Lập trình hướng đối tượng với C++

2.1 Đối tượng là gì?

Lập trình hướng đối tượng với C++

2.1 Đối tượng là gì?

Lập trình hướng đối tượng với C++

2.1 Đối tượng là gì?

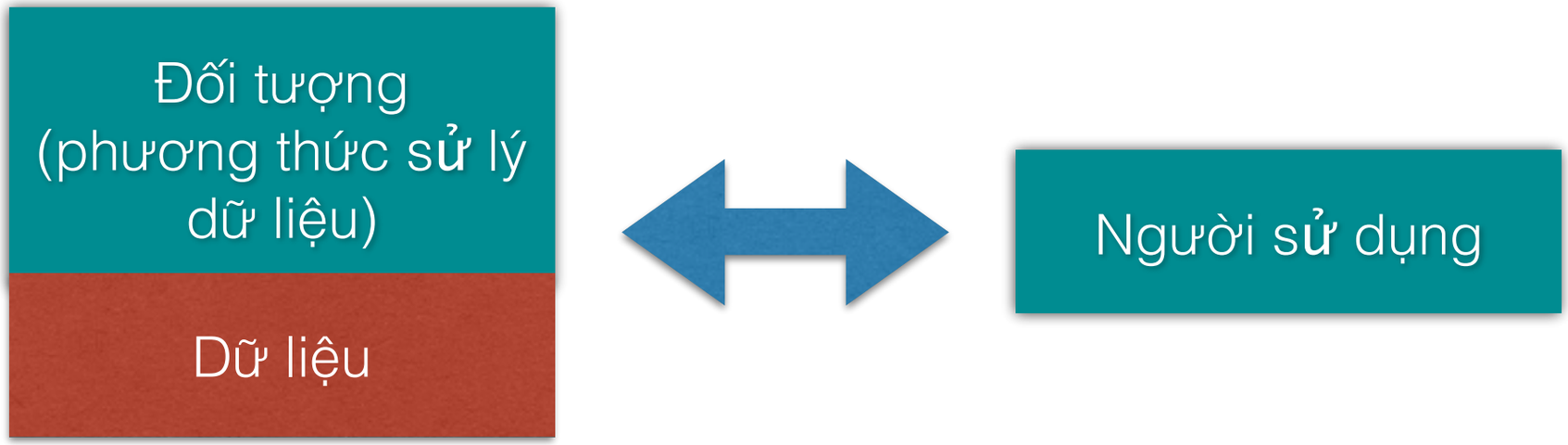
Lập trình hướng đối tượng với C++

2.1 Đối tượng là gì?

Lập trình hướng đối tượng với C++

2.1 Đối tượng là gì?

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*". Đã 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**).

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

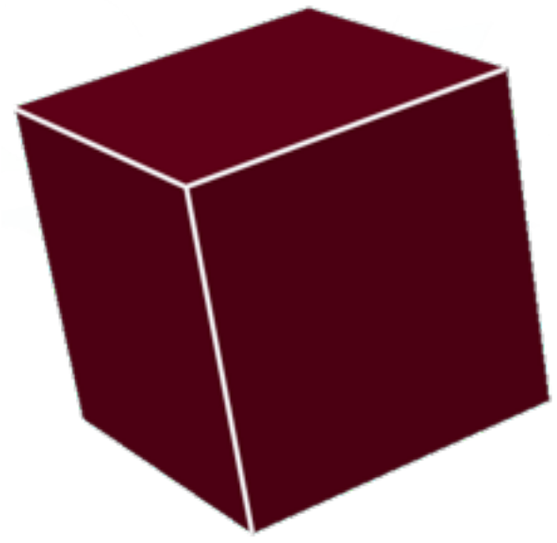
```
class   <class-name> {
        <data type 1>      member1;
        <data type 2>      member2;
         . . . . . .   . . . . . .
        <data type N>      memberN;
};
```

Lập trình hướng đối tượng với C++

2.1 Đối tượng là gì?

Lập trình hướng đối tượng với C++

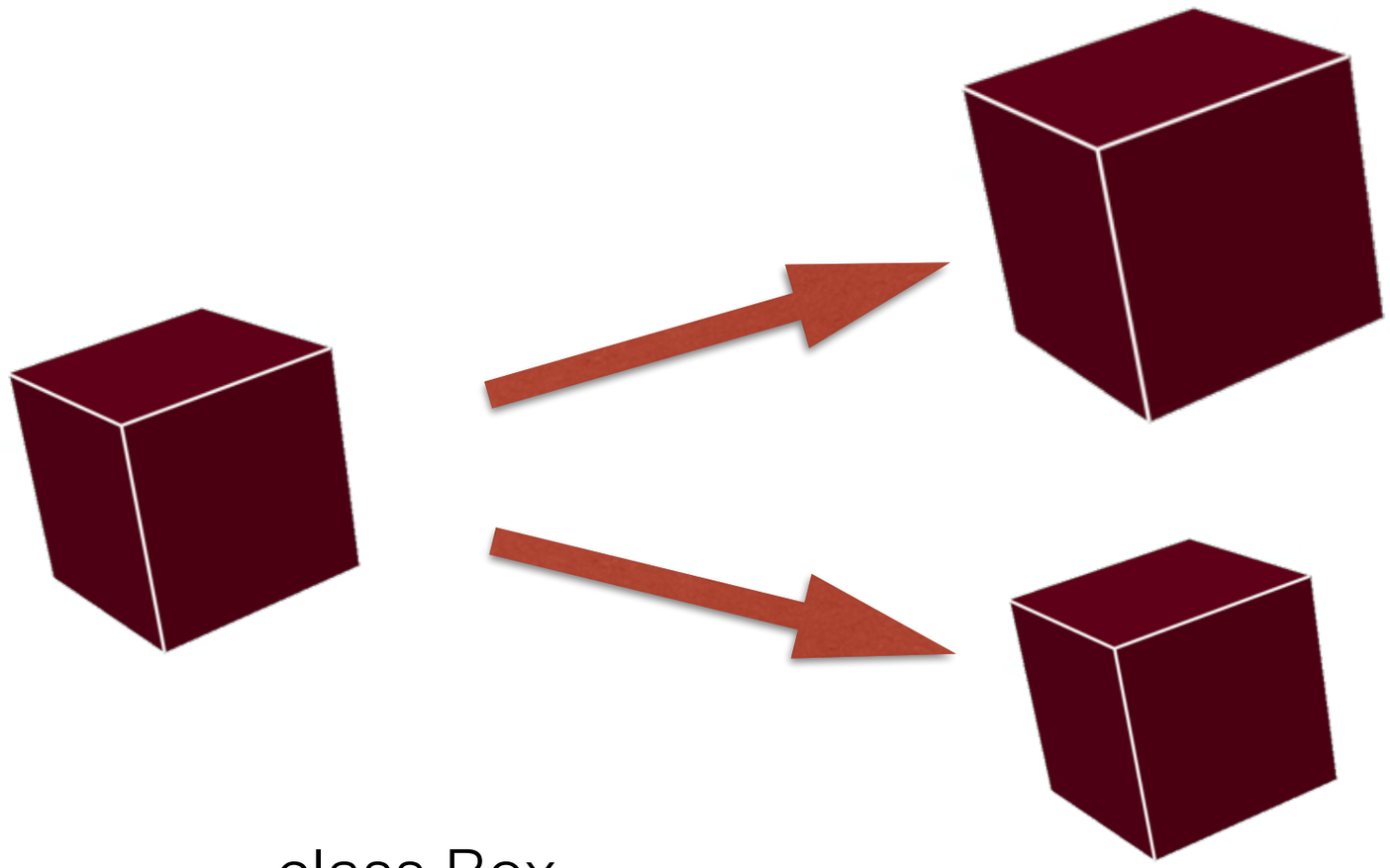
2.2 Làm sao để tạo đối tượng cái hộp ?



```
class Box
{
   public:
      double length;          // Length of a box
      double breadth;         // Breadth of a box
      double height;          // Height of a box
};
```

Lập trình hướng đối tượng với C++

2.2 Làm sao để tạo đối tượng cái hộp ?



Box Box1;

Box Box2;

```
class Box
{
   public:
      double length;          // Length of a box
      double breadth;         // Breadth of a box
      double height;          // Height of a box
};
```

Lập trình hướng đối tượng với C++

2.2 Làm sao để tạo đối tượng cái hộp ?

```
#include <iostream>
using namespace std;

class Box
{
   public:
      double length;   // Length of a box
      double breadth;  // Breadth of a box
      double height;   // Height of a box
};

int main( )
{
   Box Box1;                // Declare Box1 of type Box
   Box Box2;                // Declare Box2 of type Box
   double volume = 0.0;     // Store the volume of a box here

   // box 1 specification
   Box1.height = 5.0;
   Box1.length = 6.0;
   Box1.breadth = 7.0;

   // box 2 specification
   Box2.height = 10.0;
   Box2.length = 12.0;
   Box2.breadth = 13.0;
   // volume of box 1
   volume = Box1.height * Box1.length * Box1.breadth;
   cout << "Volume of Box1 : " << volume <<endl;

   // volume of box 2
   volume = Box2.height * Box2.length * Box2.breadth;
   cout << "Volume of Box2 : " << volume <<endl;
   return 0;
}
```

```
Volume of Box1 : 210
Volume of Box2 : 1560
```

Lập trình hướng đối tượng với C++

2.2.1 Hàm trực thuộc lớp

```
#include <iostream>
using namespace std;

class Box
{
   public:
      double length;          // Length of a box
      double breadth;         // Breadth of a box
      double height;          // Height of a box

      double getVolume();      // Returns box volume
};

class Box
{
   public:
      double length;      // Length of a box
      double breadth;     // Breadth of a box
      double height;      // Height of a box

      double getVolume(void)
      {
         return length * breadth * height;
      }
};
```

```
double Box::getVolume()
{
    return length * breadth * height;
};
```

Lập trình hướng đối tượng với C++

2.2.1 Làm sao để tạo đối tượng cái 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);
   }
};
```

Lập trình hướng đối tượng với C++

2.2.1 Làm sao để tạo đối tượng cái hộp ?

```
#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;
}
```

Lập trình hướng đối tượng với C++

2.2.1 Làm sao để tạo đối tượng cái hộp ?

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) { breadth = z; } //Mô tả chi tiết phương thức SetHeight
float  Box :: Volume( void) {
   return (length * breadth*height);
}
```

Lập trình hướng đối tượng với C++

2.2.1 Làm sao để tạo đối tượng cái hộp ?

```
#include <iostream>

using namespace std;

class Box
{
   public:
      double length;         // Length of a box
      double breadth;      // Breadth of a box
      double height;       // Height of a box

      // Member functions declaration
      double getVolume();
      void setLength( double len );
      void setBreadth( double bre );
      void setHeight( double hei );
};
double Box::getVolume(void)
{
    return length * breadth * height;
}
void Box::setLength( double len )
{
    length = len;
}
void Box::setBreadth( double bre )
{
    breadth = bre;
}

void Box::setHeight( double hei )
{
    height = hei;
}
```

```
// Main function for the program
int main( )
{
   Box Box1;                // Declare Box1 of type Box
   Box Box2;                // Declare Box2 of type Box
   double volume = 0.0;     // Store the volume of a box here

   // box 1 specification
   Box1.setLength(6.0);
   Box1.setBreadth(7.0);
   Box1.setHeight(5.0);

   // box 2 specification
   Box2.setLength(12.0);
   Box2.setBreadth(13.0);
   Box2.setHeight(10.0);

   // volume of box 1
   volume = Box1.getVolume();
   cout << "Volume of Box1 : " << volume <<endl;

   // volume of box 2
   volume = Box2.getVolume();
   cout << "Volume of Box2 : " << volume <<endl;
   return 0;
}
```

Volume of Box1 : 210
Volume of Box2 : 1560

Lập trình hướng đối tượng với C++

Ví dụ quản lý điểm sinh viên

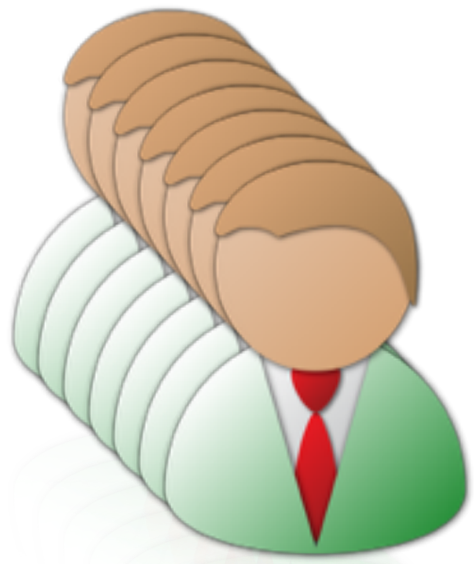
Tên sinh viên : string

Tên lớp sinh viên : string

Điểm lý thuyết : double

Điểm thực hành : double

Điểm trung bình của toàn bộ sinh viên



Lập trình hướng đối tượng với C++

Ví dụ quản lý điểm sinh viên

Tên sinh viên

Lớp sinh viên

Điểm lý thuyết

Điểm thực hành

Tên sinh viên Nông Văn Dền 1

Lớp sinh viên Ptit 13

Điểm lý thuyết 4

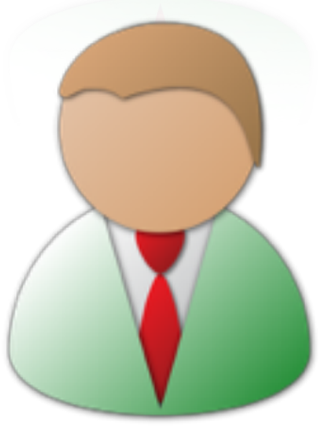
Điểm thực hành 7

Tên sinh viên Nông Văn Dền 2

Lớp sinh viên Ptit 13

Điểm lý thuyết 5

Điểm thực hành 5



Tên sinh viên Nông Văn Dền 3

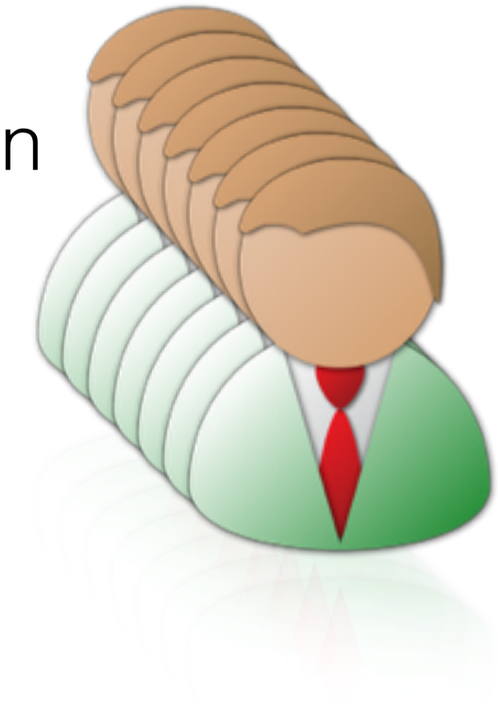
Lớp sinh viên Ptit 13

Điểm lý thuyết 8

Điểm thực hành 8

Lập trình hướng đối tượng với C++

Ví dụ quản lý điểm sinh viên



Tên sinh viên Em tên Nông văn Dền

Lớp sinh viên Ptit13

Điểm lý thuyết 7

Điểm thực hành 9

Điểm trung bình : 8

Kết quả sinh viên Nông Văn Dền Lớp Ptit13 : Qua

Lập trình hướng đối tượng với C++

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 trongMục 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 *this* có trỏ đến chính nó. Nội dung này sẽ được đề cập trong Mục 2.2.7.
Pointer to class	Thự 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.

Lập trình hướng đối tượng với C++

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 (toàn cục) , private (cục bộ), 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*.

Lập trình hướng đối tượng với C++

2.2.2 Quyền Truy nhập đến các thành viên của đối tượng

```
#include    <iostream>
using namespace std;
class Box {//Bieu dien lop 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 tich Box1:"<<V<<endl;

                                      //Truy cập hàm thành viên public từ bên ngoài class
     V = Box2.Volume();cout<<"The tich Box2:"<<V<<endl;
     system("PAUSE"); return 0;
}
```

Lập trình hướng đối tượng với C++

2.2.2 Quyền Truy nhập đến các thành viên của đối tượng

```
#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<<"Nhap chieu dai:"; cin>>length;      //OK: Truy nhập bên trong lớp
          cout<<"Nhap chieu rong:"; cin>>breadth;    //OK: Truy nhập bên trong lớp
          cout<<"Nhap chieu 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 tich Box1:"<<V<<endl;
     system("PAUSE"); return 0;
}
```

Lập trình hướng đối tượng với C++

2.2.2 Quyền Truy nhập đến các thành viên của đối tượng

```
#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<<"Nhap x="; cin>>x; length =x;}
      void SetBreadth(float y)      {cout<<"Nhap y="; cin>>y; breadth =y;}
      void SetHeight(float z)       {cout<<"Nhap z="; cin>>z; height =z; }
   public:                                  //Truy nhap ben ngoai 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 tich Box1:"<<V<<endl;
      system("PAUSE"); return 0;
}
```

Lập trình hướng đối tượng với C++

2.2.2 Quyền Truy nhập đến các thành viên của đối tượng - friend functions

```
#include <iostream>
using namespace std;
 class Box
{
   double width;
public:
   friend void printWidth( Box box );
   void setWidth( double wid );
};

// Member function definition
void Box::setWidth( double wid )
{
    width = wid;
}

// Note: printWidth() is not a member function of any class.
void printWidth( Box box )
{
   /* Because printWidth() is a friend of Box, it can
    directly access any member of this class */
   cout << "Width of box : " << box.width <<endl;
}
```

```
int main( )
{
   Box box;

   // set box width without member function
   box.setWidth(10.0);

   // Use friend function to print the wdith.
   printWidth( box );
   return 0;
}
```

Lập trình hướng đối tượng với C++

2.2.2 Quyền Truy nhập đến các thành viên của đối tượng - friend functions

```
// friend class
#include <iostream>
using namespace std;
class Square;
class Rectangle {
    int width, height;
  public:
    int area ()
      {return (width * height);}
    void convert (Square a);
};
class Square {
  friend class Rectangle;
  private:
    int side;
  public:
    Square (int a) {side = a;}
};
void Rectangle::convert (Square a) {
  width = a.side;
  height = a.side;
}
```

```
int main () {
  Rectangle rect;
  Square sqr (4);
  rect.convert(sqr);
  cout << rect.area();
  return 0;
}
```

Lập trình hướng đối tượng với C++

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 trongMục 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 *this* có trỏ đến chính nó. Nội dung này sẽ được đề cập trong Mục 2.2.7.
Pointer to class	Thự 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.

Lập trình hướng đối tượng với C++

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 construcstor 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.

Lập trình hướng đối tượng với C++

2.2.3 Constructor và Destructor

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

    system("PAUSE"); return 0;
}
```

Lập trình hướng đối tượng với C++

2.2.3 Constructor và Destructor

```
#include <iostream>
using namespace std;

class Box
{
   private:
      double length;         // Length of a box
      double breadth;       // Breadth of a box
      double height;        // Height of a box
   public:
      // Member functions declaration
      Box();
      double getVolume();
      void setLength( double len );
      void setBreadth( double bre );
      void setHeight( double hei );
};

Box::Box()
{
        cout << " Nhap Chieu cao : " ; cin>>height;
        cout << " Nhap Chieu rong : " ; cin>>breadth;
        cout << " Nhap Chieu dai : "  ; cin>>length;
}
double Box::getVolume(void){ return length * breadth * height;}
void Box::setLength( double len ){length = len;}
void Box::setBreadth( double bre ){  breadth = bre;}
void Box::setHeight( double hei ){ height = hei;}
```

Lập trình hướng đối tượng với C++

2.2.3 Constructor và Destructor

```
int main(){
    Box box;
    cout <<"The tich cua hop la : "<<box.getVolume();
}
```

```
Nhap Chieu cao : 10
Nhap Chieu rong : 10
Nhap Chieu dai : 10
The tich cua hop la : 1000
```

Lập trình hướng đối tượng với C++

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");return 0;
}
```

Lập trình hướng đối tượng với C++

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
```

Lập trình hướng đối tượng với C++

2.2.5 Mảng các đối tượng - con trỏ đến đối tượng

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.
```

Lập trình hướng đối tượng với C++

2.2.5 Mảng các đối tượng - con trỏ đến đối tượng

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;
}
```

Lập trình hướng đối tượng với C++

2.2.5 Mảng các đối tượng - con trỏ đến đối tượng

Ví dụ 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");return 0;
}
```

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

1 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.

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

2. Xây dựng tập thao tác với xâu ký tự:
   Tìm X = {xS1 hoặc xS2}.
   Tìm X = {xS1 và xS2}.
   Tìm X = {xS1 và x không thuộc S2}.
   Tìm tập ký tự và số lần xuất hiện mỗi ký tự trong cả S1, S2 (Không kể ký tự trống).
   Tìm tập ký tự và số lần xuất hiện mỗi ký tự thuộc cả S1 và S2 (Không kể ký tự trống).
   Tìm tập ký tự và số lần xuất hiện mỗi ký tự thuộc S1 nhưng không thuộc S2 (Không kể ký tự trống).
   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 S1 hoặc S2.
   Tìm tập từ và số lần xuất hiện mỗi từ trong S1 và S2.
   Tìm tập từ và số lần xuất hiện mỗi từ trong S1 nhưng không xuất hiện trong S2.

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

3. Xây dựng tập thao tác với đa thức:
   Tạo lập hai đa thức Pn(X), Qm(X).
   Tìm Pn(X0), Qm(X0).
   Tìm đạo hàm cấp L của Pn(x), Qm(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.
4. 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 thuần nhất AX=B.

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

5. 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.
   Kiếm theo chủ đề sách.
6. 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 chẫn lẻ.
   Giải mã file bằng kỹ thuật chẫn lẻ.
   Đổi tên file.
   Loại bỏ file . . .

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

7. 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.

Lập trình hướng đối tượng với C++

2.2.6 Tính kế thừa

. 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-heritance): 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;

Lập trình hướng đối tượng với C++

2.2.6 Tính kế thừa

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 tuy 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 Contructor từ lớp cơ sở.
Các phép toán từ lớp cơ sở.
Các hàm bạn (friend fuction) từ lớp cơ sở.

Lập trình hướng đối tượng với C++

2.2.6 Tính kế thừa

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 pubic. 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.

Lập trình hướng đối tượng với C++

2.2.6 Tính kế thừa

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;
```

Lập trình hướng đối tượng với C++

2.2.7 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).

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 trở 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 tở 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 (Oveload 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 chỉ khác nhau về danh sách đối của hàm, kiểu giá trị trở về của hàm. Do vậy, quá trình xử lý quá tải sẽ không phát hiện ra hàm thích hợp nhất trong trường hợp hai hàm quá tải giống nhau về đối nhưng khác nhau về kiểu giá trị của hàm. Vì vậy, ta không được phép định nghĩa các hàm quá tải chỉ khác nhau duy nhất giá trị trở về của hàm.

Lập trình hướng đối tượng với C++

2.2.7 Xử lý quá tải (Overloading)

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”);return 0;
}
```

Lập trình hướng đối tượng với C++

2.2.7 Xử lý quá tải (Overloading)

Phép toán quá tải (Operatpr Overloading)

Định nghĩa: Lập trình hướng đối tượng cho phép ta định nghĩa nhiều hơn một phép 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 trở về và danh sách các tham biến. Hầu hết các phép toán được định nghĩa thông qua việc thực hiện của tổ hợp các hàm thành viên.

Một số phép toán có thể xây dựng operator overloading

+	-	*	/	%	^
&	|	~	!	,	=
<	>	<=	>=	++	--
<<	>>	==	!=	&&	||
+=	-=	/=	%=	^=	&=
|=	*=	<<=	>>=	[]	()
->	->*	new	new []	delete	delete []

Lập trình hướng đối tượng với C++

2.2.7 Xử lý quá tải (Overloading)

```
#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;
}
```

Lập trình hướng đối tượng với C++

2.2.7 Xử lý quá tải (Overloading)

```
#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;
}
```