

Lập trình hướng đối tượng với C++

Chương I. Giới thiệu về lập trình hướng đối tượng.

1.1. Một số phương pháp lập trình:

1.1.1. Phương pháp lập trình tuyến tính:

- Chương trình tương đối ngắn, thường ít hơn 100 dòng...
- Không có khả năng kiểm soát phạm vi nhìn thấy của các dữ liệu. Mọi dữ liệu trong chương trình đều là dữ liệu toàn cục nghĩa là chúng có thể bị sửa đổi ở bất kỳ phần nào của chương trình.

1.1. Một số phương pháp lập trình:

1.1.2. Phương pháp lập trình cấu trúc:

- Ra đời vào cuối năm 1960 và 1970.
- Chương trình được tổ chức theo các công việc mà chúng thực hiện.
- Chương trình chia nhỏ thành các chương trình con riêng rẽ (còn gọi là hàm hay thủ tục) thực hiện các công việc rời rạc trong quá trình lớn hơn, phức tạp hơn. Mỗi hàm có dữ liệu và logic riêng.

1.1. Một số phương pháp lập trình:

- Lập trình viên được phân công viết một tập hợp các hàm và các kiểu dữ liệu. Vì có nhiều lập trình viên khác nhau quản lý các hàm riêng, có liên quan đến các kiểu dữ liệu dùng chung nên các thay đổi mà lập trình viên tạo ra trên một phần tử dữ liệu sẽ làm ảnh hưởng đến công việc của tất cả các người còn lại trong nhóm.

1.1. Một số phương pháp lập trình:

1.1.3. Lập trình hướng đối tượng:

- Được xây dựng trên nền tảng của khái niệm lập trình có cấu trúc và sự trừu tượng hóa dữ liệu.
- Một chương trình hướng đối tượng được thiết kế xoay quanh dữ liệu mà chúng ta có thể làm việc trên đó, hơn là theo bản thân chức năng của chương trình.
- Là phương pháp lập trình lấy đối tượng làm nền tảng để xây dựng thuật giải, xây dựng chương trình.
- Lập trình hướng đối tượng liên kết cấu trúc dữ liệu với các thao tác, theo cách mà tất cả thường nghĩ về thế giới quanh mình.

1.2. Đặc điểm của lập trình hướng đối tượng:

- ❑ Tập chung vào dữ liệu thay cho các hàm.
- ❑ Chương trình được chia thành các đối tượng.
- ❑ Các cấu trúc dữ liệu được thiết kế sao cho đặc tả được các đối tượng.
- ❑ Các hàm xác định trên các vùng dữ liệu của đối tượng được gắn với nhau trên cấu trúc dữ liệu đó.
- ❑ Dữ liệu được bao bọc, che giấu và không cho phép các hàm ngoại lai truy nhập tự do.
- ❑ Các đối tượng trao đổi với nhau thông qua các hàm.
- ❑ Dữ liệu và các hàm mới có thể dễ dàng bổ sung vào đối tượng nào đó khi cần thiết.
- ❑ Chương trình được thiết kế theo cách tiếp cận bottom-up (dưới-lên).

Một số khái niệm liên quan đến lập trình hướng đối tượng.

1. Đối tượng
2. Lớp
3. Trừu tượng hoá dữ liệu
4. Kế thừa
5. Tương ứng bội
6. Liên kết động
7. Truyền thông báo

Đối tượng(object)

- Đối tượng là một thực thể cụ thể



Lớp(Class)

- Một **lớp** là một thiết kế (blueprint) hay mẫu (prototype) cho các đối tượng cùng kiểu
 - Ví dụ: lớp XeDap là một thiết kế chung cho nhiều đối tượng xe đạp được tạo ra
 - TAO, LE, BUOI, CAM là các loại quả trong lớp HOA_QUA

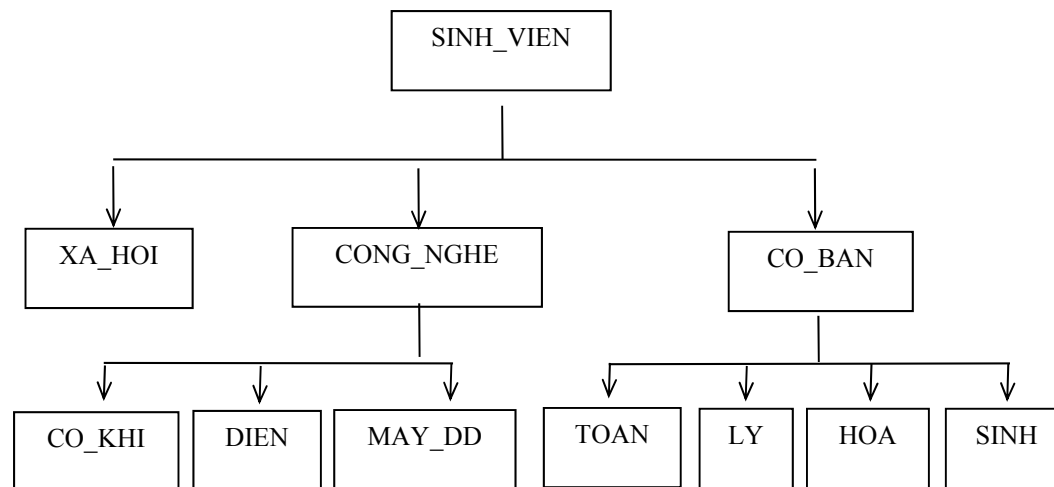
Trừu tượng hóa dữ liệu (Abstraction)

- Trừu tượng hoá là cách biểu diễn những đặc tính và bỏ qua những chi tiết vụn vặt hoặc những giải thích. Để xây dựng các lớp, chúng ta phải sử dụng khái niệm trừu tượng hoá.
- Ví dụ: Khi xét đến sinh viên ta chỉ quan tâm đến các thông tin: mã sinh viên, họ và tên, ngày sinh, giới tính, địa chỉ, số cmt.

Kế thừa (Inheritance).

- Kế thừa là quá trình mà các đối tượng của lớp này được quyền sử dụng một số tính chất của các đối tượng của lớp khác. Nguyên lý kế thừa hỗ trợ cho việc tạo ra cấu trúc phân cấp các lớp.

□ Ví dụ:

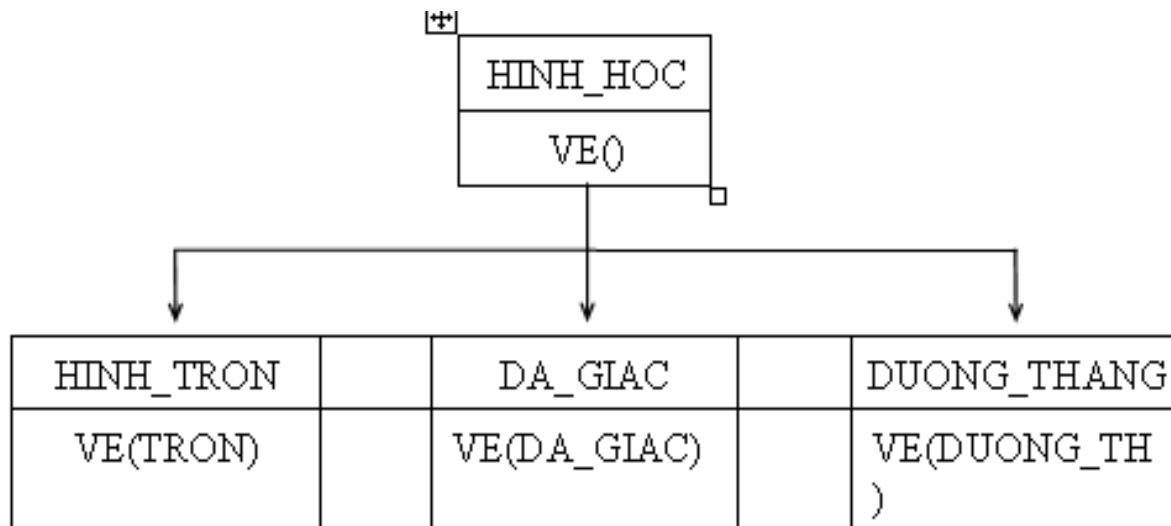


Tương ứng bội

- Tương ứng bội là khả năng của một khái niệm (như các phép toán) có thể được xuất hiện ở nhiều dạng khác nhau. Ví dụ, phép $+$ có thể biểu diễn cho phép "cộng" các số nguyên (int), số thực (float), số phức (complex) hoặc chuỗi ký tự (string) v.v...
- Hành vi của phép toán tương ứng bội phụ thuộc vào kiểu dữ liệu mà nó sử dụng để xử lý.

Tương ứng bội

- Ví dụ: Hình 3-5 cho chúng ta thấy hàm có tên là VE có thể sử dụng để vẽ các hình khác nhau phụ thuộc vào tham số (được phân biệt bởi số lượng, kiểu của tham số) khi gọi để thực hiện.



Liên kết động

- Liên kết động là dạng liên kết các hàm, thủ tục khi chương trình thực hiện các lời gọi tới các hàm, thủ tục đó. Như vậy trong liên kết động, nội dung của đoạn chương trình ứng với thủ tục, hàm sẽ không được biết cho đến khi thực hiện lời gọi tới thủ tục, hàm đó. Liên kết động liên quan chặt chẽ tới tương ứng bội và kế thừa.

Truyền thông báo

- Các đối tượng gửi và nhận thông tin với nhau giống như con người trao đổi với nhau. Chính nguyên lý trao đổi thông tin bằng cách truyền thông báo cho phép chúng ta dễ dàng xây dựng được hệ thống mô phỏng gần hơn những hệ thống trong thế giới thực.
- Truyền thông báo cho một đối tượng tức là báo cho nó phải thực hiện một việc gì đó. Cách ứng xử của đối tượng sẽ được mô tả ở trong lớp thông qua các hàm (hay còn được gọi là lớp dịch vụ).

Truyền thông báo

- Ví dụ: lớp CONG_NHAN có thể hiện là đối tượng cụ thể được đại diện bởi Ho_Ten nhận được thông báo cần tính lương thông qua hàm TINH_LUONG đã được xác định trong lớp CONG_NHAN. Thông báo đó sẽ được xử lý như sau:

CONG_NHAN.TINH_LUONG (Ho_Ten)

↑ ↑ ↑

Đối tượng Thông báo Thông tin

1.3. Những ngôn ngữ hỗ trợ lập trình hướng đối tượng.

- Lập trình hướng đối tượng không là đặc quyền của một ngôn ngữ nào đặc biệt.
- Những ngôn ngữ được thiết kế đặc biệt, hỗ trợ cho việc mô tả, cài đặt các khái niệm của phương pháp hướng đối tượng được gọi chung là ngôn ngữ hướng đối tượng.
- Ví dụ: C++, Java, C#, PHP...
- Trong môn học này ta sử dụng ngôn ngữ C++ để lập trình hướng đối tượng.

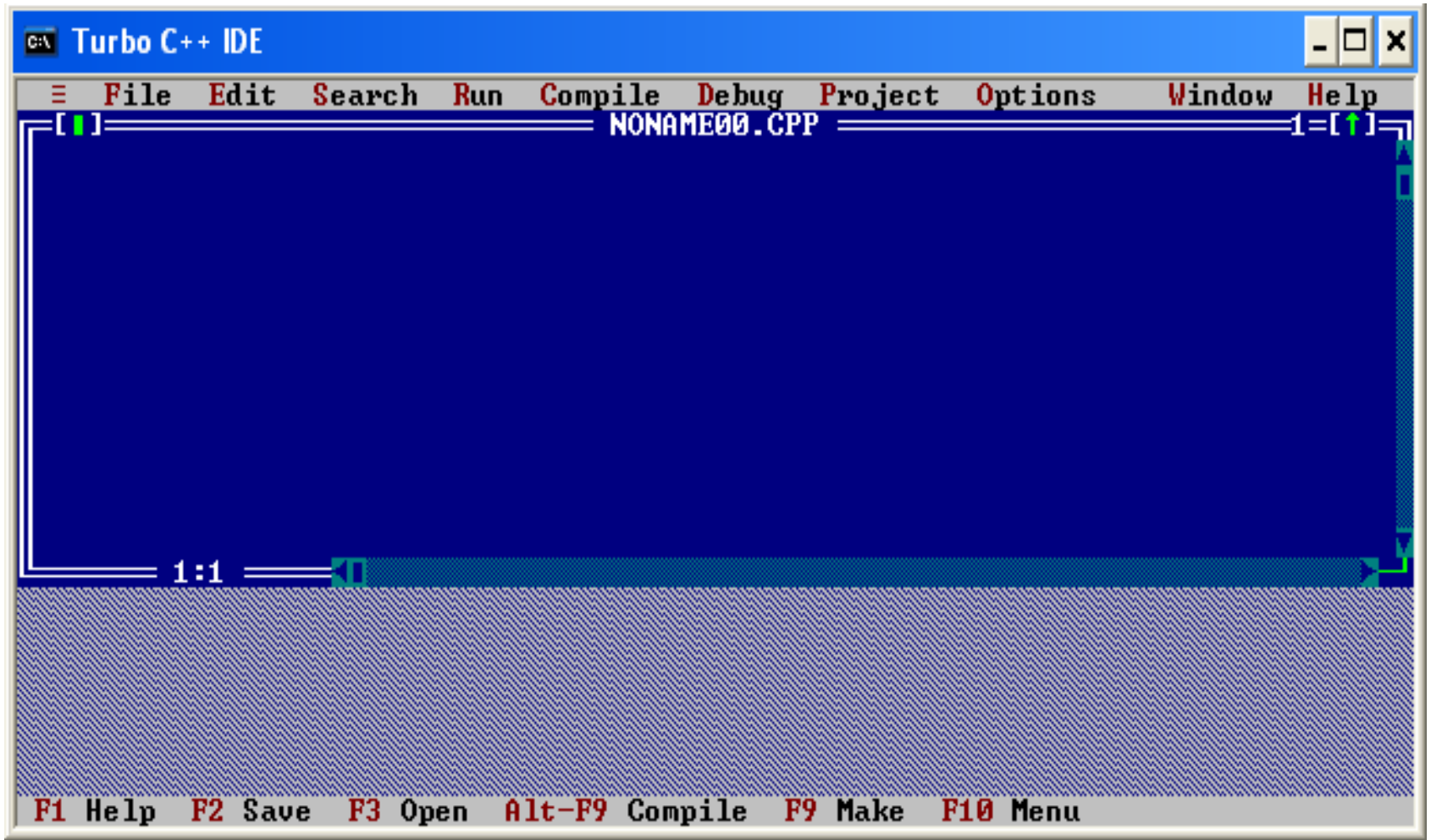
1.3.1. Cài đặt chương trình C++

- Down load Turbo C++3.0 và tiến hành Install.
- Khi cài đặt chú ý: vị trí đặt thư mục TC.

1.3.2. Khởi động chương trình C++

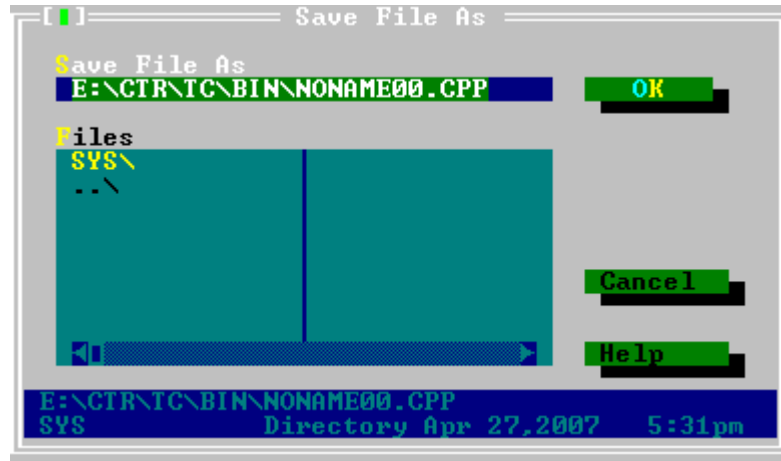
- Sau khi cài đặt chương trình Turbo C++3.0 trong thư mục TC(thông thường được đặt trong ổ C:\) có các thư mục:
 - BIN: chứa file chạy chương trình.
 - INCLUDE: chứa các thư viện của chương trình.
 - LIB: chứa các thư viện vào ra.
 - BGI: chứa thư viện đồ họa.
- Vào thư mục BIN: kích đúp chuột tại file TC.exe để khởi động chương trình C++.

1.3.3. Giao diện của chương trình C++



1.3.4. Tạo tệp mới, lưu tệp, đóng.

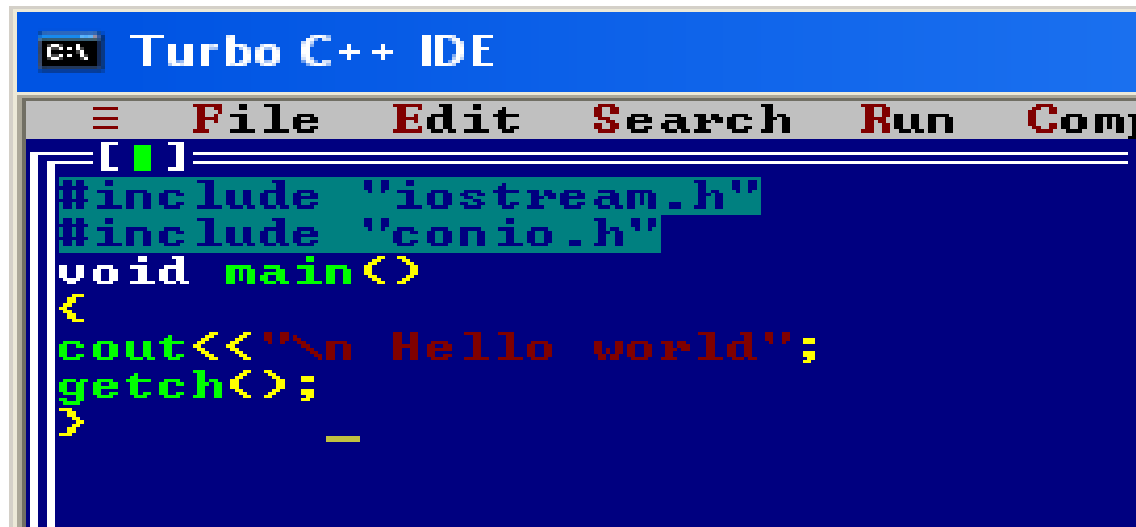
- Tạo tệp mới: File/New
- Lưu tệp:
 - File/Save



- Đóng chương trình: File/Quit hoặc Alt+X

1.3.5. Cấu trúc chương trình C++

- Tương ứng cấu trúc cơ bản của chương trình C, ngôn ngữ về cơ bản cũng giống C.
 - Ví dụ:



The screenshot shows the Turbo C++ IDE window. The title bar reads 'C:\ Turbo C++ IDE'. The menu bar includes 'File', 'Edit', 'Search', 'Run', and 'Comp'. The code editor displays the following C++ code:

```
[ ]  
#include "iostream.h"  
#include "conio.h"  
void main()  
{  
    cout<<"\n Hello world";  
    getch();  
}
```

1.3.6. Câu lệnh vào ra với ngôn ngữ C++

- Câu lệnh đưa dữ liệu ra màn hình:
 - Sử dụng thư viện `iostream.h`
 - Cú pháp:
 - `cout<<"dữ liệu 1"<<"dữ liệu 2"<<...<<"dữ liệu n";`
 - Nếu “dữ liệu i” là một biến số thì không đặt trong dấu “”, nếu là nội dung chữ cần hiển thị thì phải đặt trong cặp dấu “”.
 - Ví dụ:
 - `cout<<"Hello world";`
 - `int a=10; cout<<a;`

1.3.6. Câu lệnh vào ra với ngôn ngữ C++

■ Câu lệnh nhập dữ liệu từ bàn phím:

- Sử dụng thư viện `iostream.h`

- Cú pháp:

 - `cin>>biến 1>>biến 2>>...>>biến n;`

- Nếu “biến i” là một biến có kiểu dữ liệu cơ bản hay dữ liệu do người dùng định nghĩa. Biến không được đặt trong dấu “”.

- Ví dụ:

 - `int a; cout<<“Hay nhap gia tri a=”; cin>>a;`

Bài tập thực hành

- Cài đặt chương trình turbo C++3.0
 - Khởi động chương trình.
 - Tạo tệp VD1.CPP
 - Viết chương trình nhập một số từ bàn phím và in ra màn hình kết quả vừa nhập.
 - Tạo tệp VD2.CPP
 - Viết chương trình cộng, trừ, nhân, chia hai số nguyên và in kết quả ra màn hình.
-

Bài tập

- Ôn tập về chương trình con.

Chương II. Lớp và Đối tượng.

2.1. Thiết kế và sử dụng lớp

2.1.1. Lớp

- Lớp là khái niệm trung tâm của lập trình hướng đối tượng, nó là sự mở rộng của các khái niệm cấu trúc (struct) của C và bản ghi (record) của PASCAL.
- Giống như cấu trúc, lớp có thể xem như một kiểu dữ liệu. Vì vậy lớp còn gọi là kiểu đối tượng và lớp được dùng để khai báo các biến, mảng đối tượng (như thể dùng kiểu int để khai báo các biến mảng nguyên)

2.1. Thiết kế và sử dụng lớp

2.1.2. Định nghĩa lớp.

- Cú pháp:

```
class tên_lớp
```

```
{
```

```
    //Khai báo các thành phần dữ liệu(thuộc tính)
```

```
    //Khai báo các phương thức và xây dựng.
```

```
};//kết thúc khai báo lớp
```

```
//Xây dựng các phương thức của lớp
```

- Ví dụ: Định nghĩa lớp Điểm

2.1. Thiết kế và sử dụng lớp

```
class diem
{
    private:
        int x,y;//toa do x va toa do y
    public:
        void nhapd();//phuong thuc nhap
        void ind();//phuong thuc in
};//kết thúc khai báo lớp
void diem::nhapd()
{...};
void diem::ind()
{...};
```

2.2. Các khái niệm và thành phần cơ bản của lớp.

■ Khái niệm:

- Thuộc tính: là một hạng mục dữ liệu liên kết với một lớp cụ thể mà không liên kết với các thể hiện của lớp. Nó được định nghĩa bên trong định nghĩa lớp và được chia sẻ bởi tất cả các thể hiện của lớp.
 - Ví dụ: với lớp Điểm thì hoành độ, tung độ là thuộc tính của Điểm.
-

2.2. Các khái niệm và thành phần cơ bản của lớp.

- Phương thức: là một phương thức được triệu gọi mà không tham khảo tới bất kỳ một đối tượng nào. Tất cả các phương thức lớp ảnh hưởng đến toàn bộ lớp chứ không ảnh hưởng đến một đối tượng riêng rẽ nào.
- Ví dụ với lớp Điểm thì phương thức sẽ là:
 - Nhập điểm
 - In điểm
 - Dịch chuyển.
 - Tính khoảng cách từ nó đến 1 điểm khác.
 - ...

2.2. Các khái niệm và thành phần cơ bản của lớp.

- Thành phần cơ bản của lớp:
 - `private`
 - `public`
 - `protected`
- Các thành phần dữ liệu, phương thức được khai báo sau từ khóa nào thì thuộc thành phần ấy. Nếu không chỉ định `private` hay `public`, `protected` thì mặc định là `public`.
- Các thuộc tính thường được khai báo sau từ khóa `private`

2.2. Các khái niệm và thành phần cơ bản của lớp.

- Các phương thức thường được khai báo sau từ khóa public.
- Ý nghĩa của các thành phần:
 - private: những thành phần dữ liệu, phương thức nào được khai báo sau từ khóa này sẽ được giấu kín, bảo vệ an toàn dữ liệu của lớp, không cho phép các hàm bên ngoài xâm nhập vào dữ liệu của lớp.
 - public: những thành phần dữ liệu, phương thức nào được khai báo sau từ khóa có thể được gọi tới bởi hàm khác trong chương trình.

2.2. Các khái niệm và thành phần cơ bản của lớp.

- Ý nghĩa của các thành phần:
 - protected: những thành phần dữ liệu, phương thức nào được khai báo sau từ khóa này sẽ được giấu kín, bảo vệ an toàn dữ liệu của lớp, không cho phép các hàm bên ngoài xâm nhập vào dữ liệu của lớp. Nhưng được phép sử dụng trong kế thừa.

Một số ví dụ về lớp

- Lớp số phức, lớp đa thức, lớp sinh viên, lớp nhân viên, lớp matrix...

2.3. Đối tượng

- Một lớp có thể xem như một kiểu đối tượng và có thể dùng để khai báo các biến, mảng đối tượng.
- Biến của lớp được gọi là thể hiện của lớp hay đối tượng.

2.3.1. Khai báo đối tượng:

- Tên_lớp tên_biến; //tên_biến hay còn gọi là tên đối tượng. Vì vậy ta viết
- Tên_lớp tên_đối_tượng;
- Ví dụ khai báo đối tượng của lớp điểm
 - Diem D1,D2;

2.3.1. Khai báo đối tượng:

- ❑ Khai báo mảng đối tượng

Tên_lớp tên_mảng[số phần tử];

VD: Diem D[10]; //D là mảng đối tượng điểm gồm 10 điểm

- ❑ Khai báo con trỏ đối tượng

Tên_lớp *tên_con_trỏ;

VD: Diem *p; //p là con trỏ đối tượng điểm.

2.3.2. Truy cập các thành phần của lớp

- Truy cập đến thành phần lớp thông qua đối tượng.

- Cú pháp:

- Tên_đối_tượng.Tên_thành_viên[()]=giá trị;

- Ví dụ:

- Diem d1;

- d1.nhapd();//gọi đến phương thức nhapd() của lớp.

Chương II. Lớp và Đối tượng(tiếp)

2.4. Phương thức

2.4.1. Cách khai báo và xây dựng phương thức

a. Khai báo phương thức

- Phương thức được khai báo bên trong lớp cần xây dựng.
- Phương thức được khai báo theo cú pháp sau:

```
void|KDL  Tên_PT([Tham số]);
```

2.4.1. Cách khai báo và xây dựng phương thức

a. Khai báo phương thức

- Nếu phương thức không trả về giá trị thì khai báo bắt đầu với từ khóa **void**
- Nếu phương thức trả về giá trị thì khai báo bắt đầu với **Kiểu_Dữ_Liệu** tương ứng kiểu dữ liệu giá trị cần trả về

2.4.1. Cách khai báo và xây dựng phương thức

Ví dụ:

```
class diem
{
    private:
        int x,y;//toa do x va toa do y
    public:
        //Khai báo hai phương thức không trả về giá trị
        void nhapd();//phuong thuc nhap
        void ind();//phuong thuc in
        //Khai báo phương thức trả về giá trị
        float KC2D(diem D2);
};//kết thúc khai báo lớp
```

2.4.1. Cách khai báo và xây dựng phương thức

b. Xây dựng phương thức

- Xây dựng bên trong khai báo lớp ngay khi khai báo phương thức đó.

```
class diem
{
    private:
        int x,y;//toa do x va toa do y
    public:
        //Khai báo hai phương thức không trả về giá trị
        void nhapd() //phuong thuc nhap
        {
            cout<<"\n Nhap toa do x:";cin>>x;
            cout<<"\n Nhap toa do y:";cin>>y;
        }
        ...
};
```

2.4.1. Cách khai báo và xây dựng phương thức

b. Xây dựng phương thức

- Xây dựng bên ngoài lớp sau phần khai báo lớp theo cú pháp sau:

```
void|KDL Tên_lớp::Tên_pt( [tham số])  
{  
    //nội dung phương thức  
}
```

Ví dụ

2.4.1. Cách khai báo và xây dựng phương thức

```
class diem
{
    private:
        int x,y;//toa do x va toa do y
    public:
        //Khai báo hai phương thức không trả về giá trị
        void nhapd(); //phuong thuc nhap
        ...
};
Void diem::nhapd()
{
    cout<<"\n Nhap toa do x:";cin>>x;
    cout<<"\n Nhap toa do y:";cin>>y;
}
```

2.4.1. Cách khai báo và xây dựng phương thức

Lưu ý: Các phương thức(hàm) được xây dựng bên trong lớp được gọi là phương thức(hàm) trực tuyến.

- + Các phương thức trực tuyến không chứa các câu lệnh phức tạp như for, while, switch... tính toán phức tạp
- + Các phương thức trực tuyến không cần được biên dịch khi chạy chương trình.

2.4.2. Lời gọi phương thức

- Các phương thức được gọi theo các cách sau
 - C1: Tên_đối_tượng.Tên_pt(đối số)
 - C2: Tên_đối_tượng.Tên_lớp::tên_pt(đối số)
- Ví dụ với phương thức nhapd() ở trên ta có thể gọi như sau:

```
void main()
```

```
{
```

```
    diem D1;//D1 là tên đối tượng
```

```
    D1.nhapd();//lời gọi đến phương thức
```

```
    //hoặc D1.diem::nhapd();
```

```
}
```

Bài tập số 1

- Xây dựng lớp Phân số và các phương thức của lớp phân số
 - Phương thức nhập, xuất
 - Phương thức cộng, trừ, nhân, chia
 - Phương thức rút gọn, phủ định, nghịch đảo.

2.5. Hàm tạo và hàm hủy

2.5.1 Hàm tạo:

a. Định nghĩa:

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

2.5.1 Hàm tạo:

b. Cú pháp khai báo và xây dựng hàm tạo

Cú pháp:

```
Tên_lớp([tham số])  
{  
    //nội dung hàm tạo  
}
```

Nhận xét:

- Hàm tạo có **tên_hàm** trùng **Tên_lớp** không bắt đầu bằng từ khóa **void** hay **KDL**

2.5.1 Hàm tạo:

Nhận xét:

- Hàm tạo có **tên_hàm** trùng **Tên_lớp** không bắt đầu bằng từ khóa **void** hay **KDL**.
- Hàm tạo phải được khai báo bên trong lớp
- Hàm tạo có thể được xây dựng bên trong hay bên ngoài lớp.
- Hàm tạo có hoặc không có tham số truyền vào.

2.5.1 Hàm tạo:

c. Ví dụ: xây dựng hàm tạo cho lớp điểm

```
class diem
```

```
{ private:
```

```
    int x,y;//toa do x va toa do y
```

```
    public:
```

```
    diem() //hàm tạo không đối
```

```
    { x=0;y=0;}
```

```
    diem (int x1, int y1)//hàm tạo có đối
```

```
    {x=x1;y=y1;}
```

```
//Khai báo hai phương thức khác
```

```
};
```

2.5.1 Hàm tạo:

c. Phân loại hàm tạo

- Hàm tạo không đối

`diem()` //hàm tạo không đối

`{ x=0;y=0;}`

- Hàm tạo có đối

`diem (int x1, int y1)`//hàm tạo có đối

`{x=x1;y=y1;}`

- Hàm tạo sao chép

`diem (const diem &A)`//hàm tạo sao chép

`{x=A.x;y=A.y;}`

2.5.1 Hàm tạo:

d. Hàm tạo có đối số mặc định

```
Tên_lớp(KDL thamso1=gtr1, KDL thamso2=gtr2...)  
{  
    //nội dung  
}
```

Ví dụ

```
diem (int x1=0, int y1=0)//hàm tạo có đối số mặc định  
    {x=x1;y=y1;}
```

2.5.1 Hàm tạo:

e. Lời gọi hàm tạo

- Hàm tạo không cần phải gọi.
- Hàm tạo được tự động gọi tới khi khai báo đối tượng

2.5.2 Hàm hủy:

a. Định nghĩa:

- Hàm hủy là hàm hủy bỏ bộ nhớ của đối tượng được cấp phát bởi hàm tạo.

b. Cách khai báo và xây dựng

Cú pháp

```
~Tên_lớp()
```

```
{
```

```
    //nội dung
```

```
}
```

- Hàm hủy được khai báo bên trong lớp
- Hàm hủy được xây dựng bên trong hay bên ngoài lớp.

2.5.2 Hàm hủy:

c. Ví dụ

```
class diem
{ private:
    int x,y;//toa do x va toa do y
    public:
    diem() //hàm tạo không đối
    { x=0;y=0;}
    diem (int x1, int y1)//hàm tạo có đối
    {x=x1;y=y1;}
    //Khai báo hai phương thức khác
    ~diem()//khai bao và xay dung ham huy
    {x=0;y=0;}
};
```


2.5.2 Hàm hủy:

d. Đặc điểm

- Mỗi lớp có duy nhất một hàm hủy
- Hàm hủy không có đối
- Hàm hủy không cần được triệu gọi, sẽ tự động được gọi khi kết thúc chương trình hay đối tượng bị hủy.

2.5.3 Một số lưu ý khi viết hàm tạo và hàm hủy

1. Các lớp không có thành viên là kiểu con trỏ hay mảng thì không nhất thiết phải xây dựng hàm tạo, hàm hủy. Chương trình C++ sẽ tự phát sinh hàm tạo, hàm hủy không đối số và không làm gì cả.
2. Các lớp có thành viên kiểu con trỏ, mảng nhất thiết phải xây dựng hàm tạo, hàm hủy để cấp phát (toán tử new) và hủy bộ nhớ (toán tử delete).

Bài tập

1. Xây dựng lại lớp Diem với các hàm tạo và hàm hủy.
2. Xây dựng lại lớp phân số với các hàm tạo và hàm hủy
3. Xây dựng lớp hình chữ nhật tạo bởi chiều dài và chiều rộng, với các hàm tạo, hàm hủy, phương thức: nhập, xuất, tính diện tích, chu vi và kiểm tra đó có phải là hình chữ nhật không?

Bài tập

4. Xây dựng lớp đa thức. Với các hàm tạo, hàm hủy, phương thức nhập, xuất, cộng, trừ hai đa thức.

Chương II. Lớp và Đối tượng(tiếp)

2.6. Hàm bạn và lớp bạn

2.6.1. Hàm bạn

a. Xét ví dụ sau:

```
class diem
{
    private:
    int x,y;//toa do x va toa do y
    public:
    void nhapd();//phuong thuc nhap
};
//kết thúc khai báo lớp
```

2.6.1. Hàm bạn

```
float KC2D(diem D1, diem D2)
```

```
{  
    return sqrt((D1.x-D2.x)*(D1.x-D2.x)+(D1.y-D2.y)*(D1.y-D2.y));  
}
```

Thấy xuất hiện lỗi không cho phép truy cập thuộc tính x và y.

Nguyên nhân: x, y là thành phần private, một hàm thông thường hay tại hàm chính không truy cập trực tiếp.

Giải pháp:

- + Thay đổi thuộc tính của x,y là public (không tốt)
- + Khai báo hàm KC2D dưới danh nghĩa hàm bạn của lớp

2.6.1. Hàm bạn

a. Định nghĩa:

- Hàm bạn của lớp là hàm được khai báo bên trong lớp bắt đầu bằng từ khóa **friend**. Được xây dựng bên ngoài lớp. Được phép truy cập vào bất cứ thành viên nào của lớp mà nó nhận là bạn.
- Một hàm có thể là bạn của nhiều lớp.

b. Xây dựng hàm bạn

2.6.1. Hàm bạn

```
class Tên_lớp
{
    private:
    public:
    friend void|KDL tên_hàm([tham số]);//khai báo hàm bạn bên trong lớp
}
//xây dựng nội dung bên ngoài
void|KDL tên_hàm([tham_số])
{
    //nội dung hàm
}
```


2.6.1. Hàm bạn

c.Ví dụ:

```
class diem
```

```
{
```

```
    private:
```

```
    int x,y;//toa do x va toa do  
    y
```

```
    public:
```

```
    void      nhapd();//phuong  
    thuc nhap
```

```
    friend float  KC2D(diem  D1,  
        diem D2)
```

```
};
```

```
//kết thúc khai báo lớp
```

```
float KC2D(diem D1, diem D2)
```

```
{
```

```
    return      sqrt((D1.x-  
        D2.x)*(D1.x-D2.x)+(D1.y-  
        D2.y)*(D1.y-D2.y));
```

```
}
```

2.6.1. Hàm bạn

- d. Sự khác nhau giữa hàm bạn và hàm thành viên của lớp(phương thức)
- Hàm bạn bắt buộc phải xây dựng bên ngoài lớp không thông qua tham chiếu của lớp.
 - Hàm bạn được gọi không cần thông qua đối tượng của lớp đó.

2.6.2. Lớp bạn

a. Định nghĩa:

- **Nếu lớp A được khai báo là bạn của lớp B** thì tất cả các phương thức của A đều có thể truy nhập đến các thành phần riêng của lớp B. Một lớp có thể là bạn của nhiều lớp khác. Cũng có thể khai báo A là bạn của B và B là bạn của A.
- Một lớp có thể là bạn của vô số lớp.

2.6.2. Lớp bạn

b. Cách khai báo lớp bạn:

Giả sử có 3 lớp A, B và C. Để khai báo lớp này là bạn của lớp kia, ta viết theo mẫu sau:

// Khai báo trước các lớp

class A;

class B ;

class C;

// Định nghĩa các lớp

class A

{

...

friend class B ; // Lớp B là bạn của A

friend class C ; // Lớp C là bạn của A

...

};

class B

{

...

friend class A ; // Lớp A là bạn của B

friend class C ; // Lớp C là bạn của B

...

};

class C

{

...

friend class B ; // Lớp B là bạn của C

...

};

2.6.2. Lớp bạn

c. Chú ý: quan hệ bạn chỉ là quan hệ hai chiều khi mà khai báo bạn ở cả hai lớp.

d. Ví dụ :

2.7. Chồng hàm và chồng toán tử

2.7.1. Chồng hàm:

a. Định nghĩa:

- Là khả năng các hàm có thể trùng tên nhau nhưng khác nhau về:

- Kiểu dữ liệu trả về
- Kiểu dữ liệu của tham số
- Số lượng tham số truyền vào.

2.7.1. Chồng hàm

b. Ví dụ:

```
void hoanvi(int &a, int &b)
{
    int tg;
    tg=a;a=b;b=tg;
}
```

```
void hoanvi(float &a, float
    &b)
{
    float tg;
    tg=a;a=b;b=tg;
}
```

2.7.1. Chồng hàm

c. Nhận xét:

- Khả năng chương trình cho phép các hàm trùng tên nhưng khác nhau về kiểu dữ liệu trả về, tham số truyền vào, số lượng tham số truyền gọi là khả năng chồng hàm.
 - Chính vì khả năng này, khi kế thừa các lớp có sẵn ta không còn sợ sự trùng tên không cho phép như ở một số ngôn ngữ: PASCAL, VISUAL Basic...
-

2.7.2. Chồng toán tử

a. Định nghĩa

- Là khả năng một toán tử có thể thực hiện việc tính toán ứng với nhiều kiểu dữ liệu khác nhau.

b. Ví dụ:

2.7.2. Chồng toán tử

```
void main()
```

```
{
```

```
    int a,b,c;
```

```
    c=a+b;
```

```
}
```

```
void main()
```

```
{
```

```
    float a,b,c;
```

```
    c=a+b;
```

```
}
```

■ Nhận xét:

Cùng toán tử + thực hiện cộng được kiểu dữ liệu số thực, lẫn kiểu dữ liệu số nguyên.

2.7.2. Chồng toán tử

c. Cách xây dựng toán tử chồng:

* Xây dựng phương thức toán tử chồng:

KDL operator toán_tử([Tham số])

{

//Nội dung

}

Ví dụ: Xây dựng phương thức toán tử + cho lớp phân số

2.7.2. Chồng toán tử

Trước đây

```
class PS
{
    private:
        int ts,ms;
    public:
        void nhap();
        void xuat();
        PS cong(PS P);
}; //lop PS
```

Bây giờ

```
class PS
{
    private:
        int ts,ms;
    public:
        void nhap();
        void xuat();
        PS operator+(PS P);
}; //lop PS
```

2.7.2. Chồng toán tử

```
PS PS::Cong(PS P)
{
    PS k;
    k.ts=ts*P.ms+ms*P.ts;
    k.ms=ms*P.ms;
    return k;
}
```

```
Void main()
{   PS A,B,C;
    A.nhap();B.nhap();
        C=A.Cong(B);

}
```

```
PS PS::operator+(PS P)
{
    PS k;
    k.ts=ts*P.ms+ms*P.ts;
    k.ms=ms*P.ms;
    return k;
}
```

```
Void main()
{PS A,B,C;
    A.nhap();B.nhap();
        C=A+B;

}
```

Bài tập

Hãy xây dựng lại lớp số phức, lớp phân số. Với các toán tử chồng: $+$, $-$, $*$, $/$

Hãy xây dựng lại lớp đa thức với các toán tử $+$, $-$ hai đa thức

2.7.3. Toán tử chồng tăng trước, tăng sau, giảm trước, giảm sau

2.7.4. Toán tử chồng nhập xuất

- Riêng đối với toán tử `<<`, `>>` ta không thể xây dựng dưới dạng toán tử thông thường:
 - Không thể là phương thức toán tử của lớp
 - Không thể là hàm toán tử thông thường
- Phải xây dựng như sau:
 - Toán tử xuất dữ liệu:
`ostream & operator<<(ostream &os, KDL bien)`
{
 //Cac cau lenh xuat du lieu
 return os;
}

2.7.4. Toán tử chồng nhập xuất

- Toán tử nhập dữ liệu:

```
istream & operator>>(istream &is, KDL &bien)
```

```
{
```

```
    //Cac cau lenh xuat du lieu
```

```
    return is;
```

```
}
```

- Để xây dựng toán tử nhập xuất cho một lớp: ta phải khai báo toán tử nhập xuất là bạn của lớp.
- Ví dụ: Xây dựng toán tử nhập xuất cho lớp phân số

2.7.4. Toán tử chồng nhập xuất

Trước đây

```
class PS
{
    private:
    int ts,ms;
    public:
    void nhap();
    void xuat();
}; //lop PS

void PS::nhap()
{
    cout<<"ts=";<<cin>>ts;
    cout<<"ms=";<<cin>>ms;
}
```

Bây giờ

```
class PS
{
    private:
    int ts,ms;
    public:
    friend istream &operator>>(istream &is, PS &P);
    friend ostream &operator<<(ostream &os, PS P);
}; //lop PS

istream &operator>>(istream &is, PS &P)
{
    cout<<"ts=";<<is>>P.ts;
    cout<<"ms=";<<is>>P.ms;
    return is;
}
```

2.7.4. Toán tử chồng nhập xuất

Trước đây

```
void PS::xuat()  
{  
    cout<<ts<<"/"<<ms;  
}
```

```
void main()  
{  
    PS A;  
    A.nhap();  
    A.xuat();  
    getch();  
}
```

Bây giờ

```
ostream &operator<<(ostream &os, PS P)  
{  
    os<<P.ts<<"/"<<P.ms;  
    return os;  
}
```

```
void main()  
{  
    PS A;  
    cin>>A;  
    cout<<A;  
    getch();  
}
```

Bài tập

- Hãy xây dựng lại lớp số phức, lớp phân số.
Với các toán tử $<<, >>$
- Hãy xây dựng lại lớp đa thức với các toán tử
 $<<, >>$
- Xây dựng lớp số nguyên với toán tử
 $<<, >>, ++, --$

Chương II. Lớp và Đối tượng(tiếp)

2.8. Lớp bao

a. Định nghĩa:

- Một lớp có thuộc tính là đối tượng của lớp khác gọi là lớp bao.

ví dụ:

```
class A
```

```
{
```

```
private:
```

```
int a, b;
```

```
...
```

```
};
```

```
class B
```

```
{
```

```
private:
```

```
double x, y, z;
```

```
...
```

```
};
```

```
class C
```

```
{
```

```
private:
```

```
int m, n;
```

```
A u;
```

```
B p, q;
```

```
...
```

```
};
```

Trong ví dụ trên thì:

C là lớp bao

A, B là các lớp thành phần (của C)

2.8. Lớp bao

b. Lưu ý:

- Trong các phương thức của lớp bao không cho phép truy nhập trực tiếp đến các thuộc tính của các đối tượng của các lớp thành phần.

2.8. Lớp bao

c. Xây dựng hàm tạo cho lớp bao:

- Khi xây dựng hàm tạo của lớp bao, phải sử dụng các hàm tạo của lớp thành phần để khởi gán cho các đối tượng thành phần của lớp bao.
- **Ví dụ** khi xây dựng hàm tạo của lớp C, cần dùng các hàm tạo của lớp A để khởi gán cho đối tượng thành phần u và dùng các hàm tạo của lớp B để khởi gán cho các đối tượng thành phần p, q.

2.8. Lớp bao

- Cú pháp:

tên_lớp(danh sách đối) : tên_đối_tượng(danh
sách giá trị), ... ,tên_đối_tượng(danh sách
giá trị)

{

// Các câu lệnh trong thân hàm tạo

}

- Ví dụ:

2.8. Lớp bao

```
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;
}
};
```

2.8. Lớp bao

- d. Sử dụng các phương thức của lớp thành phần.
 - Gọi đến phương thức của lớp thành phần giống như là lời gọi phương thức ở chương trình chính thông qua tên đối tượng.

Bài tập

Xây dựng lớp Điểm trong hệ toạ độ xOy : với hàm tạo, hàm hủy, toán tử nhập xuất, dịch chuyển, tính khoảng cách hai điểm.

Xây dựng lớp Tam giác là lớp bao của lớp Điểm (Tạo bởi 3 điểm): với các hàm tạo, hàm hủy, toán tử nhập xuất, tính chiều dài 3 cạnh, tính diện tích, chu vi.

Xây dựng lớp Hình chữ nhật là lớp bao của lớp Điểm (Tạo bởi 1 điểm và 2 cạnh): với các hàm tạo, hàm hủy, toán tử nhập xuất, tính diện tích, chu vi.