Xác định giá trị cực tiểu
	Mean	Xác định độ đo trung bình

Nhiệm vụ tiếp theo của thiết kế h- ớng đối t- ợng 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 t- ợng.

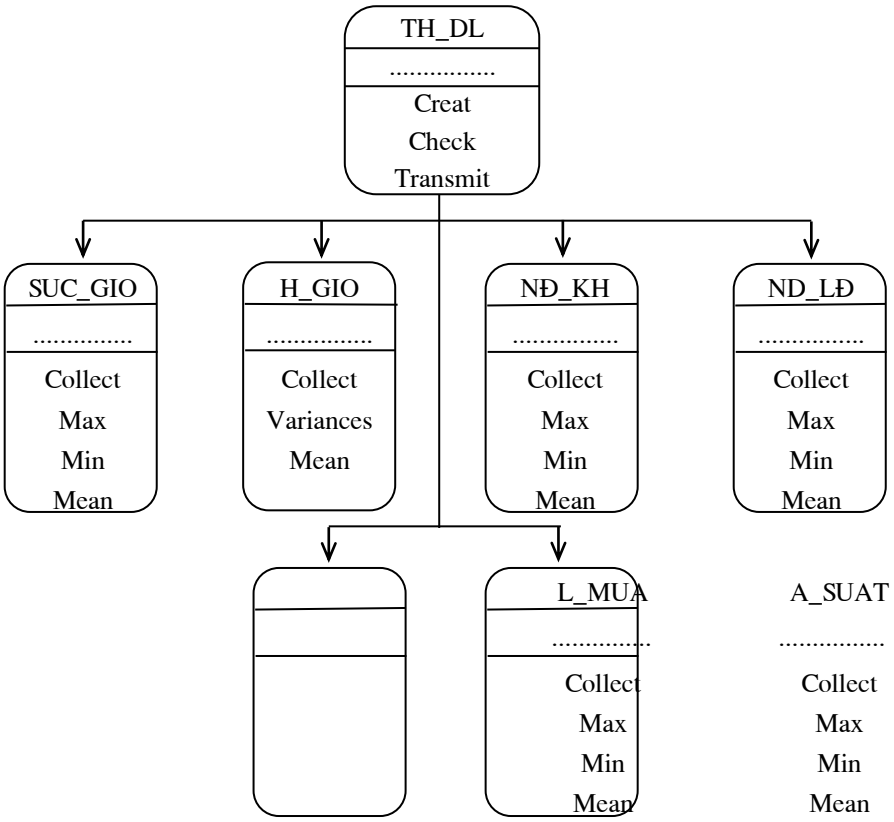
Tr- ớc tiên chúng ta xét lớp TH_DL. Lớp này làm nhiệm vụ tổng hợp dữ liệu, tạo ra các record dữ liệu nhận từ các đối t- ợng con về nhiệt độ, sức, h- ớng gió, l- ợng n- ớc m- a, áp suất khí quyển, kiểm tra tính hợp lý của dữ liệu, truyền và nhận thông báo từ trung tâm. L- u ý rằng, TH_DL không cần biết gì về cách thu nhập dữ liệu của các đối t- ợng con. Trong C++ chúng ta mô tả TH_DL nh- sau:

```
class TH_DL
    {
      private:
          ... // Khai báo các biến l- u trữ nhiệt độ, sức gió
          ... // h- ớng gió, áp suất, l- ợng m- a...
      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
    }
```

T- ơng 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 đ- ợc mô tả nh- sau:

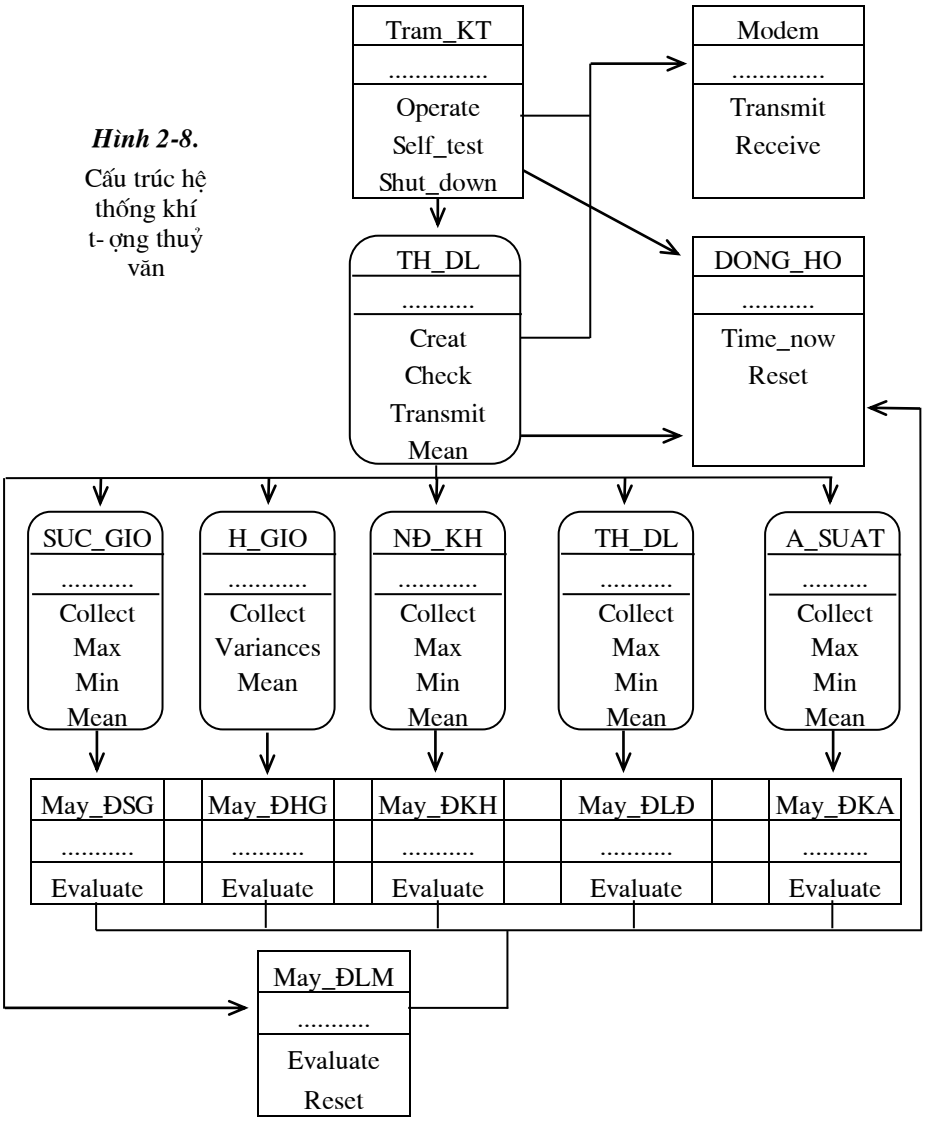
```
class ND_KH
   {
    private:
       float t[60];  // Bảng số liệu đo nhiệt độ của không khí
                     // ghi đ- ợc trong một chu kỳ 1 giờ
    public:
       float  *Collect(); // Thu nhận dữ liệu từ đối t- ợng "cứng"
       float  Max(); // Tính giá trị cực đại của nhiệt độ đo đ- ợc
       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 đ- ợc mô tả nh- trong hình 2-7.



Hình 2-7. Mô tả quan hệ của các lớp đối t- ợng "mềm"

Ở trên chúng ta đã khẳng định: các đối t- ợng "cứng" muốn trao đổi thông báo với nhau phải sử dụng các giao diện là các đối t- ợng "mềm". Dựa vào các chức năng, nhiệm vụ của từng lớp đối t- ợng và nhiệm vụ chung của hệ thống chúng ta có thể đ- a ra thiết kế tổng thể cho hệ thống khí t- ợng thuỷ văn nh- sau:



Hình 2-8.

Cấu trúc hệ thống khí t- ợng thuỷ văn

§ 3. LẬP TRÌNH H□ ỚNG ĐỐI T□ ỢNG

3.1. Giới thiệu

Với lập trình có cấu trúc, những hệ thống lớn, phức hợp, thì độ phức tạp của ch- ơng 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 đ- ợc xử lý trong nhiều thủ tục bên trong ch- ơng 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 đ- ợc sử dụng. Một nh- ợc điểm nữa của lập trình có cấu trúc là khi có nhiều ng- ời tham gia xây dựng ch- ơng trình, mỗi ng- ời đ- ợc giao viết một số hàm riêng biệt nh- ng lại có thể sử dụng chung dữ liệu. Khi có nhu cầu cần thay đổi dữ liệu sẽ ảnh h- ởng rất lớn đến công việc của nhiều ng- ời.

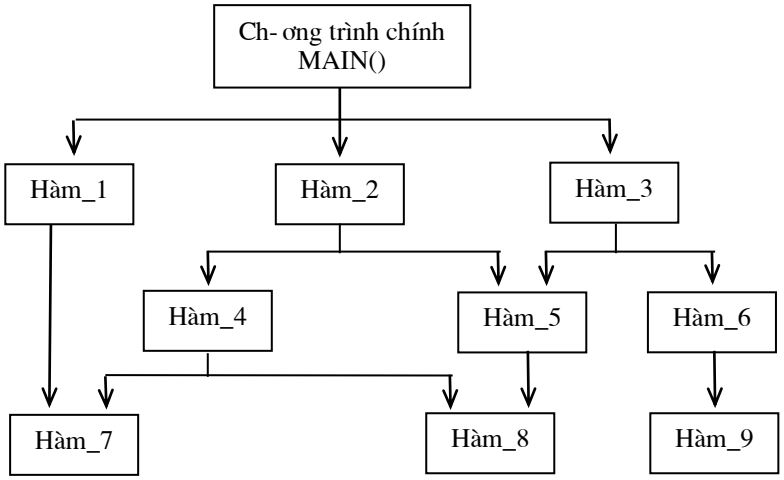
Lập trình h- ớng đối t- ợng dựa trên nền tảng là các đối t- ợng. Đối t- ợng đ- ợc xây dựng trên cơ sở gắn cấu trúc dữ liệu với các phép toán sẽ thể hiện đ- ợc đúng cách mà chúng ta suy nghĩ, bao quát về thế giới thực. Chẳng hạn, ô tô có bánh xe, di chuyển đ- ợc và h- ớng của nó thay đổi đ- ợc bằng cách thay đổi tay lái. T- ơng tự, cây là loại thực vật có thân gỗ và lá. Cây không phải là ô tô và những gì thực hiện đ- ợc với ô tô sẽ không làm đ- ợc với cây.

Lập trình h- ớng đối t- ợng cho phép chúng ta kết hợp những tri thức bao quát về các quá trình với những khái niệm trừu t- ợng đ- ợc sử dụng trong máy tính. Ch- ơng trình h- ớng đối t- ợng xác định chính xác các đặc tr- ng và hành vi của các kiểu dữ liệu, trong đó có thể tạo ra những đối t- ợng mới đ- ợc xây dựng từ những khuôn khổ có sẵn hay tổ hợp để tạo ra những đặc tr- ng mới.

Trong ch- ơng này chúng ta sẽ giới thiệu những khái niệm cơ bản và các b- ớc cần thực hiện trong lập trình h- ớng đối t- ợng.

3.2. Lập trình h□ớng thủ tục (chức năng)

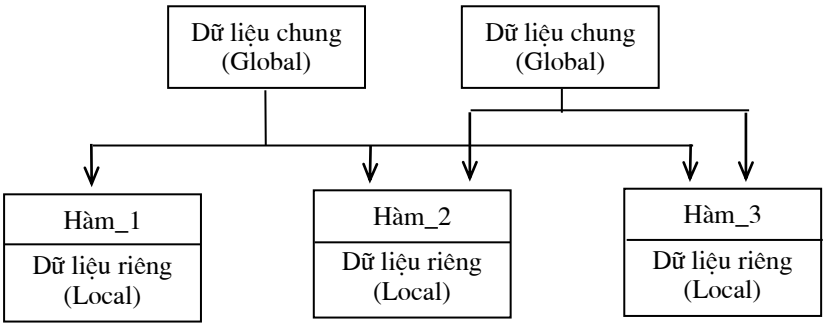
Những ngôn ngữ lập trình bậc cao truyền thống nh- COBOL, FOTRAN, PASCAL, C v.v..., đ- ợc gọi chung là ngôn ngữ lập trình h- ớng thủ tục. Theo cách tiếp cận h- ớng thủ tục thì một hệ thống phần mềm đ- ợc xem nh- là dãy các công việc cần thực hiện nh- đọc dữ liệu, tính toán, xử lý, lập báo cáo và in ấn kết quả v.v... Mỗi công việc đó sẽ đ- ợc thực hiện bởi một số hàm nhất định. Nh- vậy trọng tâm của cách tiếp cận này là các hàm chức năng. Cấu trúc của ch- ơng trình đ- ợc xây dựng theo cách tiếp cận h- ớng thủ tục có dạng nh- hình 3-1.



Hình 3-1. Cấu trúc của ch- ơng trình h- ớng thủ tục

Lập trình h- ớng 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. Ch- ơng trình đ- ợc xây dựng theo cách tiếp cận h- ớng thủ tục thực chất là danh sách các câu lệnh mà theo đó máy tính cần thực hiện. Danh sách các lệnh đó đ- ợc tổ chức thành từng nhóm theo đơn vị cấu trúc cú pháp của ngôn ngữ đặc tả hay ngôn ngữ lập trình và đ- ợc 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 ng- ời ta th- ờng 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 s- dụng để thực hiện công việc của mình. Cái gì sẽ xảy ra đối với dữ liệu và gắn dữ liệu với các hàm nh- thế nào? cùng nhiều vấn đề khác cần phải giải quyết khi chúng ta muốn xây dựng các ph- ơng pháp thích hợp để phát triển hệ thống trong thế giới thực. Trong ch- ơng trình có nhiều hàm, th- ờng thì có nhiều thành phần dữ liệu quan trọng sẽ đ- ợc 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). Mối quan hệ giữa dữ liệu và hàm trong ch- ơng trình h- ớng thủ tục đ- ợc 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 ch- ơng 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 nh- tất cả các hàm liên quan đến dữ liệu đó.

Ngoài những trở ngại mà chúng ta đã nêu ở trên thì mô hình đ- ợc xây dựng theo cách tiếp cận h- ớng thủ tục không mô tả đ- ợc đầy đủ, trung thực hệ thống trong thực tế. Bởi vì cách đặt trọng tâm vào hàm là h- ớng tới hoạt động sẽ không thực sự t- ơng ứng với các thực thể trong hệ thống của thế giới thực.

Tóm lại những đặc tính chính của lập trình h- ớng thủ tục là:

+ Tập trung vào công việc cần thực hiện (thuật toán).

+ Ch- ơng trình lớn đ- ợc chia thành các hàm nhỏ hơn.

+ Phần lớn các hàm sử dụng dữ liệu chung.

+ Dữ liệu trong hệ thống đ- ợc chuyển động từ hàm này sang hàm khác.

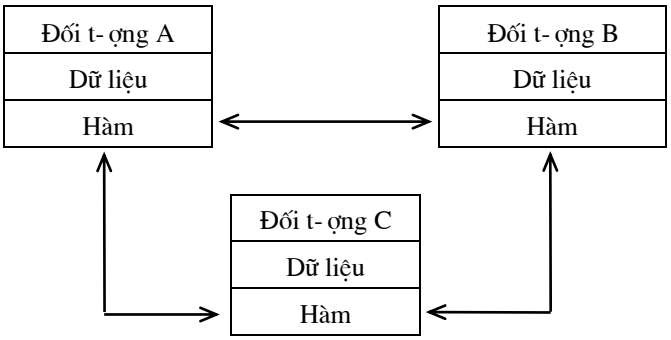
+ Hàm biến đổi dữ liệu từ dạng này sang dạng khác.

+ Sử dụng cách tiếp cận top-down trong thiết kế ch- ơng trình.

3.3. Lập trình h□ớng đối t□ợng

Nh- ở các phần tr- ớc chúng ta đã nêu, để giải quyết đ- ợc những vấn đề tồn tại trong công nghệ phần mềm thì chúng ta cần phải sử dụng những ph- ơng pháp, công cụ thích hợp để phát triển phần mềm. Trong các mục §1 và §2 chúng ta đã đề cập đến ph- ơng pháp phân tích, thiết kế h- ớng đối t- ợng. Trong mục này chúng ta tiếp tục nghiên cứu về ph- ơng pháp lập trình h- ớng đối t- ợng.

Lập trình h- ớng đối t- ợng đặt trọng tâm vào đối t- ợng, yếu tố quan trọng trong quá trình phát triển ch- ơng trình và nó không cho phép dữ liệu chuyển động tự do trong hệ thống. Dữ liệu đ- ợc gắn chặt với từng hàm thành các vùng riêng mà các hàm đó tác động lên và nó đ- ợc bảo vệ cấm các hàm bên ngoài không đ- ợc truy nhập một cách tuỳ tiện. LTHĐT cho phép chúng ta phân tích bài toán thành tập các thực thể đ- ợc gọi là các đối t- ợng và sau đó xây dựng các dữ liệu cùng với các hàm xung quanh các đối t- ợng đó. Tổ chức dữ liệu và hàm trong các ch- ơng trình h- ớng đối t- ợng đ- ợc mô tả nh- trong hình 3-3.



Hình 3-3. Tổ chức dữ liệu và hàm trong ch- ơng trình HĐT

Dữ liệu của một đối t- ợng chỉ có thể đ- ợc truy nhập bởi chính các hàm xác định trong đối t- ợng đó. Tuy nhiên các hàm của đối t- ợng này có thể truy nhập tới các hàm của đối t- ợng khác, nghĩa là các đối t- ợng trao đổi với nhau thông qua việc trao đổi thông báo. Lập trình h- ớng đối t- ợng có những đặc tính chủ yếu sau:

1. Tập chung vào dữ liệu thay cho các hàm.
2. Ch- ơng trình đ- ợc chia thành các đối t- ợng.
3. Các cấu trúc dữ liệu đ- ợc thiết kế sao cho đặc tả đ- ợc các đối t- ợng.
4. Các hàm xác định trên các vùng dữ liệu của đối t- ợng đ- ợc gắn với nhau trên cấu trúc dữ liệu đó.
5. Dữ liệu đ- ợc bao bọc, che giấu và không cho phép các hàm ngoại lai truy nhập tự do.
6. Các đối t- ợng trao đổi với nhau thông qua các hàm.
7. Dữ liệu và các hàm mới có thể dễ dàng bổ sung vào đối t- ợng nào đó khi cần thiết.
8. Ch- ơng trình đ- ợc thiết kế theo cách tiếp cận bottom-up (d- ới-lên).

Lập trình h- ớng đối t- ợng là khái niệm mới và đ- ợc hiểu rất khác nhau đối với nhiều ng- ời. Ở đây chúng ta có thể hiểu lập trình h- ớng đối t- ợng là cách tiếp cận để phân chia ch- ơng trình thành các đơn thể (module) bằng cách tạo ra các vùng bộ nhớ cho cả dữ liệu lẫn hàm và chúng sẽ đ- ợc sử dụng nh- các mẫu để tạo ra bản sao từng đơn thể khi cần thiết. Đối t- ợng ở đây đ- ợc xem nh- là vùng phân chia bộ nhớ trong máy tính để l- u trữu 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 t- ợng có thể sử dụng bởi nhiều ch- ơng trình khác nhau mà không ảnh h- ởng lẫn nhau.

Khái niệm "h- ớng đối t- ợng" nhiều ng- ời 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 t- ợng
2. Lớp

3. Trừu t- ợng hoá dữ liệu
4. Kế thừa
5. T- ơng ứng bội
6. Liên kết động
7. Truyền thông báo

Đối t□ợng

Trong các mục tr- ớc chúng ta đã nêu cách xác định đối t- ợng trong quá trình phân tích và thiết kế h- ớng đối t- ợng. Ở đây chúng ta tìm hiểu chi tiết hơn để hiểu rõ vai tò của đối t- ợng trong cách tiếp cận h- ớng đối t- ợng nói chung và LTHĐT nói riêng.

Đối t- ợng là thực thể đ- ợc xác định trong thời gian hệ thống h- ớng đối t- ợng hoạt động. Nh- vậy đối t- ợng có thể biểu diễn là con ng- ời, vật, hay một bảng dữ liệu hoặc bất kỳ một hạng thức nào đó cần xử lý trong ch- ơng trình. Đối t- ợng cũng có thể là các dữ liệu đ- ợc định nghĩa bởi ng- ời sử dụng (ng- ời lập trình) nh- vector, danh sách, các record v.v... Nhiệm vụ của LTHĐT là phân tích bài toán thành các đối t- ợng và xác định đ- ợc bản chất của sự trao đổi thông tin giữa chúng. Đối t- ợng trong ch- ơng trình cần phải đ- ợc chọn sao cho nó thể hiện đ- ợc một cách gần nhất so với những thực thể trong thế giới thực.

Khi ch- ơng trình thực hiện, các đối t- ợng sẽ trao đổi với nhau bằng cách gửi hay nhận thông báo. Ví dụ BAN_DOC và CHO_MUON là hai đối t- ợng trong hệ thống th- viện, đối t- ợng BAN_DOC có thể gửi một thông báo (bằng phiếu yêu cầu chẳng hạn) cho đối t- ợng CHO_MUON yêu cầu m- ợn cuốn "LẬP TRÌNH H□ỚNG ĐỐI T□ỢNG VỚI C++". Mỗi đối t- ợng có dữ liệu và các hàm để xử lý dữ liệu đó. Các đối t- ợng trao đổi với nhau mà không cần biết chi tiết về dữ liệu và các thuật toán xử lý của đối t- ợng khác. Để trao đổi đ- ợc với nhau, mỗi đối t- ợng chỉ cần biết kiểu thông báo mà nó nhận và kiểu thông báo mà nó sẽ gửi cho các đối t- ợng khác.

Các lớp đối t□ợng

Nh- trên chúng ta đã xác định, đối t- ợng trong ch- ơng trình gồm cả dữ liệu và các hàm xử lý trên dữ liệu đó. Một tập dữ liệu và các hàm của một đối t- ợng có thể đ- ợc xem nh- một kiểu dữ liệu đ- ợc định nghĩa bởi ng- ời sử dụng. Kiểu dữ liệu ở đây đ- ợc gọi là lớp (class). Trong lập trình, các đối t- ợng là các biến có kiểu class. Khi một lớp đ- ợc định nghĩa, thì nó có thể tạo ra số l- ợng các đối t- ợng tuỳ ý của lớp đó. Nh- vậy, TOA, LE, BUOI, CAM là các loại quả trong lớp HOA_QUA. Lớp là kiểu đ- ợc ng- ời sử dụng định nghĩa và nó cũng có các tính chất nh- các kiểu chuẩn integer, float trong các ngôn ngữ lập trình. T- ơng tự nh- kiểu dữ liệu đã đ- ợc định nghĩa trong ch- ơng trình, lệnh khai báo

HOA_QUA TAO;

sẽ tạo ra đối t- ợng TAO trong lớp HOA_QUA.

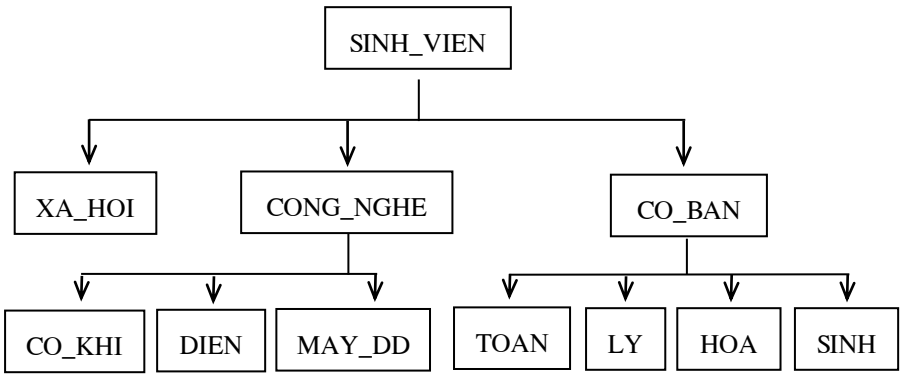
Trừu t□ợng hoá dữ liệu và bao gói thông tin

Việc đóng gói dữ liệu và các hàm vào một đơn vị cấu trúc (đ- ợc gọi là lớp) đ- ợc xem nh- một nguyên tắc bao gói (che giấu) thông tin. Dữ liệu đ- ợc tổ chức sao cho thế giới bên ngoài (các đối t- ợng ở lớp khác) không truy nhập đ- ợc vào mà chỉ cho phép các hàm trong cùng lớp hoặc trong những lớp có quan hệ kế thừa với nhau đ- ợc quyền truy nhập. Chính các hàm thành phần của lớp sẽ đóng vai trò nh- là giao diện giữa dữ liệu của đối t- ợng và phần còn lại của ch- ơng trình. Nguyên tắc bao gói dữ liệu để ngăn cấm sự truy nhập trực tiếp trong lập trình đ- ợc gọi là sự che giấu thông tin.

Trừu t- ợng 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 t- ợng hoá. Ví dụ chúng ta có thể định nghĩa một lớp là danh sách các thuộc tính trừu t- ợng nh- là kích th- ớc, hình dáng, mầu và các hàm xác định trên các thuộc tính này để mô tả các đối t- ợng trong không gian hình học. Trong lập trình, lớp sử dụng nh- kiểu dữ liệu trừu t- ợng.

Kế thừa

Kế thừa là quá trình mà các đối t- ợng của lớp này đ- ợc quyền sử dụng một số tính chất của các đối t- ợng của lớp khác. Nguyên lý kế thừa hỗ trợ cho việc tạo ra cấu trúc phân cấp các lớp. Ví dụ, một tr- ờng 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 kề nhau nh- 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 tr- ờng ví dụ nh- : Họ và tên, quê, tuổi. Những đặc tính đó đ- ợc kế thừa ở trong các lớp XA_HOI, CONG_NGHE, CO_BAN. Các lớp dẫn xuất đó đ- ợc bổ sung thêm những thuộc tính, các hàm t- ơng ứng mô tả cho sinh viên. Các lớp XA_HOI, CONG_NGHE, CO_BAN đ- ợc bổ sung thêm những thuộc tính mới để phân biệt giữa các khối với nhau. Trong khối CO_BAN lại chia thành nhiều khoa nh- : 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 t- ởng sử dụng lại. Nghĩa là từ một lớp đã đ- ợc 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.

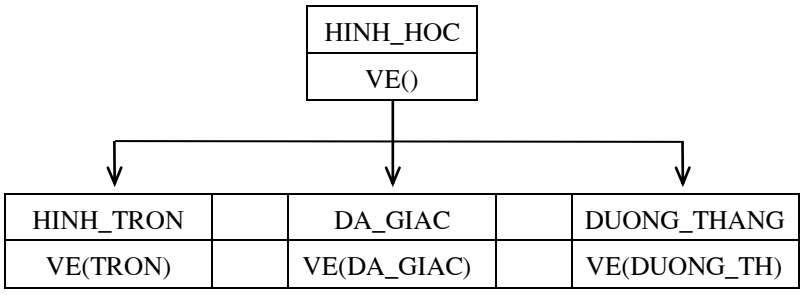
Khái niệm kế thừa đ- ợc hiểu nh- cơ chế sao chép ảo không đơn điệu. Trong thực tế, mọi việc xảy ra tựa nh- những lớp cơ sở đều đ- ợc sao vào trong lớp con (lớp dẫn xuất) mặc dù điều này không đ- ợc cài đặt t- ờng minh (nên gọi là sao chép ảo) và việc sao chép chỉ thực hiện đối với những thông tin ch- a đ- ợc xác định trong các lớp cơ sở (sao chép không đơn điệu). Do vậy, có thể diễn đạt cơ chế kế thừa nh- sau:

1. Lớp A kế thừa lớp B sẽ có (không t- ờng minh) tất cả các thuộc tính, hàm đã đ- ợc xác định trong B.

2. Bổ sung thêm một số thuộc tính, hàm để mô tả đ- ợc đúng các hành vi của những đối t- ợng mà lớp A quản lý.

T ơng ứng bội

Một khái niệm quan trọng nữa trong LTHĐT là khái niệm t- ơng ứng bội. T- ơng ứng bội là khả năng của một khái niệm (nh- các phép toán) có thể đ- ợc 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 t- ơng 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 (đ- ợc phân biệt bởi số l- ợng, kiểu của tham số) khi gọi để thực hiện.



Hình 3-5. T- ơng ứng bội của hàm VE()

Hàm VE() là hàm t- ơng ứng bội và nó đ- ợc xác định tuỳ theo ngữ cảnh khi sử dụng.

T- ơng ứng bội đóng vai trò quan trọng trong việc tạo ra các đối t- ợng có cấu trúc bên trong khác nhau nh- ng có khả năng cùng dùng chung một giao diện bên ngoài (nh- tên gọi). Điều này có nghĩa là một lớp tổng quát các phép toán đ- ợc định gnhĩa theo cùng một cách giống nhau. T- ơng 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 ch- ơng trình thực hiện các lời gọi tới các hàm, thủ tục đó. Nh- vậy trong liên kết động, nội dung của đoạn ch- ơng trình ứng với thủ tục, hàm sẽ không đ- ợc biết cho đến khi thực hiện lời gọi tới thủ tục, hàm đó. Liên kết động liên quan chặt chẽ tới t- ơng ứng bội và kế thừa. Chúng ta hãy l- u ý hàm VE() trong Hình 4-5. Theo nguyên lý kế thừa thì mọi đối t- ợng 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 t- ợng HINH_TRON, DA_GIAC, DUONG_THANG và vì vậy hàm VE() sẽ đ- ợc định nghĩa lại khi các đối t- ợng t- ơng ứng đ- ợc xác định. Khi thực hiện, ví dụ nh- khi vẽ một hình tròn, đoạn ch- ơng trình ứng với hàm VE() hình tròn đ- ợc gọi ra để thực hiện.

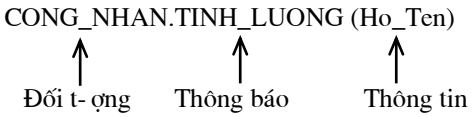
Truyền thông báo

Ch- ơng trình h- ớng đối t- ợng (đ- ợc thiết kế và lập trình theo h- ớng đối t- ợng) bao gồm một tập các đối t- ợng và mối quan hệ giữa các đối t- ợng với nhau. Vì vậy, lập trình trong ngôn ngữ h- ớng đối t- ợng bao gồm các b- ớc sau:

1. Tạo ra các lớp xác định các đối t- ợng và hành vi của chúng.
2. Tạo ra các đối t- ợng theo định nghĩa của các lớp.
3. Xác định sự trao đổi giữa các đối t- ợng.

Các đối t- ợng gửi và nhận thông tin với nhau giống nh- con ng- ời trao đổi với nhau. Chính nguyên lý trao đổi thông tin bằng cách truyền thông báo cho phép chúng ta dễ dàng xây dựng đ- ợc hệ thống mô phỏng gần hơn những hệ thống trong thế giới thực. Truyền thông báo cho một đối t- ợng tức là báo cho nó phải thực hiện một việc gì đó. Cách ứng xử của đối t- ợng sẽ đ- ợc mô tả ở trong lớp thông qua các hàm (hay còn đ- ợc gọi là lớp dịch vụ).

Trong ch- ơng trình, thông báo gửi đến cho một đối t- ợng chính là yêu cầu thực hiện một công việc cụ thể, nghĩa là sử dụng những hàm t- ơng ứng để xử lý dữ liệu đã đ- ợc khai báo trong đối t- ợng đó. Vì vậy, trong thông báo phải chỉ ra đ- ợc hàm cần thực hiện trong đối t- ợng nhận thông báo. Hơn thế nữa,thông báo truyền đi phải xác định tên đối t- ợng, tên hàm (thông báo) và thông tin truyền đi. Ví dụ, lớp CONG_NHAN có thể hiện là đối t- ợng cụ thể đ- ợc đại diện bởi Ho_Ten nhận đ- ợc thông báo cần tính l- ơng thông qua hàm TINH_LUONG đã đ- ợc xác định trong lớp CONG_NHAN. Thông báo đó sẽ đ- ợc xử lý nh- sau:



Mỗi đối t- ợng chỉ tồn tại trong thời gian nhất định. Đối t- ợng đ- ợc tạo ra khi nó đ- ợc khai báo và sẽ bị huỷ bỏ khi ch- ơng trình ra khỏi miền xác định của đối t- ợng đó. Sự trao đổi thông tin chỉ có thể thực hiện trong thời gian đối t- ợng tồn tại.

Các u điểm của lập trình h ớng đối t ợng

Nh- trên chúng ta đã phân tích, lập trình h- ớng đối t- ợng đem lại một số lợi thế cho cả ng- ời thiết kế lẫn ng- ời lập trình. Cách tiếp cận h- ớng đối t- ợng giải quyết đ- ợc nhiều vấn đề tồn tại trong quá trình phát triển phần mềm và tạo ra đ- ợc những sản phẩm phần mềm có chất l- ợng cao. Những ph- ơng pháp này mở ra một triển vọng to lớn cho những ng- ời lập trình. Hy vọng sẽ có nhiều sản phẩm phần mềm tốt hơn, đáp ứng đ- ợc những tính chất về sản phẩm chất l- ợng cao trong công nghệ phần mềm và nhất là bảo trì hệ thống ít tốn kém hơn. Những - u điểm chính của LTHĐT là:

1. Thông qua nguyên lý kế thừa, chúng ta có thể loại bỏ đ- ợc những đoạn ch- ơng trình lặp lại, d- thừa trong quá trình mô tả các lớpp và mở rộng khả năng sử dụng các lớp đã đ- ợc xây dựng.

2. Ch- ơng trình đ- ợc xây dựng từ những đơn thể (đối t- ợng) trao đổi với nhau nên việc thiết kế và lập trình sẽ đ- ợc thực hiện theo quy trình nhất định chứ không phải dựa vào kinh nghiệm và kỹ thuật nh- tr- ớc. Điều này đảm bảo rút ngắn đ- ợc thời gian xây dựng hệ thống và tăng năng suất lao động.

3. Nguyên lý giấu thông tin giúp ng- ời lập trình tạo ra đ- ợc những ch- ơng trình an toàn không bị thay bởi những đoạn ch- ơng trình khác.

4. Có thể xây dựng đ- ợc ánh xạ các đối t- ợng của bài toán vào đối t- ợng của ch- ơng trình.

5. Cách tiếp cận thiết kế đặt trọng tâm vào đối t- ợng, giúp chúng ta xây dựng đ- ợc mô hình chi tiết và gần với dạng cài đặt hơn.

6. Những hệ thống h- ớng đối t- ợng 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 t- ợng giúp cho việc mô tả giao diện

với các hệ thống bên ngoài trở nên đơn giản hơn.

8. Có thể quản lý đ- ợc độ phức tạp của những sản phẩm phần mềm.

Không phải trong hệ thống h- ớng đối t- ợng 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 ph- ơng pháp thực hiện của ng- ời phát triển phần mềm.

3.4. Các ngôn ngữ h□ớng đối t□ợng

Lập trình h- ớng đối t- ợng không là đặc quyền của một ngôn ngữ nào đặc biệt. Cũng giống nh- lập trình có cấu trúc, những khái niệm trong lập trình h- ớng đối t- ợng có thể cài đặt trong những ngôn ngữ lập trình nh- C hoặc Pascal. Tuy nhiên, đối với những ch- ơng 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 h- ớng đối t- ợng thì phải thực hiện nhiều thoả hiệp. Những ngôn ngữ đ- ợc thiết kế đặc biệt, hỗ trợ cho việc mô tả, cài đặt các khái niệm của ph- ơng pháp h- ớng đối t- ợng đ- ợc gọi chung là ngôn ngữ h- ớng đối t- ợng.

Dựa vào khả năng đáp ứng các khái niệm về h- ớng đối t- ợng, chúng ta có thể chia ra làm hai loại:

1. Ngôn ngữ lập trình dựa trên đối t- ợng (object - based)

2. Ngôn ngữ lập trình h- ớng đối t- ợng (object - oriented)

Lập trình dựa trên đối t- ợng là kiểu lập trình hỗ trợ chính cho việc bao gói, che giấu thông tin và định danh các đối t- ợng. Lập trình dựa trên đối t- ợng 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 t- ợng

+ Phép toán tải bội

Ngôn ngữ hỗ trợ cho kiểu lập trình trên đ- ợc gọi là ngôn ngữ lập trình dựa trên đối t- ợng. 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 t- ợng.

Lập trình h- ớng đối t- ợng là kiểu lập trình dựa trên đối t- ợng 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 nh- sau:

Các đặc tính dựa trên đối t- ợng + kế thừa + liên kết động.

Ngôn ngữ hỗ trợ cho những đặc tính trên đ- ợc gọi là ngôn ngữ LTHĐT, ví dụ nh- 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 ch- ơng trình đã có và vào tổ chức của nhóm tham gia xây dựng phần mềm. T- ơng tự nh- ngôn ngữ lập trình C, C++ đang đ- ợc 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 đ- ợc nhắc đến nhiều nhất hiện nay trong công nghệ phần mềm và nó đ- ợc 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 ng- ời sử dụng, ví dụ nh- Windows. Hàng trăm hệ thống với giao diện Windows đã đ- ợc phát triển dựa trên kỹ thuật LTHĐT. Những hệ thông tin doanh nghiệp trong thực tế rất phức tạp, chứa nhiều đối t- ợng với các thuộc tính và hàm phức tạp. Để giải quyết những hệ thống phức hợp thế thì LTHĐT lạ 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 h- ớng đối t- ợng.

+ 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 ph- ơng pháp phát triển h- ớng đối t- ợng nói chung, chúng ta hy vọng nền công nghiệp phần mềm sẽ cải tiến không những về chất l- ợng mà còn gia tăng nhanh về số l- ợng trong t- ơng lai. Kỹ nghệ h- ớng đối t- ợng 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.