

## **PART 2. LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG TRONG C++**

### **Nội dung.**

- 2.1. Một số nguyên lý cơ bản của lập trình hướng đối tượng
- 2.2. Biểu diễn đối tượng (Object Representation)
- 2.3. Kế thừa (Inheritance)
- 2.4. Xử lý quá tải (Overloading)
- 2.5. Xử lý đa hình (Polymorphism)
- 2.6. Trừu tượng dữ liệu (data abstraction)
- 2.7. Đóng gói (Encapsulation)
- 2.8. Thiết kế lớp (Class Design)
- 2.9. CASE STUDY

## 2.1. Các nguyên lý cơ bản của lập trình hướng đối tượng

**Đối tượng (Object)**: là đơn vị cơ sở của lập trình hướng đối tượng, trong đó dữ liệu và các hàm xử lý trên dữ liệu được gói chung như một đơn vị gọi là đối tượng. Trên thực tế, đối tượng còn có nghĩa là vật chất “things”. Đó là vật chất khi để khám phá nó ta cần phải biết nó có gì (*data*) và nó thực hiện được những gì (*method, function*).

**Lớp (class)** : chỉ đơn thuần là một từ khóa dùng để *biểu diễn đối tượng* (*định nghĩa đối tượng, thiết kế đối tượng*). Khi biểu diễn đối tượng (*định nghĩa, biểu diễn hoặc thiết kế*) trong một lớp ta cần mô tả đối tượng đó có những đặc trưng gì (**data abstraction**) và những hành vi nào của đối tượng áp đặt trên các đặc trưng đó (**functional abstraction**).

**Trùu tượng dữ liệu (Data Abstraction)**: trùu tượng dữ liệu hàm ý việc cung cấp thông tin cần thiết ra thế giới bên ngoài (*thế giới của các đối tượng khác*) và ẩn dấu đi những thông tin riêng biệt. Ví dụ, các hệ cơ sở dữ liệu ẩn dấu đi việc làm thế nào để tạo lập, lưu trữ, và duy trì CSDL. Một class cung cấp các phương thức khác nhau ra thế giới bên ngoài nhưng ẩn dấu đi những đặc trưng riêng của khi xử lý dữ liệu và các phương thức cục bộ

**Tính đóng gói (Encapsulation)**: đặt dữ liệu và các hàm thích hợp cùng trong một đối tượng.

**Tính kế thừa (Inheritance)**: Một trong những đặc trưng quan trọng của OOP là sử dụng lại code có trước. Phép kế thừa cho phép định nghĩa một lớp (đối tượng) có nguồn gốc từ một hoặc nhiều lớp có trước được gọi là lớp cơ sở.

**Tính đa trạng thái (Polymorphism)**: Hình thái của các phép toán hoặc các hàm có thể được thay đổi theo mỗi đối tượng khác nhau.

**Tính chịu tải (Overloading)**: Giống như Polymorphism, mỗi khi có sự thay đổi về hình thái dẫn đến sự thay đổi về dữ liệu và phương thức thích hợp.

## 2.2. Biểu diễn đối tượng

**Định nghĩa lớp (class):** khi định nghĩa một class, về bản chất là ta định hình nên các đặc trưng thuộc tính (dữ liệu) và hành vi phản ánh đối tượng ở thế giới thực. Nói cách khác ta xây dựng một ánh xạ từ tập các đối tượng ở thế giới thực thành tập đặc trưng thuộc tính và hành vi của đối tượng trong hệ thống máy tính.

**Biểu diễn lớp:** định hình nên đối tượng có những gì (data) và làm được những gì (method).

```
class <class-name> {  
    <data type 1>      member1;  
    <data type 2>      member2;  
    .....  
    <data type N>      memberN;  
};
```

**Ví dụ :**

```
class Box { //lớp có tên là Hộp  
    double length; //đặc trưng chiều dài của hộp  
    double breadth; // đặc trưng chiều rộng của hộp  
    double height; // đặc trưng chiều cao của hộp  
};
```

**Khai báo đối tượng:** class đưa ra hình hài của một đối tượng trong hệ thống máy tính. Khi đó:

**Box      Box1, Box2;** //định nghĩa hai đối tượng Box1, Box2 có kiểu Box.

**Truy nhập đến thành viên đối tượng:** để truy nhập đến một thành viên của đối tượng ta chỉ cần sử dụng toán tử (.). Ví dụ: *Box1.length = 3 hoặc Box2.height = 5;*

**Ví dụ:** Thao tác với class.

```
#include <iostream>
using namespace std;
class Box {//Biểu diễn hoặc định nghĩa hình hài của một hình khối chữ nhật
    public: // từ khóa dùng để xác định quyền truy nhập đến các thành viên
        float length; //đặc trưng chiều dài
        float breadth; //đặc trưng chiều rộng
        float height; //đặc trưng độ cao
};
int main(void ) {
    Box Box1, Box2; //Box1, Box2 trở thành hai đối tượng kiểu Box
    Box1.length = 5.0; //Truy nhập đến thành viên length của Box1
    Box1.breadth = 6.0; //Truy nhập đến thành viên breadth của Box1
    Box1.height = 7.0; //Truy nhập đến thành viên height của Box1
    Box2.length = 10.0;
    Box2.breadth = 11.0;
    Box2.height = 12.0;
    float Volume;//Tính thể tích của hình khối chữ nhật
    Volume = Box1.length*Box1.breadth*Box1.height;
    cout<<"Thể tích Box1:"<<Volume<<endl;
    Volume = Box2.length*Box2.breadth*Box2.height;
    cout<<"Thể tích Box2:"<<Volume<<endl;
    system("PAUSE"); return 0;
}
```

**Ví dụ:** Thao tác với class members.

```
#include <iostream>
using namespace std;
class Box {//Biểu diễn lớp Box
public:
    float length, breadth, height;
    float Volume( float x, float y, float z) {
        return (x*y*z);
    }
};
int main(void ) {
    Box Box1, Box2; //Box1, Box2 trở thành hai đối tượng kiểu Box
//Thiết lập dữ liệu cho Box1
    Box1.length = 5.0;Box1.breadth = 6.0;Box1.height = 7.0;
    float V = Box1.Volume(Box1.length,Box1.breadth, Box1.height);
    cout<<"Thể tích Box1:"<<V<<endl;
//Thiết lập dữ liệu cho Box2
    Box2.length = 10.0;Box2.breadth = 11.0;Box2.height = 12.0;
    V = Box2.Volume(Box2.length,Box2.breadth, Box2.height);
    cout<<"Thể tích Box2:"<<V<<endl;
    V = Box2.Volume(3.0,4.0, 5.0);
    cout<<"Thể tích Box2:"<<V<<endl;
    system("PAUSE");
}
```

## Một số khái niệm về lớp

Class member functions	Được định nghĩa giống như các thành viên dữ liệu nhưng chỉ khác đó là một hàm. Đối với lập trình hướng đối tượng người ta còn gọi là phương thức (method). Ta sẽ xem xét nội dung này trong Mục 2.2.1.
Class assess modifier	Mỗi thành viên của lớp được qui định một trong các kiểu truy nhập: public, private, protected. Nội dung này sẽ được đề cập trong Mục 2.2.2
Constructor – Destructor	Constructor là một hàm đặc biệt được gọi đến mỗi khi có một đối tượng mới của lớp được tạo lập. Destructor là một hàm đặc biệt được gọi đến mỗi khi có một đối tượng bị loại bỏ. Nội dung này sẽ được đề cập trong Mục 2.2.3.
Copy constructor	Là một constructor tạo chung cho tất cả các đối tượng thuộc cùng một lớp đã được tạo lập trước đó. Nội dung này sẽ được đề cập trong Mục 2.2.4.
Friend functions	Một hàm friend của lớp được chấp thuận truy nhập đầy đủ đến các thành viên private và protected của lớp. Nội dung này sẽ được đề cập trong Mục 2.2.5.
Inline function	Hàm được gọi theo kiểu inline sẽ cố gắng mở rộng mã của hàm đến các lời gọi khác nhau. Nội dung này sẽ được đề cập trong Mục 2.2.6.
The this pointer	Mỗi đối tượng có một con trỏ <i>this</i> có trỏ đến chính nó. Nội dung này sẽ được đề cập trong Mục 2.2.7.
Pointer to class	Thực hiện giống như con trỏ của lập trình cấu trúc. Nội dung này sẽ được đề cập trong Mục 2.2.8.
Static members	Cả hai thành viên dữ liệu và hàm đều có thể được biểu thị như các thành phần tĩnh. Nội dung này sẽ được đề cập trong Mục 2.2.9.

## 2.2.1. Các thành viên của đối tượng

Khi biểu diễn (khai báo) đối tượng, các thành viên của đối tượng được chia thành hai loại : *thành viên dữ liệu* và *hành vi của đối tượng thực hiện trên dữ liệu của đối tượng*.

**Thành viên dữ liệu** : là các đặc trưng thông tin phản ánh đối tượng. Ví dụ trong class Box được định nghĩa ở trên, thành viên length, height, breadth là các đặc trưng thông tin mô tả một hình hộp chữ nhật. Thông qua các đặc trưng dữ liệu ta có thể nhận biết được đối tượng có những thông tin gì.

**Hành vi của đối tượng**: là các đặc trưng thông tin phản ánh đối tượng. Ví dụ với hình hộp chữ nhật được định nghĩa trong class Box với ba thành viên dữ liệu, ta cần phải hình dung ra tất cả những thao tác có thể thực hiện trên các đặc trưng length, breadth, height. Các thao tác đó có thể là : thiết lập chiều dài hình hộp (SetLength), thiết lập chiều rộng hình hộp (SetBreadth), thiết lập chiều cao hình hộp (SetHeight), tính thể tích hình hộp (Volume).

**Ví dụ: Định nghĩa các thành viên của class Box.**

```
class Box {  
    Public:  
        float length;//định nghĩa chiều dài hình hộp  
        float breadth;//định nghĩa chiều rộng hình hộp  
        float height;//định nghĩa chiều cao hình hộp  
        SetLength( float x); //Phương thức hay hàm thiết lập độ dài hộp  
        SetBreadth( float y); //Phương thức hay hàm thiết lập độ rộng hộp  
        SetHeight( float x); //Phương thức hay hàm thiết lập chiều cao hộp  
        Volume(void); //Phương thức hay hàm tính thể tích hộp  
};
```

**Mô tả chi tiết các thành viên hàm (phương thức) của lớp:** để mô tả các thành viên của lớp ta có thể thực hiện bằng hai cách: mô tả bên trong lớp hoặc mô tả bên ngoài lớp. Dưới đây là các cách mô tả chi tiết các hàm của lớp.

**Ví dụ: Cách thứ nhất: Mô tả trực tiếp trong lớp.**

```
class Box {  
    public:  
        float length;//định nghĩa chiều dài hình hộp  
        float breadth;//định nghĩa chiều rộng hình hộp  
        float height;//định nghĩa chiều cao hình hộp  
        void SetLength( float x) { //Phương thức hay hàm thiết lập độ dài hộp  
            length =x;  
        }  
        void SetBreadth( float y){//Phương thức hay hàm thiết lập độ rộng hộp  
            breadth = y;  
        }  
        void SetHeight( float x) { //Phương thức hay hàm thiết lập chiều cao hộp  
            height = z;  
        }  
        float Volume(void) { ;//Phương thức hay hàm tính thể tích hộp  
            return (length *breadth*height);  
        }  
};
```

## Ví dụ. Cách biểu diễn lớp thứ nhất.

```
#include <iostream>
using namespace std;
class Box {          //Biểu diễn lớp Box
public:
    float length, breadth, height; //Thành phần dữ liệu của lớp
    void SetLength( float x ) { length = x; } //Mô tả phương thức SetLength
    void SetBreadth( float y ) { breadth = y; } //Mô tả phương thức SetBreadth
    void SetHeight( float z ) { height = z; } //Mô tả phương thức SetHeigth
    float Volume( void ) { //Mô tả phương thức SetVolume
        return (length*breadth*height);
    }
};
int main(void ) {
    Box Box1, Box2;//Box1, Box2 là hai biến kiểu Box
    Box1.SetLength(4.0);Box2.SetLength(10.0);
    Box1.SetBreadth(5.0);Box2.SetBreadth(11.0);
    Box1.SetHeight(6.0);Box2.SetHeight(12.0);
    float V = Box1.Volume();
    cout<<"The tich Box1:"<<V<<endl;
    V = Box2.Volume();
    cout<<"The tich Box2:"<<V<<endl;
    system("PAUSE"); return 0;
}
```

**Cách thứ hai: Mô tả bên ngoài lớp.**

**Cú pháp mô tả hàm bên ngoài lớp:**

```
Kiểu-hàm Tên-lớp :: Tên-Hàm ( đối của hàm) {  
    <Thân hàm:  
        return (giá trị);  
}
```

**Ví dụ:**

```
class Box {  
    public:  
        float length;//định nghĩa chiều dài hình hộp  
        float breadth;//định nghĩa chiều rộng hình hộp  
        float height;//định nghĩa chiều cao hình hộp  
        void SetLength( float x ); //Phương thức ( hàm ) thiết lập độ dài hộp  
        void SetBreadth( float ); //Phương thức ( hàm ) thiết lập độ rộng hộp  
        void SetHeight( float ); //Phương thức ( hàm ) thiết lập chiều cao hộp  
        float Volume(void) //Phương thức ( hàm ) tính thể tích hộp  
};  
void Box :: SetLength( float x ) { length = x; } //Mô tả chi tiết phương thức SetLength  
void Box :: SetBreadth( float y ) { breadth = y; } //Mô tả chi tiết phương thức SetBreadth  
void Box :: SetHeight( float z ) { height = z; } //Mô tả chi tiết phương thức SetHeight  
float Box :: Volume( void ) {  
    return (length * breadth*height);  
}
```

## Ví dụ. Cách biểu diễn lớp thứ 2.

```
#include <iostream>
using namespace std;
class Box { // Biểu diễn lớp Box
public:
    float length, breadth, height;
    void SetLength( float );
    void SetBreadth( float );
    void SetHeight( float z );
    float Volume( void );
};

void Box:: SetLength( float x){ length = x; } // Mô tả hàm SetLength bên ngoài lớp
void Box:: SetBreadth( float y){ breadth = y; } // Mô tả hàm SetBreadth bên ngoài lớp
void Box:: SetHeight( float z){ height = z; } // Mô tả hàm SetHeight bên ngoài lớp
float Box::Volume( void){ return ( length * breadth * height); }

int main(void ) {
    Box Box1, Box2; // Box1, Box2 bao gồm dữ liệu và hàm
    Box1.SetLength(4.0);Box1.SetBreadth(5.0);Box1.SetHeight(6.0);
    Box2.SetLength(10.0);Box2.SetBreadth(11.0);Box2.SetHeight(12.0);
    float V = Box1.Volume(); cout<<"The tich Box1:"<<V<<endl;
    V = Box2.Volume(); cout<<"The tich Box2:"<<V<<endl;
    system("PAUSE");
}
```

## 2.2.2. Quyền truy nhập đến các thành viên của đối tượng

Ẩn dấu dữ liệu, ẩn dấu chức năng là một trong những đặc trưng quan trọng của lập trình hướng đối tượng. Lập trình OOP cho phép ngăn cản các hàm của một chương trình truy nhập trực tiếp bên trong biểu diễn của lớp. Hạn chế quyền truy nhập đến mỗi thành viên của lớp (dữ liệu, hàm) có thể được định nghĩa một trong các từ khóa: public (tổng thể), private (cực bô), protected (bảo vệ). Chế độ ngầm định khi không có chỉ thị gì là private.

**public:** một hàm thành viên hoặc biến dữ liệu thành viên được định nghĩa ngay sau từ khóa public sẽ được phép truy nhập ở bất kỳ vị trí nào trong chương trình. Ta có thể thiết lập, lấy giá trị, thay đổi nội dung của biến, hoặc thực hiện hàm từ bên ngoài class.

**private:** một hàm thành viên hoặc biến dữ liệu thành viên được định nghĩa ngay sau từ khóa private sẽ chỉ được phép truy nhập trong nội bộ lớp. Không được phép thay đổi, thiết lập hoặc thực hiện truy nhập từ bên ngoài lớp. Chỉ có duy nhất các lớp được định nghĩa là lớp bạn (friend class) hoặc hàm bạn (friend function) mới được phép truy nhập đến các thành viên private. *Lớp bạn và hàm bạn ta sẽ đề cập đến trong những mục sau của bài học.*

**protected:** một hàm thành viên hoặc biến dữ liệu thành viên được định nghĩa ngay sau từ khóa protected cùng giống như thành viên private. Tuy vậy, các thành viên protected thêm vào quyền truy nhập từ lớp con của lớp cơ sở vào thành viên protected. *Nội dung này sẽ được đề cập đến trong nội dung lớp kế thừa.*

## Ví dụ về truy nhập thành viên public của lớp:

```
#include <iostream>
using namespace std;
class Box {//Biểu diễn lớp Box
public:
    float length, breadth, height; //thành viên dữ liệu có quyền truy cập public
    float Volume( void); //Hàm thành viên có quyền truy cập public
}
float Box::Volume( void){
    return ( length * breadth * height);
}
int main(void ) {
    Box Box1, Box2;//Khai báo Box1, Box2 là hai biến kiểu Box
    //Thiết lập giá trị các thành viên public từ bên ngoài class
    Box1.length=4.0;Box1.breadth = 5.0;Box1.height=6.0;
    //Thiết lập giá trị các thành viên public từ bên ngoài class
    Box2.length=10.0;Box2.breadth = 11.0;Box2.height=12.0;
    float V = Box1.Volume();cout<<"The tích Box1:"<<V<<endl;
    //Truy cập hàm thành viên public từ bên ngoài class
    V = Box2.Volume();cout<<"The tích Box2:"<<V<<endl;
    system("PAUSE");
}
```

## Ví dụ về truy nhập thành viên private và protected của lớp:

```
#include <iostream>
using namespace std;
class Box {//Bieu dien lop Box
    private:
        float length, breadth; //thành viên private
    protected:
        float height; // thành viên protected
    public: float Volume( void); //thành viên public
};

float Box::Volume( void){
    cout<<"Nhập chiều dài:"; cin>>length; //OK: Truy nhập bên trong lớp
    cout<<"Nhập chiều rộng:"; cin>>breadth; //OK: Truy nhập bên trong lớp
    cout<<"Nhập chiều cao:"; cin>>height; //OK: Truy nhập bên trong lớp
    return ( length * breadth * height);
}

int main(void ) {    Box Box1;//Box1 bao gom ca du lieu va ham
    //Box1.length = 10; Box1.breadth = 10; //Error : Truy nhập bên ngoài lớp
    //Box1.breadth = 10; //Error : Truy nhập bên ngoài lớp
    //Box1.height = 10; //Error : Truy nhập bên ngoài lớp
    float V = Box1.Volume(); // OK:truy nhập vào thành phần public
    cout<<"The tích Box1:"<<V<<endl;
    system("PAUSE"); return 0;
}
```

## Ví dụ về truy nhập hàm thành viên private và protected của lớp:

```
#include <iostream>
using namespace std;
class Box {//Bieu dien lop Box
    private: float length, breadth,height;
        //Truy nhập bên trong lớp
    void SetLength(float x){cout<<"Nhập x="; cin>>x; length =x;}
    void SetBreadth(float y){cout<<"Nhập y="; cin>>y; breadth =y;}
    void SetHeight(float z){cout<<"Nhập z="; cin>>z; height =z; }
public://Truy nhap ben trong lop
    float Volume( void){    int x, y, z;
        SetLength(x); SetBreadth(x); SetHeight(x); //OK: Truy nhập bên trong lớp
        return ( length *breadth*height);
    }
};

int main(void ) {
    Box Box1;float x, y, z;//Box1 bao gom ca du lieu va ham
    //Box1.SetLength(x); //Error : Truy nhập bên ngoài lớp
    //Box1.SetBreadth(y);// Error : Truy nhập bên ngoài lớp
    //Box1.Height(z); // Error : Truy nhập bên ngoài lớp
    float V = Box1.Volume();// OK:Truy nhập thành phần public
    cout<<"The tích Box1:"<<V<<endl;
    system("PAUSE"); return 0;
}
```

## BÀI TẬP. CASE STUDY 1 (Ver 2.0): Tạo lập các class.

### I. Xây dựng tập thao tác với số nguyên

```
class INT {  
    private:  
        long n; //số tự nhiên n  
        long Init(long ); //khởi tạo số n  
        int Tong( long); //tính tổng các chữ số  
        void DoiSo( long, int ); //đổi n thành số hệ cơ số b  
        void PTS (long ); //phân tích n thành tích các số nguyên tố  
        int TestNgto(long);// Kiểm tra tính nguyên tố của n  
        int TesDx( long); //Kiểm tra tính đối xứng của n  
        void ListNt (long); //liệt kê các số nguyên tố nhỏ hơn n  
        void ListDx (long); //liệt kê các số đối xứng nhỏ hơn n  
        void ListHn (long); //liệt kê các cặp số hữu nghị nhỏ hơn n  
        void ListHh (long); //liệt kê các số hoàn hảo nhỏ hơn n  
        void ListNgt(long, int); //tập các số ng.tố có tổng là S  
        void ListDx(long, int); //tập các số đối xứng ở hệ cơ số b  
    public:  
        Void Function(void);  
};
```

## BÀI TẬP. CASE STUDY 1 (Ver 2.0): Tạo lập các class.

### II. Xây dựng tập thao tác với xâu ký tự

```
class STRING {
```

```
    private:
```

```
        char *s; //khai báo một con trỏ đến string
```

```
        void InitString( void );//Khởi tạo một dòng
```

```
        int StrLen( char * );//Tìm độ dài string
```

```
        char *ToLower( char * );//Đổi string sang in thường
```

```
        char *ToUpper( char * );// Đổi string sang in hoa
```

```
        char Code( unsigned char ); //Mã hóa chẵn lẻ ký tự
```

```
        char Decode( char ); //Giải mã ký tự bằng kỹ thuật chẵn lẻ
```

```
        void ShowWord( char * );//Tìm tập từ và số lần xuất hiện mỗi từ
```

```
        int Search_Tu( WORD *, int , char * ); //Tìm từ trên tập từ
```

```
        int Search_Char( CHAR *, int , char ); //Tìm ký tự trên tập ký tự
```

```
        void ShowChar( char * );//Tìm tập ký tự và số lần xuất hiện mỗi ký tự
```

```
        char *CodeStr( char * );//Mã hóa string theo kỹ thuật chẵn lẻ
```

```
        char *DecodeStr( char * );//giải mã string theo kỹ thuật chẵn lẻ
```

```
    public:
```

```
        void Function( void ); //Test các hàm thành viên
```

```
};
```

## BÀI TẬP. CASE STUDY 1 (Ver 2.0): Tạo lập các class.

### III. Xây dựng tập thao tác trên đa thức

```
class DATHUC {
```

```
    private:
```

```
        float *P, *Q, *R; //Hệ số đa thức P, Q
```

```
        int n, m, k; //Bậc các đa thức
```

```
        int InitDathuc( float *, int );//Khởi tạo đa thức bậc n
```

```
        int Show( float *, int );//Hiển thị đa thức bậc n
```

```
        float *Value(float *, int n, float );//Tính giá trị đa thức
```

```
        float *Dao-ham(float *, int , int );//Tính đạo hàm đa thức
```

```
        float *Tong(float *, int, float *, int, float *, int );// Tổng hai đa thức
```

```
        float *Hieu(float *, int, float *, int, float *, int );// Hiệu hai đa thức
```

```
        float *Chia(float *, int, float *, int, float *, int );// Thương hai đa thức
```

```
    public:
```

```
        void Function(void); //Test các hàm thành viên
```

```
    };
```

## BÀI TẬP. CASE STUDY 1 (Ver 2.0): Tạo lập các class.

### IV. Xây dựng tập thao tác trên ma trận

```
class MATRAN {  
    private:  
        int n; //Bậc của ma trận vuông  
        float *A[n], *B[n], *C[n]; //Khai báo ma trận vuông A, B, C  
        int Init ( float *, int );//Khởi tạo ma trận vuông cấp n  
        void Display(float [], int );//Hiển thị ma trận vuông cấp n  
        void DoiHang(float [], int, int, int );//Đổi hàng i cho hàng j  
        void DoiCot(float [], int, int, int );//Đổi cột i cho cột j  
        void Tong (float [], float [], int n );//Tổng hai ma trận  
        void Hieu (float [], float [], int n );//Hiệu hai ma trận  
        void Tich (float [], float [], int n );//Tích hai ma trận  
        void Chuyenvi (float [], int n );// Ma trận chuyển vị  
        void NghichDao (float [], int n );// Ma trận nghịch đảo  
        float Dinhthuc (float [], int n );// Định thức ma trận vuông  
        void Grame (float [], float *, int n );// Nghiệm hệ phương trình  
    public:  
        void Function(void); //Test các hàm thành viên  
};
```

## BÀI TẬP. CASE STUDY 1 (Ver 2.0): Tạo lập các class.

### V. Xây dựng tập thao tác trên cấu trúc: Sinh Viên (SV)

```
class CAUTRUC {  
    private:  
        SV *A, *B, *C; //Con trỏ A, B, C có kiểu SV  
        void Init(SV *X, int n); //Khởi tạo sinh viên  
        void Display( SV *, int n); //Hiển thị danh sách sinh viên  
        void Suadoi( SV *, int, int ); //Sửa đổi thông tin sinh viên thứ n  
        void Sapxep( SV *, int ); // Sắp xếp theo thứ tự tăng dần  
        void Phan-Loai( SV *, int n); //Phân loại sinh viên  
        SV Search( SV *, int, SV x ); // Tìm kiếm sinh viên x  
        void GhiFile( SV*, int n); // Lưu trữ vào file  
        void LoadFile(SV *, int n); // Tải từ file  
        void Hop( SV *A, int, SV *B, int ); // Hợp của A và B  
        void Giao( SV *A, int, SV *B, int ); // Giao của A và B  
        void Hieu( SV *A, int, SV *B, int ); // Hiệu của A và B  
    public:  
        void Function(void); //Test các hàm thành viên  
};
```

## BÀI TẬP. CASE STUDY 1 (Ver 2.0): Tạo lập các class.

### VI. Xây dựng tập thao tác trên số phức:

```
class SOPHUC {  
    private:  
        sophuc a, b, c;  
        sophuc Init( sophuc x);// Khởi tạo số phức x  
        void Hienthi(sophuc x) ;// hiển thị số phức x  
        sophuc Tong(sophuc x, sophuc y);//tổng hai số phức  
        sophuc Hieu(sophuc x, sophuc y);//Hiệu hai số phức  
        sophuc Tich(sophuc x, sophuc y);//Tích hai số phức  
        sophuc Thuong(sophuc x, sophuc y);//thương hai  
        sophuc Luythua(sophuc x, int);//lũy thừa số phức  
    public:  
        void Function(void); //Test các hàm thành viên  
};
```

## BÀI TẬP. CASE STUDY 1 (Ver 2.0): Tạo lập các class.

### VII. Xây dựng tập thao tác trên số phức:

```
class FILES {  
    private:  
        sophuc *name;  
        void Init( char *); //Tạo lập file  
        void CopyFile (char *, char *); // Copy File  
        void Delete(char *); //Loại bỏ file  
        void Rename(char *); //Loại bỏ file  
        long Length( char *); // Xác định độ lớn file  
        void Taptu_File (char *); //Tìm tập từ & số lần xuất hiện từ trong File  
        void Hop( char *, char *); //Hợp hai tập từ & số lần xuất hiện từ  
        void Giao( char *, char *); //Giao hai tập từ & số lần xuất hiện từ  
        void Hieu( char *, char *); //Hợp hai tập từ & số lần xuất hiện từ  
        void Code( char *, char *); //Mã hóa file theo kỹ thuật chẵn lẻ  
        void DeCode( char *, char *); //Giải mã file theo kỹ thuật chẵn lẻ  
    public:  
        void Function(void); //Test các hàm thành viên  
};
```

### 2.2.3. Constructor và Destructor

**Constructor** là một thành viên đặc biệt của lớp nó tự động thực hiện khi nào ta tạo nên một đối tượng thuộc lớp. Đối với một lớp cụ thể, Constructor của lớp có một số tính chất sau:

- Mỗi lớp có duy nhất một hàm constructor dùng để xác thực đối tượng đã được tạo ra bởi class.
- Tên của hàm constructor trùng với tên của lớp (đây là qui định của ngôn ngữ). Constructor không có kiểu và không trả lại bất kỳ giá trị nào kể cả kiểu void.
- Hàm constructor được ngầm định là không có biến. Tuy vậy, ta có thể sử dụng biến cho constructor để khởi tạo các biến trong lớp tại thời điểm ta tạo ra đối tượng.

**Destructor:** là một thành viên đặc biệt của lớp nó tự động thực hiện khi nào đối tượng thuộc lớp đã ra khỏi phạm vi hoạt động của chương trình hoặc khi có toán tử delete hủy bỏ con trỏ đến đối tượng. Một Destructor có tính chất sau:

- Tên của destructor bắt đầu bằng ký tự ‘~’ và có tên với tên của lớp (trùng tên với constructor).
- Không sử dụng khai báo kiểu và tham biến cho Destructor.
- Destructor là một dạng giải phóng tài nguyên khi đối tượng không còn ảnh hưởng đến động thái hoạt động của phần còn lại của chương trình.

## Ví dụ về constructor, destructor của lớp:

```
#include <iostream>
using namespace std;
class Box {//Bieu dien lop Box
public:
    float length , breadth, height;
    Box (void ) { //Đây là constructor xác nhận đối tượng được tạo ra
        cout<<"Doi tuong da duoc tao ra" << endl;
    }
    ~Box (void) { //Đây là constructor xác nhận đối tượng bị hủy bỏ
        cout<<"Doi tuong da bi huy bo" << endl;
    }
    float Volume(void) {
        return(length*breadth*height);
    }
};

int main(void ) {
    Box Box1;//Box1 tro thanh doi tuong kieu Box
    Box1.length=4.0;Box1.breadth=5.0;Box1.height=6.0;
    cout<<"The tich:" << Box1.Volume() << endl;
    //Box Box2;//Box1 tro thanh doi tuong kieu Box
    system("PAUSE"); return 0;
}
```

## 2.2.4. Con trỏ This

Mỗi đối tượng có một con trỏ đặc biệt trỏ đến các thành viên của chính nó được gọi là con trỏ **this**. Vì vậy, con trỏ this được sử dụng bên trong hàm thành viên để thay thế cho chính đối tượng.

```
#include <iostream>
using namespace std;
class Box {
public:
    Box(double l=2.0, double b=2.0, double h=2.0){//Khởi tạo constructor
        cout << "Constructor duoc goi den" << endl;
        length = l;breadth = b;height = h;
    }
    double Volume(){ return length * breadth * height; }
    int compare(Box box) { return this->Volume() > box.Volume(); }
private:
    double length, breadth, height;
};
int main(void){ Box Box1(3.3, 1.2, 1.5); // Declare box1
    Box Box2(8.5, 6.0, 2.0); // Declare box2
    if(Box1.compare(Box2)) cout << "Box2 nhỏ hơn Box1" << endl;
    else cout << "Box2 lớn hơn hoặc bằng Box1" << endl;
    system("PAUSE");
}
```

## 2.2.5. Mảng các đối tượng – Con trỏ đến đối tượng

Mảng các đối tượng cũng được khai báo giống như mảng thông thường, các phép truy nhập phần tử của mảng cũng giống như mảng thông thường. Đối với lập trình cấu trúc, mảng là dãy có thứ tự các phần tử cùng chung một kiểu dữ liệu và được tổ chức liên tục nhau trong bộ nhớ. Đối với lập trình hướng đối tượng, mảng các đối tượng là một mảng mà mỗi phần tử của nó là một đối tượng được tổ chức liên tục nhau trong bộ nhớ. Điểm khác biệt duy nhất ở đây là biến của lập trình cấu trúc là dữ liệu, còn biến của lập trình hướng đối tượng bao gồm cả dữ liệu và hàm.

Ví dụ. Giả sử ta có định nghĩa lớp Box như dưới đây.

```
class Box {  
    private:  
        float length, breadth, height;  
    public:  
        Box( float x, float y, float z ) { //Tạo lập constructor  
            length = x; breadth = y; height = z;  
        }  
        float Volume(void ) {  
            return (length * breadth * height);  
        }  
};
```

Khi đó khai báo:

```
Box X[20]; //Khai báo mảng gồm 20 Box  
float V = X[10].Volume(); //thực hiện lấy thể tích của Box thứ 10
```

## Con trỏ đến đối tượng:

Con trỏ đến đối tượng cũng được khai báo giống như con trỏ thông thường, các phép truy nhập thành viên đối tượng cũng giống như phép truy nhập thành viên cấu trúc. Con trỏ đến đối tượng cũng được cấp phát miền nhớ bằng new, giải phóng miền nhớ bằng delete.

Ví dụ. Giả sử ta có định nghĩa lớp Box như dưới đây.

```
class Box {  
    private:  
        float length;  
        float breadth;  
        float height;  
    public:  
        Box( float x, float y, float z ) { //Tạo lập constructor  
            length = x; breadth = y; height = z;  
        }  
        float Volume(void) { return (length * breadth * height); }  
};
```

Khi đó khai báo:

```
Box Box1, Box2; //Khai báo Box1, Box2 kiểu Box  
Box *ptrBox; // Khai báo một con trỏ Box  
ptrBox = &Box1; // ptrBox trỏ đến Box1;  
float V = ptrBox -> Volume(); // Lấy thể tích Box1.  
ptrBox = &Box2; // ptrBox trỏ đến Box2;  
float V = ptrBox -> Volume(); // Lấy thể tích Box2.
```

## Ví dụ về mảng các đối tượng:

```
#include <iostream>
using namespace std;
class Box{
public:
    Box(double l=2.0, double b=2.0, double h=2.0){//Constructor
        cout << "Thuc hien Constructor" << endl;
        length = l;breadth = b;height = h;
    }
    double Volume() { return length * breadth * height; }
private:
    double length; // Length of a box
    double breadth; // Breadth of a box
    double height; // Height of a box
};
int main(void){
    Box Box1[20];
    cout << "Volume of Box1[10] =" << Box1[10].Volume() << endl;
    system("PAUSE");
    return 0;
}
```

## Ví dụ về con trỏ đến đối tượng:

```
#include <iostream>
using namespace std;
class Box{
public:
    Box(double l=2.0, double b=2.0, double h=2.0){//Constructor
        cout << "Thuc hien Constructor" << endl;
        length = l;breadth = b;height = h;
    }
    double Volume() { return length * breadth * height; }
private:
    double length, breadth, height;
};
int main(void){
    Box Box1(3.3, 1.2, 1.5); // Declare box1
    Box Box2(8.5, 6.0, 2.0); // Declare box2
    Box *ptrBox ; // Declare pointer to a class.
    ptrBox = &Box1; // Tro den Box1
    cout << "Volume of Box1: " << ptrBox[10].Volume() << endl;
    ptrBox = &Box2;//Tro den Box2
    cout << "Volume of Box2: " << ptrBox->Volume() << endl;
    system("PAUSE");
}
```

## 2.2.6. CASE STUDY 1 (Ver 2.0) : Tạo lập các class.

### I. Xây dựng tập thao tác với số nguyên

- Biểu diễn N ở hệ cơ số b.
- Phân tích N thành tích các thừa số nguyên tố.
- Duyệt các số nguyên tố có N chữ số.
- Duyệt các cặp số hữu nghị a, b nhỏ hơn N.
- Duyệt các số hoàn hảo nhỏ hơn N.
- Duyệt các cặp số P, 4P + 1 là nguyên tố nhỏ hơn N.
- Duyệt các số nguyên tố nhỏ hơn N có tổng các chữ số là S.
- Duyệt các số thuận nghịch có N chữ số có tổng các chữ số là S.
- Duyệt các số thuận nghịch có N chữ số sao cho biểu diễn số đó ở hệ cơ số b cũng là số thuận nghịch.
- Xây dựng phép cộng, trừ, nhân chia giữa hai số lớn (512 chữ số).
- Tìm số nguyên tố lớn (512 chữ số).
- Đưa ra 5 thuật toán tìm số nguyên tố khác nhau, so sánh độ phức tạp tính toán của các thuật toán.

## II. Xây dựng tập thao tác với xâu ký tự:

- Tìm  $X = \{x \in S_1 \text{ hoặc } x \in S_2\}$ .
- Tìm  $X = \{x \in S_1 \text{ và } x \in S_2\}$ .
- Tìm  $X = \{x \in S_1 \text{ và } x \text{ không thuộc } S_2\}$ .
- Tìm tập ký tự và số lần xuất hiện mỗi ký tự trong cả  $S_1, S_2$  (Không kể ký tự trùng).
- Tìm tập ký tự và số lần xuất hiện mỗi ký tự thuộc cả  $S_1$  và  $S_2$  (Không kể ký tự trùng).
- Tìm tập ký tự và số lần xuất hiện mỗi ký tự thuộc  $S_1$  nhưng không thuộc  $S_2$  (Không kể ký tự trùng).
- Mã hóa  $X$  bằng kỹ thuật chẵn lẻ.
- Giải mã  $X$  bằng kỹ thuật chẵn lẻ.
- Tìm tập từ và số lần xuất hiện mỗi từ trong  $S_1$  hoặc  $S_2$ .
- Tìm tập từ và số lần xuất hiện mỗi từ trong  $S_1$  và  $S_2$ .
- Tìm tập từ và số lần xuất hiện mỗi từ trong  $S_1$  nhưng không xuất hiện trong  $S_2$ .

### **III. Xây dựng tập thao tác với đa thức:**

- Tạo lập hai đa thức  $P_n(X)$ ,  $Q_m(X)$ .
- Tìm  $P_n(X_0)$ ,  $Q_m(X_0)$ .
- Tìm đạo hàm cấp  $L$  của  $P_n(x)$ ,  $Q_m(x)$ .
- Tìm  $R = P + Q$ .
- Tìm  $R = P - Q$ .
- Tìm  $R = P^*Q$ .
- Tìm  $R = P/Q$  và đa thức dư.
- Xây dựng các thao tác cộng, trừ nhân, chia hai số nguyên bằng đa thức.

### **IV. Xây dựng tập thao tác trên ma trận**

- Tạo lập ma trận.
- Nhân hai ma trận
- Tìm phần tử lớn nhất của ma trận.
- Tìm hạng của ma trận.
- Tìm các vector riêng & giá trị riêng.
- Tìm chuyển vị của ma trận.
- Tìm định thức của ma trận.
- Tìm nghịch đảo của ma trận.
- Giải hệ phương trình tuyến tính thuận nhất  $AX=B$ .

## V. Xây dựng tập thao tác với đa thức bằng cấu trúc

- Tạo lập hai đa thức  $P_n(X)$ ,  $Q_m(X)$ .
- Tìm  $P_n(X_0)$ ,  $Q_m(X_0)$ .
- Tìm đạo hàm cấp  $L$  của  $P_n(x)$ ,  $Q_m(x)$ .
- Tìm  $R = P + Q$ .
- Tìm  $R = P - Q$ .
- Tìm  $R = P^*Q$ .
- Tìm  $R = P/Q$  và đa thức dư.

## VI. Xây dựng tập thao tác trên số phức bằng cấu trúc

- Tạo lập hai số phức.
- Cộng hai số phức.
- Nhân hai số phức.
- Chia hai số phức.
- Lũy thừa số phức.
- Căn bậc hai của số phức.

## VII. Xây dựng hệ quản lý sách bao gồm những thao tác sau

- Nhập sách.
- Hiển thị thông tin về sách
- Sắp xếp theo chủ đề sách.
- Kiểm theo chủ đề sách.

## VIII. Xây dựng tập thao tác trên FILE

- Đếm số dòng trong file.
- Đếm số từ trong file.
- Tìm tập từ và số lần xuất hiện mỗi từ trong Data1.in.
- Tìm tập từ và số lần xuất hiện từ trong Data1.in hoặc data2.in.
- Tìm tập từ và số lần xuất hiện từ trong Data1.in và data2.in.
- Tìm tập từ và số lần xuất hiện từ trong Data1.in nhưng không xuất hiện trong data2.in.
- Mã hóa file bằng kỹ thuật chẵn lẻ.
- Giải mã file bằng kỹ thuật chẵn lẻ.
- Đổi tên file.
- Loại bỏ file . . .

## **VIII. Xây dựng tập thao tác với tập hợp**

- Duyệt các xâu nhị phân có độ dài n.
- Duyệt các tập con K phần tử của 1, 2,..,n.
- Duyệt các hoán vị của 1, 2, ..,n.
- Duyệt các cách chi số N thành tổng các số tự nhiên nhỏ hơn N.
- Duyệt các xâu nhị phân độ dài N có đúng 1 dãy K số 0 và 1 dãy m M số 1 liên tiếp.
- Duyệt các dãy con K phần tử tăng dần tự nhiên của dãy số An.
- Duyệt dãy số gồm N phần tử có tổng số K phần tử bất kỳ là nguyên tố
- Giải bài toán N quân hậu.
- Giải bài toán mã đi tuần.
- Giải bài toán cái túi.
- Giải bài toán người du lịch.
- Giải bài toán cho thuê máy.

## 2.3. Kế thừa (Inheritance)

**Nguyên lý kế thừa:** một trong những nguyên lý quan trọng của lập trình hướng đối tượng đó là kế thừa. Nguyên lý kế thừa cho phép ta định nghĩa một class bên trong một lớp khác. Điều này khiến cho ta thuận tiện hơn trong quá trình phát triển, duy trì ứng dụng, ta sử dụng lại code có trước, các hàm và giảm chi phí thời gian cài đặt.

**Lớp cơ sở (based class) và lớp dẫn xuất (derived class) :** khi tạo nên một class, thay thế bằng việc định nghĩa lại toàn bộ các thành viên của class người lập trình chỉ cần định nghĩa một class mới kế thừa các thành viên của một hoặc nhiều class đã tạo ra trước đó. Các class được tạo ra trước đó được gọi là class cơ sở (**base class**), class mới được tạo ra từ các lớp cơ sở được gọi là class dẫn xuất (**derived class**). Hình thức tạo nên một lớp mới từ các lớp cơ sở cho trước được gọi là hình thức kế thừa.

**Kế thừa bội (Multi-inheritance):** một class có thể được dẫn xuất từ nhiều class cơ sở. Điều này có nghĩa nó kế thừa dữ liệu và các hàm từ nhiều lớp cơ sở khác nhau. Để định nghĩa một class dẫn xuất từ các class cơ sở ta chỉ cần thực hiện theo cú pháp như sau:

*class derived\_class : access-specifier base-class;*

Trong đó:

*derived\_class* : tên của class dẫn xuất;

*access-specifier* : hình thức kế thừa (public, private, protected);

*base-class* : danh sách các class cơ sở;

**Điều khiển quyền truy cập:** một class dẫn xuất được phép truy cập đến các thành viên không phải là thành viên **private** của class cơ sở. Ngược lại, các hàm của lớp cơ sở không được phép truy cập đến các thành viên private của lớp dẫn xuất. Bảng dưới đây tóm tắt lại quyền truy cập của các lớp cơ sở và lớp dẫn xuất.

Quyền truy cập	Thành viên Public	Thành viên Protected	Thành viên Private
Cùng trong một lớp	Yes	Yes	Yes
Lớp dẫn xuất	Yes	Yes	No
Bên ngoài lớp	Yes	No	No

**Một số kế thừa mặc định của lớp dẫn xuất:** một class dẫn xuất ngầm định được kế thừa các thành viên (*các thành viên bắt buộc phải định nghĩa là public*):

- Constructor , Destructor và bản sao của Constructor từ lớp cơ sở.
- Các phép toán từ lớp cơ sở.
- Các hàm bạn (friend function) từ lớp cơ sở.

**Các hình thức kế thừa:** một class dẫn xuất có thể kế thừa các thành viên *public*, *private* hoặc *protected* từ một hoặc nhiều class cơ sở. Trong đó, việc kế thừa các thành viên *private*, *protected* thường ít được sử dụng. Thông dụng nhất vẫn là kế thừa thành phần *public*. Nguyên tắc kế thừa được thực hiện như sau:

- **Public inheritance:** khi định nghĩa một class dẫn xuất kế thừa kiểu *public* từ class cơ sở thì thành viên *public* của class cơ sở trở thành thành viên *public* của class dẫn xuất, thành viên *protected* của class cơ sở trở thành thành viên *protected* của class dẫn xuất. Thành viên *private* của class cơ sở không là thành viên của lớp class dẫn xuất nhưng vẫn bị truy cập gián tiếp thông qua các lời gọi hàm từ các thành viên *public* và *protected* của class cơ sở.
- **Protected inheritance:** khi định nghĩa một class dẫn xuất kế thừa kiểu *protected* của class cơ sở thì thành viên *public* và *protected* của class cơ sở trở thành thành viên *protected* của class dẫn xuất.
- **Private inheritance:** khi định nghĩa một class dẫn xuất kế thừa kiểu *private* của class cơ sở thì thành viên *public* và *protected* của class cơ sở trở thành thành viên *private* của class dẫn xuất.
- **Multi-inheritance:** một lớp dẫn xuất có thể được kế thừa các thành viên khác nhau từ nhiều lớp cơ sở. Khi đó ta chỉ cần định nghĩa lớp dẫn xuất theo nguyên tắc sau:

class derived-class: access baseA, access baseB,...

Ví dụ lớp X được kế thừa thành phần *public* của lớp A, kế thừa thành phần *protected* của lớp B, kế thừa thành phần *private* của lớp C ta định nghĩa như sau:

class X : public A, protected B, private C;

## Ví dụ về public inheritance:

```
#include <iostream>
using namespace std;
class Box {
    private:
        float length, breadth, height;
    protected:
        void SetLength(float x) { length = x; }
        void SetBreadth(float y) { breadth = y; }
        void SetHeight(float z) { height = z; }
    public:
        Box (void){length=4.0;breadth=5.0;height=6.0;}
        float Volume(void) { return(length * breadth* height); }
};
class Box1: public Box {
    private:
        float l, b;
        void SetL( float x) { l=x; }
        void SetB( float y) { b=y; }
    public:
        float Volume1(void) { float x = 4, y = 5, z = 3, h = z;
            float V, V1, V2; SetL(1); SetB(2);
            SetLength(x); SetBreadth(y); SetHeight(z);
            V1 = Volume(); V2 = (l*b*h);
            if (V1>V2) V = V2 + (V1-V2)/2;
            else V = V1 + (V2-V1)/2;
            return(V);
        }
    };
int main(void) { Box1 X;
    cout<<"The tich:"<<X.Volume()<<endl;
    cout<<"The tich:"<<X.Volume1()<<endl;
    system("PAUSE");return 0;
}
```

## Ví dụ về protected inheritance:

```
#include <iostream>
using namespace std;
class Box {
    private:
        float length, breadth, height;
    protected:
        void SetLength(float x) { length = x; }
        void SetBreadth(float y) { breadth = y; }
        void SetHeight(float z) { height = z; }
    public:
        Box (void){length=4.0;breadth=5.0;height=6.0;}
        float Volume(void) { return( length * breadth* height); }
};
class Box1: protected Box {
    private:
        float l, b;
        void SetL( float x) { l=x; }
        void SetB( float y) { b=y; }
    public:
        float Volume1(void) { float x = 4, y = 5, z =3, h = z;
            float V, V1, V2; SetL(1); SetB(2);
            SetLength(x); SetBreadth(y); SetHeight(z);
            V1 = Volume();V2 = (l*b*h);
            if (V1>V2) V = V2 + (V1-V2)/2;
            else V = V1 + (V2-V1)/2;
            return(V);
        }
    };
int main(void) { Box1 X;
    //cout<<"The tich:"<<X.Volume()<<endl; //Error
    cout<<"The tich:"<<X.Volume1()<<endl;
    system("PAUSE");return 0;
}
```

## Ví dụ về private inheritance:

```
#include <iostream>
using namespace std;
class Box {
protected :
    float length, breadth, height;
    void SetLength(float x) { length = x; }
    void SetBreadth(float y) { breadth = y; }
    void SetHeight(float z) { height = z; }
public:
    Box (void){length=4.0; breadth=5.0; height=6.0;}
    float Volume(void) { return(length * breadth* height);}
};

class Box1: private Box { //Ke thua protected
private:
    float l, b;
    void SetL( float x) { l =x; }
    void SetB( float y) { b =y; }
public:
    float Volume1(void) {
        float x = 4, y = 5, z =3, h;
        float V, V1, V2; SetL(1); SetB(2);
        SetLength(x); SetBreadth(y); SetHeight(z); h = height;
        V1 = Volume(); V2 = (l*b*h);
        if (V1>V2) V = V2 + (V1-V2)/2;
        else V = V1 + (V2-V1)/2;
        return(V);
    }
};

int main(void) {
    Box1 X;
    //cout<<"The tich:"<<X.Volume1()<<endl; //Error
    cout<<"The tich:"<<X.Volume1()<<endl;
    system("PAUSE");
}
```

## Ví dụ về multi inheritance:

```
#include <iostream>
using namespace std;
class A {
protected:
    float length, breadth, height;
    void SetLength(float x) { length = x; }
    void SetBreadth(float y) { breadth = y; }
    void SetHeight(float z) { height = z; }
public:
    A(void) {length=3;breadth=4;height=5;}
};

class B {
protected:
    float l, b;
    void SetL( float x) { l=x; }
    void SetB( float y) { b =y; }
public:
    B(void){l = 1; b = 1; }
};

class C:private A, public B {
public:
    float Volume(void ) { return( length * breadth* height); }
    float Volume1(void) {
        float V = (((length*breadth)+(l*b))*height)/2;
        return(V);
    }
};

int main(void) {
    C X;
    cout<<"The tich:"<<X.Volume()<<endl;
    cout<<"The tich:"<<X.Volume1()<<endl;
    system("PAUSE");
}
```

## 2.4. Xử lý quá tải (Overloading)

**Định nghĩa:** Lập trình hướng đối tượng cho phép ta định nghĩa nhiều hơn một hàm hoặc toán tử cùng tên trong cùng một phạm vi. Cơ chế thực hiện như vậy được gọi là cơ chế xử lý hàm quá tải (function overloading) hoặc toán tử quá tải (operator overloading).

### 2.4.1. Hàm quá tải

- **Function Overloading:** OOP cho phép ta định nghĩa nhiều hàm với cùng một tên thực hiện trong cùng một phạm vi. Các hàm chỉ khác nhau về kiểu, danh sách tham đối của hàm. Function Overloading chỉ không được phép khai báo đối với các hàm chỉ khác nhau kiểu giá trị trả về của hàm.
- **Thực thi đối với Function Overloading:** khi có một lời gọi hàm, chương trình dịch (compiler) xác định hàm phù hợp nhất để thực hiện bằng cách so sánh đối của hàm, kiểu giá trị trả về của hàm. Quá trình chọn hàm quá tải phù hợp nhất để thực hiện còn được gọi là quá trình xử lý quá tải (Overload Resolution).
- **Thiết kế hàm quá tải:** hàm quá tải gồm nhiều hàm có tên giống nhau được định nghĩa trong cùng một phạm vi. Mỗi hàm có thể khác nhau về danh sách đối của hàm, kiểu giá trị trả về của hàm. Không được phép xây dựng hai hàm quá tải giống nhau về đối nhưng chỉ khác nhau về kiểu giá trị của hàm.

## Ví dụ. Function Overloading.

```
#include <iostream>
using namespace std;
class printData { //Định nghĩa lớp printData
public:
    void print(int i) { //Hàm thứ nhất in ra một số nguyên
        cout << "Số nguyên: " << i << endl;
    }
    void print(double f) { //Hàm thứ hai in ra một số thực độ chính xác kép
        cout << "Số thực: " << f << endl;
    }
    void print(char* s) { //Hàm thứ ba in ra một xâu ký tự
        cout << "Xâu ký tự: " << s << endl;
    }
};
int main(void){
    printData pd; //pd là đối tượng có kiểu printData
    pd.print(5); //Gọi đến hàm in ra số nguyên
    pd.print(500.263); //Gọi đến hàm in ra số thực
    pd.print("Hello C++"); //Gọi đến hàm in ra xâu ký tự
    system("PAUSE");
}
```

**BÀI TẬP (Function Overloading).** Dãy số nguyên gồm n phần tử, dãy số thực gồm n phần tử, dãy số thực có độ chính xác đơn gồm n phần tử, dãy số thực có độ chính xác kép gồm n phần tử. Hãy xây dựng một lớp hàm quá tải thực hiện được các nhiệm vụ dưới đây:

- Có thể khởi tạo dãy các số nguyên int gồm n phần tử.
- Có thể khởi tạo dãy các số nguyên long gồm n phần tử.
- Có thể khởi tạo dãy các số thực float gồm n phần tử.
- Có thể khởi tạo dãy các số thực double gồm n phần tử.
- Có thể tìm được số nguyên int lớn nhất trong dãy số.
- Có thể tìm được số nguyên long lớn nhất trong dãy số.
- Có thể tìm được số thực float lớn nhất trong dãy số.
- Có thể tìm được số thực double lớn nhất trong dãy số.
- Có thể tìm được số nguyên int nhỏ nhất trong dãy số.
- .....
- Có thể tìm sắp xếp dãy số nguyên int theo thứ tự tăng dần.
- .....
- Có thể tìm được vị trí số nguyên int có giá trị x trong dãy số.
- .....
- Có thể tìm được vị trí số thực double có giá trị x trong dãy số.

```
class OverLoad_Function {  
    private:  
        int *A, n, m, p, q; long *B; float *C; double *D;  
    protected:  
        void *Init( int *X, int n); // Khởi tạo mảng số nguyên int gồm n phần tử  
        void *Init( float *X, int n); // Khởi tạo mảng số nguyên long gồm n phần tử  
        void *Init( long *X, int n); // Khởi tạo mảng số thực float gồm n phần tử  
        void *Init( double *X, int n); // Khởi tạo mảng số thực double gồm n phần tử  
        int MAX ( int *X, int n); //Tìm vị trí phần tử lớn nhất trong dãy số nguyên int  
        int MAX ( long *X, int n); //Tìm vị trí phần tử lớn nhất trong dãy số nguyên long  
        int MAX ( float *X, int n); //Tìm vị trí phần tử lớn nhất trong dãy số thực float  
        int MAX ( double *X, int n); //Tìm vị trí phần tử lớn nhất trong dãy số thực double  
        int MIN ( int *X , int n); //Tìm vị trí phần tử nhỏ nhất trong dãy số nguyên int  
        int MIN ( long * x, int n); //Tìm vị trí phần tử nhỏ nhất trong dãy số nguyên long  
        int MIN ( float *X, int n); //Tìm vị trí phần tử nhỏ nhất trong dãy số thực float  
        int MIN ( double *X, int n); //Tìm vị trí phần tử nhỏ nhất trong dãy số thực double  
        void Sort( int *X, int n); //Sắp xếp dãy số nguyên int theo thứ tự tăng dần  
        void Sort( long *X, int n); //Sắp xếp dãy số nguyên long theo thứ tự tăng dần  
        void Sort( float *X, int n); //Sắp xếp dãy số thực float theo thứ tự tăng dần  
        void Sort( double *X, int n); //Sắp xếp dãy số thực float theo thứ tự tăng dần  
        int Search( int * X, int n, int k); //Tìm vị trí số nguyên k trong dãy số nguyên X  
        int Search( long *X, int n, long k); //Tìm vị trí số nguyên k trong dãy số nguyên X  
        int Search( float *X, int n, float k); //Tìm vị trí số thực k trong dãy số thực X  
        int Search( double *X, int n, double k); //Tìm vị trí số thực k trong dãy số thực X  
    public:  
        void Test (void){ }; // Kiểm tra lại các hàm quá tải.  
};
```

**BÀI TẬP (Function Overloading):** Cho xâu ký tự S, dãy số nguyên A[] gồm N phần tử, dãy số thực X[] gồm N phần tử, dãy số phức C[N] gồm N phần tử, ma trận vuông cấp N, file lưu trữ các số nguyên, file lưu trữ các số thực, file văn bản bất kỳ. Hãy xây dựng một lớp hàm quá tải thực hiện được các nhiệm vụ dưới đây:

- Có thể tìm ký tự c đầu tiên xuất hiện trong xâu S.
- Có thể tìm số x đầu tiên xuất hiện trong dãy số A[].
- Có thể tìm số phức x đầu tiên xuất hiện trong dãy số phức X[].
- Có thể tìm chỉ số hàng, chỉ số cột của phần tử có giá trị x của ma trận.
- Có thể tìm số nguyên x đầu tiên xuất hiện trong file.
- Có thể tìm số thực x đầu tiên xuất hiện trong file.
- Có thể đưa ra từ s đầu tiên xuất hiện trong file.
- Có thể tìm tập ký tự và số lần xuất hiện mỗi ký tự trong xâu S.
- Có thể tìm tập các số nguyên và số lần xuất hiện mỗi số trong dãy số.
- Có thể tìm tập các số thực và số lần xuất hiện mỗi số trong dãy số.
- Có thể tìm tập các số nguyên và số lần xuất hiện mỗi số trong file.
- Có thể tìm tập các số thực và số lần xuất hiện mỗi số trong file.
- Có thể tìm tập các từ và số lần xuất hiện mỗi từ trong file. . .

**BÀI TẬP (Function Overloading):** File các file số nguyên, file văn bản. Hãy xây dựng một lớp hàm quá tải thực hiện được các nhiệm vụ dưới đây:

- Có thể tìm tập các số nguyên và số lần xuất hiện mỗi số trong file.
- Có thể tìm tập các từ và số lần xuất hiện mỗi từ trong file.
- Có thể tìm tập các số và số lần xuất hiện mỗi số trong cả hai file.
- Có thể tìm tập các từ và số lần xuất hiện mỗi từ trong cả hai file.
- Có thể tìm tập các số và số lần xuất hiện mỗi số trong file1 và file2 .
- Có thể tìm tập các từ và số lần xuất hiện mỗi từ trong file1 và file2 .
- Có thể tìm tập các số và số lần xuất hiện mỗi số trong file1 nhưng không xuất hiện trong file2
- Có thể tìm tập các từ và số lần xuất hiện mỗi từ trong file1 nhưng không xuất hiện trong file2.
- Có thể tìm sắp xếp tập các số và số lần xuất hiện mỗi số theo thứ tự tăng dần.
- Có thể tìm sắp xếp tập các từ xuất hiện mỗi số theo thứ tự tăng dần.
- Có thể tìm kiếm một số trên tập các số và số lần xuất hiện mỗi số.
- Có thể tìm kiếm một từ trên tập các từ và số lần xuất hiện mỗi từ....

## 2.4.2. Phép toán quá tải (Operator Overloading)

**Định nghĩa:** Lập trình hướng đối tượng cho phép ta định nghĩa nhiều hơn một toán tử cùng tên trong cùng một phạm vi. Cơ chế thực hiện như vậy được gọi là cơ chế xử lý phép toán quá tải (operator overloading).

**Khai báo Operator Overloading:** là một tên đặc biệt đi sau từ khóa “operator” bằng một ký hiệu (ví dụ ký hiệu: +, -, \*, /..). Giống như hàm, kiểu toán tử cũng có một giá trị trả về và danh sách các tham biến. Hầu hết các phép toán được định nghĩa thông qua việc thực hiện sự tổ hợp các hàm thành viên.

### Một số phép toán có thể xây dựng operator overloading

+	-	*	/	%	$\wedge$
&		~	!	,	=
<	>	$\leq$	$\geq$	$\oplus$	$\ominus$
$\ll$	$\gg$	$\equiv$	$\mid=$	$\&\&$	$\parallel$
$\oplus=$	$\ominus=$	$\mid=$	$\%=$	$\wedge=$	$\&=$
$\mid=$	$\equiv=$	$\ll\equiv$	$\gg\equiv$	$\parallel$	$\emptyset$
$\rightarrow$	$\rightarrow^*$	new	new []	delete	delete []

## Ví dụ. Operator Overloading: Các phép toán số học.

```
#include <iostream>
using namespace std;
class Complex {
private:    float real, im;
public:
    void Init(float x, float y ) { cout<<"Phan thuc:"; cin>>x;real = x;
        cout<<"Phan ao:"; cin>>y; im = y;
    }
    void PrintComplex( void) { cout<<"Phan thuc:"<<real<<" Phan ao:"<<im<<endl;
    }
    Complex operator + (Complex &c){ Complex temp;
        temp.real = this->real + c.real;  temp.im = this->im + c.im;
        return(temp);
    }
};
int main(void) {
    Complex X1, X2, X3, X; float a, b;
    X1.Init(a, b); X1.PrintComplex(); X2.Init(a, b); X2.PrintComplex();
    X3.Init(a, b); X3.PrintComplex(); X = X1 + X2; X.PrintComplex();
    X = X1+X2+X3; X.PrintComplex();
    system("PAUSE");
    return 0;
}
```

## Ví dụ. Operator Overloading: Các phép toán so sánh.

```
#include <iostream>
using namespace std;
class Complex {
private: float real, im;
public:
    void Init(float x, float y ) {
        cout<<"Phan thuc:"; cin>>x;real = x; cout<<"Phan ao:"; cin>>y; im = y;
    }
    void PrintCom( void) {
        cout<<"Phan thuc:"<<real<<" Phan ao:"<<im<<endl;
    }
    bool operator != (Complex &c){
        if ( (this->real != c.real)|| (this->im != c.im))
            return(true);
        return(false);
    }
};
int main(void) {
    Complex X1, X2, X3, X; float a, b;
    X1.Init(a, b);X1.PrintCom(); X2.Init(a, b);X2.PrintCom();
    cout<<"X1!=X2:"<<(X1!=X2)<<endl;
    system("PAUSE"); return 0;
}
```

## Ví dụ. Operator Overloading: Các phép toán trên phân số.

```
#include <iostream>
using namespace std;
class Complex {
private: float real, im;
public:
    void Init(float x, float y ) {
        cout<<"Phan thuc:"; cin>>x;real = x; cout<<"Phan ao:"; cin>>y; im = y;
    }
    void PrintCom( void) {
        cout<<"Phan thuc:"<<real<<" Phan ao:"<<im<<endl;
    }
    bool operator != (Complex &c){
        if ( (this->real != c.real)|| (this->im != c.im))
            return(true);
        return(false);
    }
};
int main(void) {
    Complex X1, X2, X3, X; float a, b;
    X1.Init(a, b);X1.PrintCom(); X2.Init(a, b);X2.PrintCom();
    cout<<"X1!=X2:"<<(X1!=X2)<<endl;
    system("PAUSE"); return 0;
}
```

## BÀI TẬP (Operator Overloading):

1) Xây dựng các phép toán cho tập các thao tác trên số phức

- Phép cộng (+) giữa hai số phức.
- Phép trừ (-) giữa hai số phức.
- Phép nhân (\*) giữa hai số phức.
- Phép chia (/) giữa hai số phức.
- Phép so sánh khác nhau ( $\neq$ ) giữa hai số phức.
- Phép so sánh đúng bằng ( $=$ ) giữa hai số phức . . .

2) Xây dựng thao tác và phép toán cho tập các thao tác trên vector

- Phép cộng (+) giữa hai vector.
- Phép trừ (-) giữa hai vector.
- Phép nhân (\*) giữa hai vector (tích vô hướng).
- Phép so sánh khác nhau ( $\neq$ ) giữa hai vector.
- Phép so sánh đúng bằng ( $=$ ) giữa hai vector.
- Tính khoảng cách giữa hai vector
- Tính Cosin giữa hai vector ...

## BÀI TẬP (Operator Overloading):

3) Xây dựng các phép toán quá tải cho tập các xâu ký tự

- Phép cộng (+) giữa hai xâu ký tự.
- Phép so sánh (==) giữa hai xâu ký tự.
- Phép so sánh (!=) giữa hai xâu ký tự.
- Phép so sánh (>) giữa hai xâu ký tự.
- Phép so sánh (>=) giữa hai xâu ký tự.
- Phép so sánh (<) giữa hai xâu ký tự.
- Phép so sánh (<=) giữa hai xâu ký tự.
- Phép cộng độ dài (+) giữa hai xâu ký tự.
- Phép so sánh độ dài(==) giữa hai xâu ký tự.
- Phép so sánh độ dài(!=) giữa hai xâu ký tự.
- Phép so sánh độ dài(>) giữa hai xâu ký tự.
- Phép so sánh độ dài(>=) giữa hai xâu ký tự.
- Phép so sánh độ dài(<) giữa hai xâu ký tự.
- Phép so sánh độ dài(<=) giữa hai xâu ký tự.

## BÀI TẬP (Operator Overloading):

3) Xây dựng thao tác và phép toán trên đa thức.

- Phép cộng (+) hai đa thức.
- Phép trừ (-) hai đa thức.
- Phép nhân (\*) hai đa thức.
- Phép chia hai đa thức.
- Phép so sánh khác nhau (!=) giữa hai đa thức.
- Phép so sánh đúng bằng (==) giữa hai đa thức.

## BÀI TẬP (Operator Overloading):

4) Xây dựng thao tác và phép toán trên ma trận.

- Phép cộng (+) hai đa ma trận.
- Phép trừ (-) hai đa ma trận.
- Phép nhân (\*) hai ma trận.
- Phép so sánh khác nhau ( $\neq$ ) giữa hai ma trận.
- Phép so sánh đúng bằng ( $=$ ) giữa hai ma trận...

5) Xây dựng thao tác và phép toán trên tập từ của file.

- Phép hợp (+) hai tập từ của file: là tập các từ hoặc thuộc file thứ nhất hoặc thuộc file thứ 2.
- Phép trừ (-) hai tập từ của file : là tập các từ thuộc file thứ nhất nhưng không thuộc file thứ 2.
- Phép giao (.) hai tập từ của file: là tập các từ thuộc file thứ nhất và thuộc file thứ 2.
- Phép so sánh khác nhau ( $\neq$ ) giữa hai tập từ.
- Phép so sánh đúng bằng ( $=$ ) giữa hai ma trận...

## 2.6. Xử lý đa hình (Polymorphism)

**Polymorphism:** Từ khóa “*Polymorphism*” có nghĩa khác nhau. Đa hình thái xảy ra khi có một quan hệ kế thừa theo phân cấp tồn tại trong các lớp (ví dụ: lớp cha, lớp con,...). Trong tình huống này, một lời gọi hàm thành viên là nguyên nhân để hàm thành viên khác thực hiện. Hàm được thực hiện phụ thuộc vào kiểu của đối tượng liên đới đến hàm.

**Virtual Functions** (hàm ảo): là một hàm được mô tả trong các lớp cơ sở được khai báo bằng từ khóa “**virtual**”. Virtual Function giúp ta chọn hàm nào sẽ được gọi đến ở mọi thời điểm trong chương trình dựa vào kiểu của đối tượng được gọi đến.

## Ví dụ. Polymorphism.

```
#include <iostream>
using namespace std;
class Shape { //Khai bao lop da giac
protected: int width, height;
public: Shape( int a=0, int b=0){ width = a; height = b; }
        virtual int area(){cout << "Khong biet tinh:" << endl;return 0;}
};
class Rectangle: public Shape{ //Khai bao lop hinh chu nhat
public: Rectangle( int a=0, int b=0):Shape(a, b) {}
        int area (){ cout << "Dien tich hinh chu nhat:" << endl; return (width*height);}
};
class Triangle: public Shape{
private: int x, y;
public: Triangle( int a=0, int b=0):Shape(a, b) {}
        int area (){ cout << "Dien tich tam giac:" << endl;return (width * height / 2); }
};
int main( ){
    Shape *shape; Rectangle rec(10,7); Triangle tri(10,5);
    shape = &rec; shape->area(); // store the address of Rectangle
    shape = &tri; shape->area(); // call triangle area
    system("PAUSE"); return 0;
}
```

## 2.7. Trùu tượng dữ liệu (abstraction data )

**Data Abstraction:** là kỹ thuật lập trình, kỹ thuật thiết kế chương trình nhằm cung cấp ra thế giới bên ngoài những thông tin thực sự cần thiết, ẩn dấu đi những thông tin cơ bản riêng biệt. Kỹ thuật này dựa vào giao diện (interface) và cài đặt cụ thể (implementation) của lớp. Ví dụ, ta thường dùng chỉ thị “cout” của C++ chỉ là một giao diện của lớp iostream ra thế giới bên ngoài. Nhưng thực sự chỉ thị cout được cài đặt thế nào lại là điều che dấu đối với người dùng.

**Truy nhập nhãn tuân theo sự trùu tượng :** Ta sử dụng các nhãn để định nghĩa giao tiếp trùu tượng đến lớp. Mỗi lớp có thể có một hoặc nhiều nhãn: public, private, protected. Trong đó:

- Các thành viên được định nghĩa với nhãn public được phép truy nhập trong tất cả các phần khác nhau củ chương trình. Điều này có nghĩa, các thành viên này được phép giao tiếp với thế giới bên ngoài lớp.
- Các thành viên được định nghĩa với nhãn private không được phép truy nhập từ bên ngoài lớp. Điều này có nghĩa, những gì ta cần ẩn dấu sẽ được định nghĩa trong thành phần private của lớp.

**Quan điểm thiết kế lớp:** thiết kế giao tiếp và cài đặt cụ thể nên tiến hành độc lập nhau. Khi có sự thay đổi về cài đặt không ảnh hưởng đến giao tiếp của lớp trong khi xây dựng ứng dụng. Điều này có nghĩa xác định rõ thành phần nào là public, thành phần nào là private và thành phần nào là protected.

## Ví dụ về trừu tượng dữ liệu.

- I. Xây dựng tập thao tác với số nguyên
  - Biểu diễn N ở hệ cơ số b.
  - Phân tích N thành tích các thừa số nguyên tố.
  - Duyệt các số nguyên tố có N chữ số.
  - Duyệt các cặp số hữu nghị a, b nhỏ hơn N.
  - Duyệt các số hoàn hảo nhỏ hơn N.
  - Duyệt các cặp số P,  $4P + 1$  là nguyên tố nhỏ hơn N.
  - Duyệt các số nguyên tố nhỏ hơn N có tổng các chữ số là S.
  - Duyệt các số thuận nghịch có N chữ số có tổng các chữ số là S.
  - Duyệt các số thuận nghịch có N chữ số sao cho biểu diễn số đó ở hệ cơ số b cũng là số thuận nghịch.

## Bước cuối cùng mong muốn ta nhận được:

```
#include      "CLASS1.h"
.....
#include      "CLASS13.h"
class TEST : public CLASS1,..,CLASS13;
public :
    void Function(void) {
        int phim;
        do { system("cls");
            cout<<"1. Tập thao tác với số"<<endl;
            ....;
            cout<<"13. Tập thao tác với đồ thị"<<endl;
            cout<<"0. Kết thúc test"<<endl;
            cout<<" Lựa chọn chức năng:"; cin>>phim;
            switch (phim) {
                case 1: Function1(); break;
                ....;
                case 13: Function13(); break;
            }
        } while (phim!=0);
    }
int main (void ) { TEST X; X.Function(); }
```