

Các khái niệm cơ sở

Nội dung

Các khái niệm

- Đối tượng, lớp
- Thuộc tính và phương thức
- Thông điệp và truyền thông điệp
- Tính bao gói, tính kế thừa, tính đa hình

Đối Tượng (Object)

- **Đối tượng:** là một thực thể mà chúng ta quan sát, bao gồm các trạng thái, các đặc điểm, hành vi.
- Trong hệ thống hướng đối tượng, mọi thứ đều là đối tượng





NBT

3

Đối Tượng Thế Giới Thực (Real Object)

- Một **đối tượng thế giới thực** là một thực thể cụ thể mà thông thường bạn có thể sờ, nhìn thấy hay cảm nhận được.
- Tất cả có trạng thái (state) và hành động (behaviour)

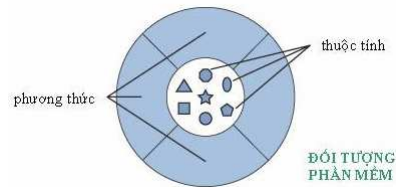
	Trạng thái	Hành động	
Con chó	Tên	Sủa	
	Màu	Vẫy tai	
	Giống	Chạy	
	Vui sướng	Ăn	
Xe đạp	Bánh răng	Tăng tốc	
	Bàn đạp	Giảm tốc	
	Dây xích	Chuyển bánh răng	
	Bánh xe	...	

NBT

4

Đối Tượng Phần Mềm (Software Object)

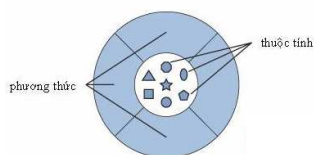
- Các **đối tượng phần mềm** có thể được dùng để *biểu diễn* các đối tượng thế giới thực.
- Cũng có trạng thái và hành động
 - Trạng thái: **thuộc tính** (attribute; property)
 - Hành động: **phương thức** (method)



NBT

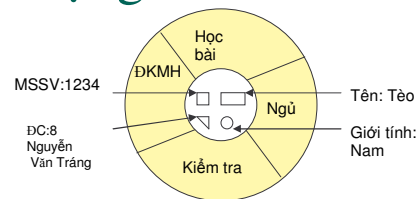
5

Đối Tượng



Đối tượng phần mềm

Đối tượng (object) là một thực thể phần mềm bao gồm các **thuộc tính** và các **phương thức** liên quan.



Đối tượng phần mềm **Sinh Viên**

Thuộc tính được xác định bởi giá trị cụ thể gọi là **thuộc tính thể hiện**. Một đối tượng cụ thể được gọi là một **thể hiện**.

ĐỐI TƯỢNG = THUỘC TÍNH + PHƯƠNG THỨC

NBT

6

Lớp (Class)

- Trong thế giới thực có nhiều đối tượng cùng loại.
- Chương trình hướng đối tượng có nhiều đối tượng cùng loại chia sẻ những đặc điểm chung.
- Ví dụ



NBT

7

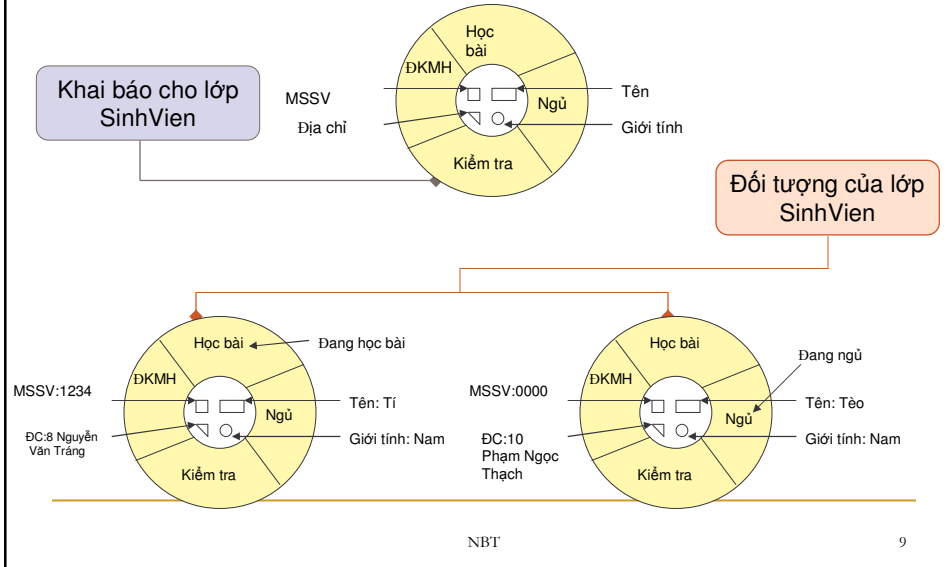
Lớp Là Gì?

- Một **lớp** là một thiết kế hay mẫu (prototype) cho các đối tượng cùng kiểu
 - Ví dụ: lớp `SinhVien` là một thiết kế chung cho nhiều đối tượng sinh viên được tạo ra
- Lớp định nghĩa các thuộc tính và các phương thức chung cho tất cả các đối tượng của cùng một loại nào đó
- Một đối tượng là một **thể hiện** cụ thể của một lớp.
 - Ví dụ: mỗi đối tượng sinh viên là một thể hiện của lớp `SinhVien`
- Mỗi thể hiện có thể có những thuộc tính thể hiện khác nhau
 - Ví dụ: một sinh viên có thể đang học bài trong khi một sinh viên khác có thể là đang ngủ.

NBT

8

Lớp SinhVien



Lớp & Thể hiện của Lớp



Thuộc Tính Lớp & Phương Thức Lớp

- ❑ **Thuộc tính lớp** (class attribute) là một thành phần dữ liệu liên kết với một lớp cụ thể. Nó được định nghĩa bên trong định nghĩa lớp và được sử dụng bởi tất cả các thể hiện của lớp.
- ❑ **Phương thức lớp** (class method) là một thể hiện cách thức thực thi hành vi nào đó của đối tượng. Tất cả các phương thức lớp ảnh hưởng đến toàn bộ lớp chứ không ảnh hưởng đến một lớp riêng rẽ nào.

Thuộc Tính & Phương Thức

- **Thuộc tính** (attribute) là dữ liệu trình bày các đặc điểm về một đối tượng.
- **Phương thức** (method) có liên quan tới những thứ mà đối tượng có thể làm. Một phương thức đáp ứng một chức năng tác động lên dữ liệu của đối tượng (thuộc tính).

Thông điệp & Truyền Thông điệp

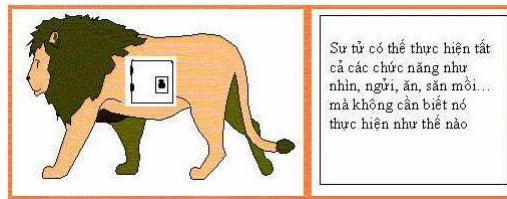
- **Thông điệp** (message) là một lời yêu cầu một hoạt động. Gồm có:
 - Đối tượng nhận thông điệp
 - Tên của phương thức thực hiện
 - Các tham số mà phương thức cần
- **Truyền thông điệp**: một đối tượng triệu gọi một hay nhiều phương thức của đối tượng khác để yêu cầu thông tin.

Trừu tượng hoá (Abstraction)

- Là quá trình loại bỏ 1 số chi tiết không quan trọng để tập trung vào đặc tính cốt lõi.
- Chú ý đến **những gì (WHAT)** phương thức/đối tượng thực hiện, không quan tâm đến **cách thực hiện (HOW)**.

Tính Bao Gói (Encapsulation)

- **Đóng gói** (encapsulation) là tiến trình che giấu việc thực thi chi tiết của một đối tượng.



NBT

15

Ẩn Thông Tin (Information Hiding)

- Đóng gói → Thuộc tính được lưu trữ hay phương thức được cài đặt như thế nào → được che giấu đi từ các đối tượng khác



Việc che giấu những chi tiết thiết kế và cài đặt từ những đối tượng khác được gọi là **ẩn thông tin**

NBT

16

Tính Kế Thừa (Inheritance)

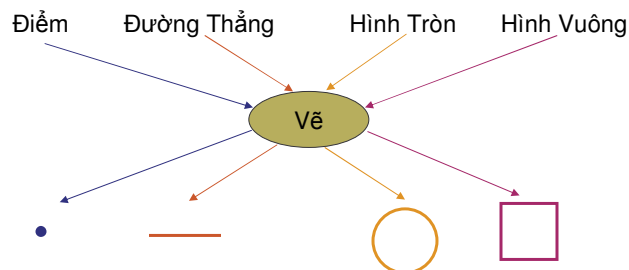
- Hệ thống hướng đối tượng cho phép các lớp được định nghĩa kế thừa từ các lớp khác
 - Ví dụ, lớp **SinhVien** và lớp **GiangVien** là những lớp con (subclass) của lớp **ConNguoi**.
- **Thừa kế** nghĩa là các phương thức và các thuộc tính được định nghĩa trong một lớp có thể được thừa kế hoặc được sử dụng lại bởi lớp khác.

NBT

17

Tính Đa Hình (Polymorphism)

- **Đa hình**: “nhiều hình thức”, hành động cùng tên có thể được thực hiện khác nhau đối với các đối tượng/các lớp khác nhau.
- Ngữ cảnh khác → kết quả khác

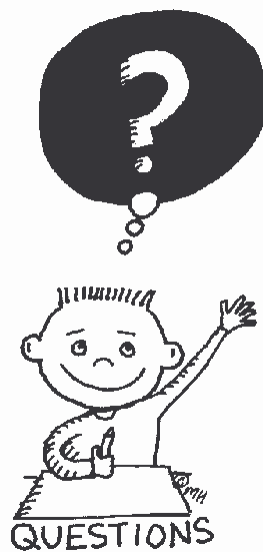


NBT

18

Bài tập

- Tập phân tích một đối tượng để xác định thuộc tính và hành vi: Viên gạch trong trò chơi xếp gạch, phân số, ma trận.....



Lớp

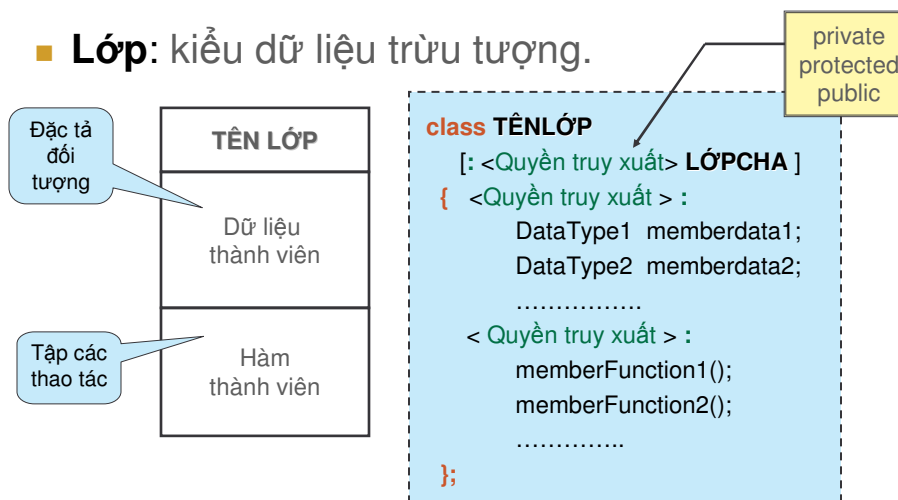
- Lớp
- Khai báo, định nghĩa 1 lớp đơn giản
- Hàm xây dựng (constructor)
- Hàm hủy (destructor)
- Hàm bạn (friend) – Lớp bạn
- Đối số mặc định
- Đối số thành viên ẩn (con trỏ this)

NBT

21

Khái niệm lớp

- **Lớp**: kiểu dữ liệu trừu tượng.



NBT

22

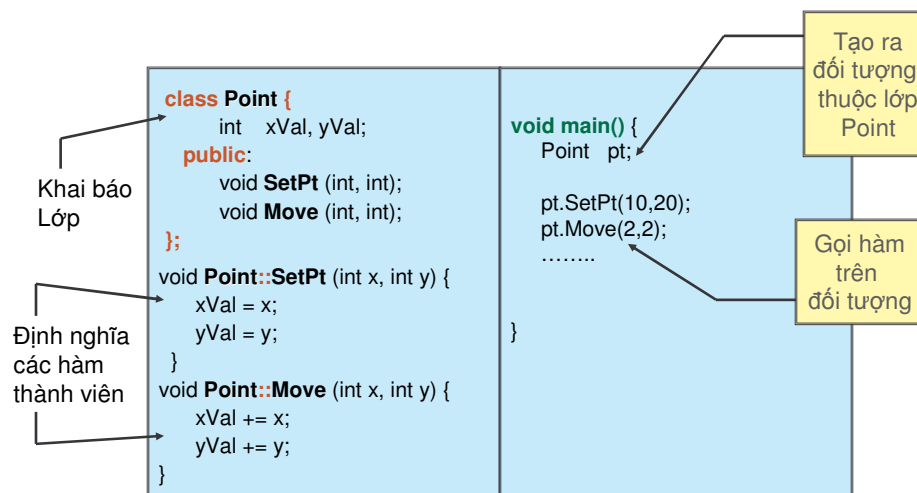
Lớp Point

TÊN LỚP	Point
Dữ liệu thành viên	Int xVal, yVal;
Hàm thành viên	setPoint(int x, int y); MovePt(int x, int y)

NBT

23

Ví dụ lớp



NBT

24

Phạm vi lớp và truy xuất các thành viên lớp

- Trong phạm vi lớp các thành viên có thể được trực tiếp bởi tất cả các hàm thành viên của lớp đó.
- Ngoài phạm vi lớp các thành viên được tham chiếu thông qua tên của đối tượng, tham chiếu đối tượng, hoặc con trỏ đối tượng.
- Biến được khai báo trong hàm thành viên được xem như là biến cục bộ của hàm thành viên đó.
- Nếu trong hàm thành viên, biến được khai báo trùng với tên thuộc tính (biến thành viên của lớp) thì biến thành viên bị che đi và được truy xuất bởi toán tử định vị.

NBT

25

Toán tử phạm vi

- **Toán tử ::** dùng để xác định chính xác hàm (thuộc tính) được truy xuất thuộc lớp nào.
- Câu lệnh: `pt.OffsetPt(2,2);`
`<=>` `pt.Point::OffsetPt(2,2);`
- Cần thiết trong một số trường hợp:
 - Cách gọi hàm trong thừa kế.
 - Tên thành viên bị che bởi biến cục bộ.

Ví dụ: `Point(int xVal, int yVal) {`
`Point::xVal = xVal;`
`Point::yVal = yVal;`
`}`

NBT

26

Ví dụ

```
#include <iostream.h>
class Point{
public:
    int x, y;
    void setPt(int x, int y);
    void MovePt(int x, int y);
    void Print(){
        cout<< "(" << x << "," << y << ")" << endl;
    }
};
void Point::setPt(int x, int y){
    Point::x = x;
    Point::y = y;
}
void Point::setPt(int x, int y){
    Point::x = Point::x + x;
    Point::y = Point::y + y;
}
```

NBT

27

Ví dụ

```
void main(){
    Point point; // tạo đối tượng point
    Point *pointPtr = &point; // con trỏ trỏ tới point
    Point &pointRef = point; // tham chiếu tới point

    cout<<"Su dung ten doi tuong de truy xuat";
    point.setPt(1,4);
    point.Print();

    cout<<"Su dung con trỏ doi tuong de truy xuat";
    pointPtr->setPt(3,5);
    pointPtr->Print();

    cout<<"Su dung tham chiếu de truy xuat";
    pointRef.setPt(4,2);
    pointRef.Print();
}
```

NBT

28

Phạm vi lớp

- Lớp toàn cục: đại đa số lớp trong C++.
- Lớp lồng nhau: lớp chứa đựng lớp.
- Lớp cục bộ: trong 1 hàm hoặc 1 khối.

```
class Rectangle { // Lớp lồng nhau
public:
    Rectangle (int, int, int, int);
    //...
private:
    class Point {
    public:
        Point(int a, int b) { ... }
    private:
        int x, y;
    };
    Point topLeft, botRight;
};
Rectangle::Point pt(1,1); // sđ ở ngoài
```

```
void Render (Image &i)
{
    class ColorTable {
    public:
        ColorTable () { /* ... */ }
        AddEntry (int r, int g, int b)
            { /* ... */ }
        //...
    };
    ColorTable colors;
    //...
}
ColorTable ct; // SAI
```

NBT

29

Các từ khoá modifier

- public:
- protected:
- private:

NBT

30

Hàm khởi tạo (Constructor)

- Có tên trùng với tên lớp, không có kiểu trả về.
- Không gọi trực tiếp, sẽ được tự động gọi khi khởi tạo đt.
- **Gán giá trị, cấp vùng nhớ** cho các *dữ liệu thành viên*.
- Hàm khởi tạo mặc định tự động tạo ra nếu trong lớp chưa có hàm khởi tạo

```
class Point {  
    int xVal, yVal;  
public:  
    Point (int x, int y) {  
        xVal = x; yVal = y;  
    }  
    void MovePt (int x, int y) {  
        xVal += x; yVal += y;  
    }  
};
```

```
void main() {  
    Point pt1(10,20);  
    pt1.MovePt(2,2);  
    .....  
    // Khai báo nào là sai ?  
    Point pt2;  
    Point pt3();  
    Point pt4 = Point(5,5);  
    Point pt5 = new Point(5,5);  
    .....  
}
```

NBT

31

Hàm khởi tạo

```
class Point {  
    int xVal, yVal;  
public:  
    Point () {  
        xVal = 0; yVal = 0;  
    }  
    Point (int x, int y) {  
        xVal = x; yVal = y;  
    }  
    Point (float len, float angle) {  
        xVal = (int) (len * cos(angle));  
        yVal = (int) (len * sin(angle));  
    }  
    void Move (int x, int y){....}  
};  
void main() {  
    Point p1;  
    Point p2(10,20);  
    Point p3(60.3, 3.14);  
}
```

NBT

32

Hàm hủy

- Dọn dẹp 1 đối tượng *trước khi* nó được thu hồi.
- Cú pháp: `~TenLop() { }`
- Không gọi trực tiếp, sẽ được tự động gọi khi hủy bỏ đt.
- **Thu hồi vùng nhớ** cho các dữ liệu thành viên là con trỏ.

```
class Set {  
    private:  
        int *elems;  
        int maxCard;  
        int card;  
    public:  
        Set(const int size) { ..... }  
        ~Set() { delete elems; }  
        ....  
};
```

```
void TestFunc1(Set s1) {  
    Set *s = new Set(50);  
}  
  
void main() {  
    Set s1(40), s2(50);  
    TestFunc1(s1);  
}
```

Tổng cộng
có bao nhiêu
lần hàm hủy
được gọi ?

NBT

33

Đối số mặc định

- Đối số mặc định tính từ bên phải.

```
class Point {  
    int xVal, yVal;  
    public:  
        Point (int x = 0, int y = 0);  
        //...  
};  
  
void main() {  
    Point p1;           // như là ???  
    Point p2(10);       // như là ???  
    Point p3(10,20);  
    Point p4(, 20); // ??????  
    .....  
}
```

```
class Point {  
    int xVal, yVal;  
    public:  
        Point (int x = 0, int y = 0);  
        Point (float x=0, float y=0);  
        //...  
};  
  
void main() {  
    Point p2(1.6, 5.0); // như là ???  
    Point p3(10,20);    // như là ???  
    Point p4;            // ??????  
    .....  
}
```

NBT

34

Danh sách khởi tạo thành viên

- Tương đương việc gán giá trị dữ liệu thành viên.

```
class Point {  
    int xVal, yVal;  
    public:  
        Point (int x, int y) {  
            xVal = x;  
            yVal = y;  
        }  
    // .....  
};
```

```
Point::Point (int x, int y)  
: xVal(x), yVal(y)  
{ }
```

```
class Image {  
    public:  
        Image(const int w, const int h);  
    private:  
        int width;  
        int height;  
        //...  
};  
Image::Image(const int w, const int h) {  
    width = w;  
    height = h;  
    //.....  
}
```

```
Image::Image (const int w, const int h)  
: width(w), height(h)  
{ //..... }
```

NBT

35

Bài tập

- Viết lớp Stack

NBT

36

Hàm khởi tạo sao chép

```
#include <iostream>
class Person {
public: int age;
      Person(int age) : age(age) {}
};
void main() {
    Person timmy(10);
    Person sally(15);
    Person timmy_clone = timmy;
    cout << timmy.age << " " << sally.age << " " << timmy_clone.age << endl;
    timmy.age = 23;
    cout << timmy.age << " " << sally.age << " " << timmy_clone.age << endl;
}
```

```
10 15 10
23 15 10
```

NBT

37

Hàm khởi tạo sao chép

```
#include <iostream>
class Person {
public: int age;
      Person(int age) : age(age) {}

      Person(Person const& copy) : age(copy.age) {}
};
void main() {
    Person timmy(10);
    Person sally(15);
    Person timmy_clone = timmy;
    cout << timmy.age << " " << sally.age << " " << timmy_clone.age << endl;
    timmy.age = 23;
    cout << timmy.age << " " << sally.age << " " << timmy_clone.age << endl;
}
```

NBT

38

Hàm khởi tạo sao chép

```
#include <iostream.h>
class Array {
public: int size;
       int* data;
       Array(int size) : size(size), data(new int[size]) {}
       ~Array() { delete data; }
};
int main() {
    Array first(20);
    first.data[0] = 25;
    Array copy = first;
    cout << first.data[0] << " " << copy.data[0] << endl;

    first.data[0] = 10;
    cout << first.data[0] << " " << copy.data[0] << endl;
}
```

```
g++ "D:\TEACHING\LAP TRÌNH HUONG DOI TUONG _ C++\BAITAP\vd1V
25 25
10 10
Press any key to continue
```

39

Hàm khởi tạo sao chép

```
#include <iostream.h>
class Array {
public: int size;
       int* data;
       Array(int size) : size(size), data(new int[size]) {}
       ~Array() { delete data; }
       Array(const Array &copy) : size(copy.size), data(copy.data) {}
       //tự động tạo
};
int main() {
    Array first(20);
    first.data[0] = 25;
    Array copy = first;
    cout << first.data[0] << " " << copy.data[0] << endl;

    first.data[0] = 10;
    cout << first.data[0] << " " << copy.data[0] << endl;
}
```

NBT

40

Hàm khởi tạo sao chép

```
#include <iostream.h>
#include <string.h>
class Array {
public: int size;
       int* data;
       Array(int size) : size(size), data(new int[size]) {}
       ~Array() { delete data; }
       Array(const Array &copy)
           : size(copy.size), data(new int[copy.size]) {
           memcpy((void*)data, copy.data, sizeof(copy.data));
       }
};

int main() {
    Array first(20);
    first.data[0] = 25;
    Array copy = first;
    cout << first.data[0] << " " << copy.data[0] << endl;

    first.data[0] = 10;
    cout << first.data[0] << " " << copy.data[0] << endl;
}
```



"D:\TEACHING\LAP TRÌNH HUONG DỐI TUONG _ C++\BAIT

25 25
10 25
Press any key to continue

41

Phương thức chồng(Overloading)

```
class PhanSo {
private:
    int tuso, mauso;
public :
    PhanSo(int t=0, int m=1) : tuso(t), mauso(m){}

    PhanSo Cong(PhanSo);
    PhanSo Cong(PhanSo , PhanSo);

    float layGiatri();
    void gianUoc();
    void Print();
};
```

NBT

42

```

D:\teaching\lap trình huong doi tuong _ C++\baitap\PhanSoDebug
#include "PhanSo.h"
#include <iostream.h>

PhanSo PhanSo::Cong(PhanSo p)
{
    PhanSo q;
    q.tuso = tuso*p.mauso+maus
    q.mauso = mauso*p.mauso;

    q.gianUoc();
    return q;
}

PhanSo PhanSo::Cong(PhanSo p, PhanSo q){
    PhanSo res;
    res.tuso = q.tuso*p.mauso+q.mauso*p.tuso;
    res.mauso = q.mauso*p.mauso;

    res.gianUoc();
    return res;
}

float PhanSo::layGiatri(){
    return (float)tuso/mauso;
}

void PhanSo::Print(){
    cout<<tuso<<"/"<<mauso<<"="<<
    this->layGiatri()<<endl;
}

    if(tu>mau) tu=tu-mau;
    else mau = mau-tu;
}while(tu!=mau);
tuso = tuso/tu;
mauso = mauso/mau;
}

void main()
{
    PhanSo p(4,5), p1(1,3);
    p.Print();
    p1.Print();
    PhanSo res = p.Cong(p,p1);
    res.Print();
    p.Cong(p1);
    p.Print()
}
4/5=0.8
1/3=0.333333
17/15=1.13333
29/15=1.93333
Press any key to continue

```

NBT

43

Tham trị

```

class Point
{
    int xVal, yVal;
public:
    Point(){xVal = 0; yVal = 0;}
    Point(int, int){
        xVal = x; yVal = y;
    }

    void PrintPt();
    void InputPoint(Point p);
    ~Point(){}
};

```

```

void Point::PrintPt(){
    cout<<"("<<xVal<<","
    <<yVal<<")"endl;
}

void Point::InputPoint(Point p){
    cout<<"Nhap x"; cin>>p.xVal;
    cout<<"Nhap y"; cin>>p.yVal;
}

void main(){
    Point p1,p2;
    p1.InputPoint(p2);
}

```

NBT

44

Tham chiếu

```
class Point
{
    int xVal, yVal;
public:
    Point(){xVal = 0; yVal = 0;}
    Point(int, int){
        xVal = x; yVal = y;
    }

    void PrintPt();
    void InputPoint(Point &p);
    ~Point(){}
};
```

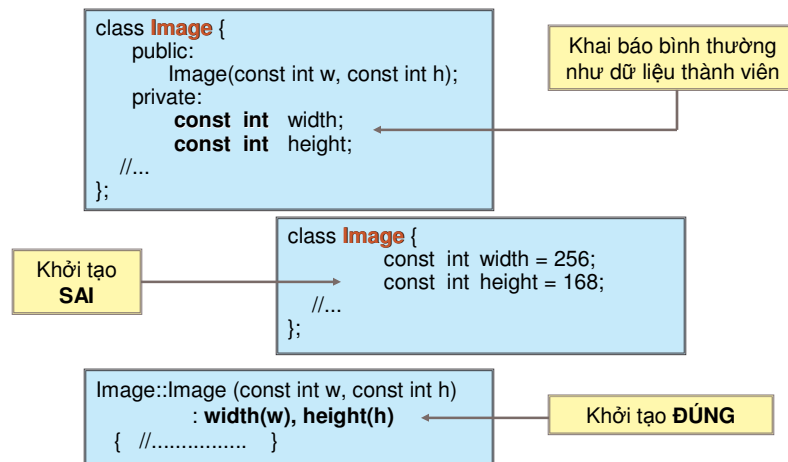
```
void Point::PrintPt(){
    cout<<" "<<xVal<<" "
        <<yVal<<"\n"endl;
}
void Point::InputPoint(Point &p){
    cout<<"Nhập x"; cin>>p.xVal;
    cout<<"Nhập y"; cin>>p.yVal;
}
void main(){
    Point p1,p2;
    p1.InputPoint(p2);
}
```

NBT

45

Thành viên hằng

■ Hằng dữ liệu thành viên:



NBT

46

Thành viên hằng

- **Hằng đối tượng:** không được thay đổi giá trị.
- **Hàm thành viên hằng:**
 - Được phép gọi trên hằng đối tượng.
 - Không được thay đổi giá trị dữ liệu thành viên.

<pre>class Point { private: int xVal, yVal; public: Point(int x, int y): xVal(x), yVal(y){} void PrintPt() const; void SetPt(int x, int y){ xVal = x; yVal = y; } };</pre>	<pre>void Point::PrintPt () const { //... } void main() { const Point p(10,13); p.SetPt(10,14); // SAI p.PrintPt(); // Đúng }</pre>
--	--

NBT

47

Bài tập

- Viết 1 class quản lý tập hợp các số nguyên. Tập hợp này có thể thực hiện các chức năng sau:
 - Khởi tạo tập hợp
 - Kiểm tra 1 số nguyên có phải là phần tử của tập hợp không
 - Thêm một phần tử vào tập hợp
 - Xóa 1 một tử ra khỏi tập hợp
 - Sao chép các ptử từ tập hợp này sang tập hợp khác. (1 pthức truyền 1 tham số, 1 pthức truyền 2 tham số)
 - Tạo một phương thức sao chép constructor
 - Giao 2 tập hợp
 - Hội 2 tập hợp
 - Xuất tập hợp ra màn hình (khai báo hàm hằng)
 - Thực hiện tạo ra bộ số liệu để test các chức năng trên.

<http://mail.hoasen.edu.vn/home/nbtrung/Briefcase/LTHDT>

NBT

48

Ví dụ: Tập các số nguyên

```
class IntSet{
private: int *elems;
        int num,max;
public:
    IntSet();
    IntSet(int);
    int IsMember(const int);
    void AddElem(const int);
    void RmvElem(const int);
    void Copy(IntSet&);
    int Equal(IntSet&);
    void InterSet(IntSet&, IntSet&);
    void Union(IntSet&, IntSet&);
    void Print();
    void IntSet::SetToReal(RealSet &set);
    ~IntSet(){ }
};
```

```
IntSet::IntSet(int size){
    max = size; num=0;
    elems = new int[size];
}
IntSet::IntSet(){
    elems = new int[100];
    num = 100;
}
int IntSet::IsMember(const int e){
    for(int i=0; i<num; i++)
        if(elems[i]==e)
            return 1;
    return 0;
}
.....
```

NBT

49

Ví dụ: Tập các số thực

```
class RealSet{
private: int *elems;
        int num, max;
public:
    RealSet();
    RealSet(int);
    void EmptySet(){num =0;}
    int IsMember(const float);
    void AddElem(const float);
    void RmvElem(const float);
    void Copy(RealSet&);
    int Equal(RealSet&);
    void InterSet(RealSet&, RealSet&);
    void Union(RealSet&, RealSet&);
    void Print();
    ~RealSet(){ }
};
```

```
RealSet::RealSet(int size){
    max = size; num=0;
    elems = new float[size];
}
RealSet::RealSet(){
    num = 100;
    elems = new float[100];
}
int RealSet::IsMember(const float e){
    for(int i=0; i<num; i++)
        if(elems[i]==e)
            return 1;
    return 0;
}
.....
```

NBT

50

Truy xuất tới thành viên private ???

```
class IntSet {  
    public:  
        //...  
    private:  
        int *elems;  
        int num;  
};  
class RealSet {  
    public:  
        //...  
    private:  
        float *elems;  
        int num;  
};
```

```
void IntSet::SetToReal (RealSet &set) {  
    set.num = num;  
    for (int i = 0; i < num; ++i)  
        set.elems[i] = (float) elems[i];  
}
```

Hàm SetToReal dùng để chuyển tập số nguyên thành tập số thực



Làm thế nào để thực hiện được việc truy xuất đến thành viên **Private** ?

NBT

51

Hàm bạn (Friend)

- **Cách 1:** Khai báo hàm thành viên của lớp **IntSet** là bạn (friend) của lớp **RealSet**.

Giữ nguyên định nghĩa của lớp **IntSet**

Thêm dòng khai báo **friend** cho hàm thành viên **SetToReal**

```
class IntSet {  
    public:  
        //...  
    private:  
        int *elems;  
        int num;  
};  
class RealSet {  
    public:  
        //...  
    private:  
        float *elems;  
        int num;  
};  
  
friend void IntSet::SetToReal (RealSet&);
```

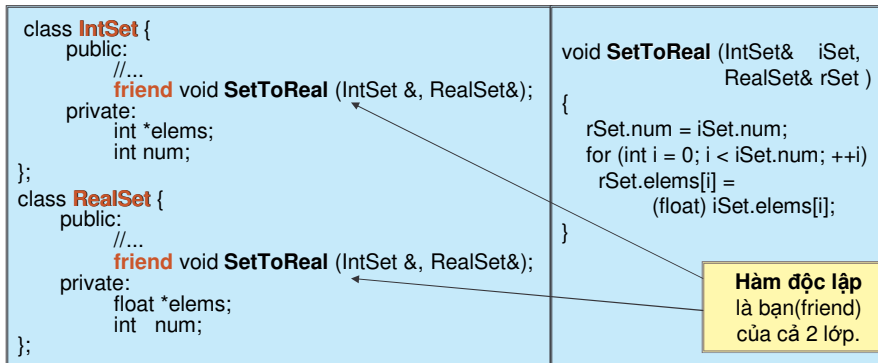
NBT

52

Hàm bạn (Friend)

■ Cách 2:

- ❑ Chuyển hàm SetToReal ra ngoài (**độc lập**).
- ❑ Khai báo hàm đó là **bạn** của cả 2 lớp.



NBT

53

Bạn (Friend)

■ Hàm bạn:

- ❑ Có quyền truy xuất đến tất cả các *dữ liệu* và *hàm* thành viên (protected + private) của 1 lớp.

■ Lớp bạn:

- ❑ Tất cả các hàm trong lớp bạn: là hàm bạn.

```
class A;
class B { // .....
    friend class A;
};
```

```
class IntSet { ..... }
class RealSet { // .....
    friend class IntSet;
};
```

NBT

54

Thành viên tĩnh

■ Dữ liệu thành viên tĩnh:

- Dùng chung 1 bản sao chép (1 vùng nhớ) chia sẻ cho tất cả đối tượng của lớp đó.
- Sử dụng: <TênLớp>::<TênDữLiệuThànhViên>
- Thường dùng để đếm số lượng đối tượng.

Khởi tạo
dữ liệu
thành viên
tĩnh

```
#include <iostream.h>
class Student{
    int ID;
public:
    static int nextID;
    Student(int id){
        ID = id; nextID++;
    }
    int getNextID(){
        return nextID;
    }
};
```

Khai báo

```
int Student::nextID =0;
void main(){
    Student st(4);
    Student st1(5);
    Student st2(6);

    cout<<"so doi tuong"
         <<Student::nextID;
    cout<<"so doi tuong"
         <<st.nextID;
}
```

NBT

55

Thành viên tĩnh

■ Hàm thành viên tĩnh:

- Tương đương với hàm toàn cục.
- Gọi thông qua: <TênLớp>::<TênHàm>

```
#include <iostream.h>
class Student{
    int ID;
public:
    static int nextID;
    Student(int id){
        ID = id; nextID++;
    }
    static int getNextID(){
        return nextID;
    }
};
```

```
int Student::nextID =0;
void main(){
    Student st(4);
    Student st1(5);
    Student st2(6);

    cout<<"so doi tuong"
         <<Student::getNextID();
    cout<<"so doi tuong"
         <<st.getNextID();
}
```

NBT

56

Đối số thành viên ẩn

■ Con trỏ ***this**:

- Là 1 thành viên ẩn, có thuộc tính là private.
- Trỏ tới chính bản thân đối tượng.

```
void Point::OffsetPt (int x, int y) {  
    xVal += x;  
    yVal += y;  
}
```



```
void Point::OffsetPt (int x, int y) {  
    this->xVal += x;  
    this->yVal += y;  
}
```



- Có những trường hợp sử dụng ***this** là dư thừa (Ví dụ trên)
- Tuy nhiên, có những trường hợp phải sử dụng con trỏ ***this**

NBT

57

Các đối tượng được cấp phát động

- Các đối tượng có thể cấp phát động giống như các dữ liệu khác bằng toán tử **new**, **delete**.

```
Time *timePtr = new Time(1,26,30);
```

```
.....
```

```
delete timePtr;
```

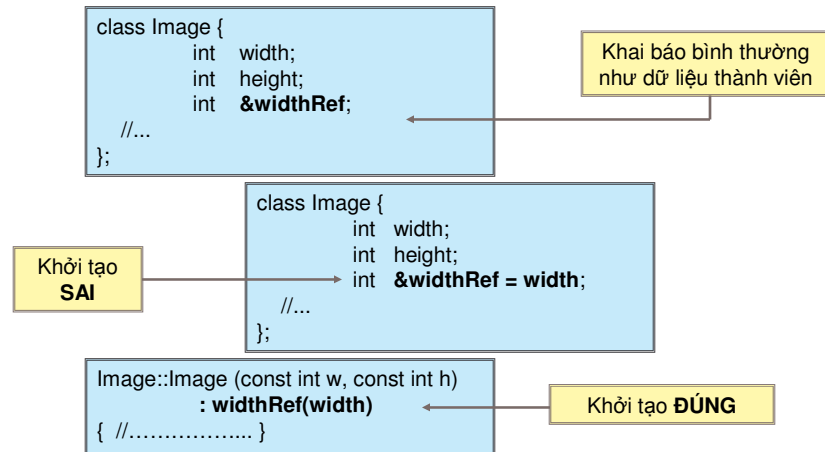
- Toán tử **new** tự động gọi hàm constructor và toán tử **delete** tự động gọi hàm destructor

NBT

58

Thành viên tham chiếu

■ Tham chiếu dữ liệu thành viên:



NBT

59

Mảng các đối tượng

■ Sử dụng **hàm khởi tạo không đối số** (hàm xây dựng mặc nhiên - default constructor).

VD: `Point pentagon[5];`

■ Sử dụng bộ khởi tạo mảng:

VD: `Point triangle[3] =
{ Point(4,8), Point(10,20), Point(35,15) };`

Ngắn gọn:

`Set s[4] = { 10, 20, 30, 40 };`

tương đương với:

`Set s[4] = { Set(10), Set(20), Set(30), Set(40) };`

NBT

60

Mảng các đối tượng

■ Sử dụng dạng con trỏ:

- Cấp vùng nhớ:

VD: `Point *pentagon = new Point[5];`

- Thu hồi vùng nhớ:

```
delete[] pentagon;  
delete pentagon;    // Thu hồi vùng nhớ đầu
```

```
class Polygon {  
public:  
    //...  
private:  
    Point *vertices; // các đỉnh  
    int num;         // số các đỉnh  
};
```

NBT

61

Thành viên là đối tượng của 1 lớp

■ Dữ liệu thành viên có thể có kiểu:

- Dữ liệu (lớp) chuẩn của ngôn ngữ.
- Lớp do người dùng định nghĩa (có thể là chính lớp đó).

```
class Point { ..... };  
class Rectangle {  
public:  
    Rectangle (int left, int top, int right, int bottom);  
    //...  
private:  
    Point topLeft;  
    Point botRight;  
};  
Rectangle::Rectangle (int left, int top, int right, int bottom)  
: topLeft(left,top), botRight(right,bottom)  
{ }
```

Khởi tạo cho các
dữ liệu thành viên
qua danh sách
khởi tạo thành viên

NBT

62

Sửa bài tập

- Viết class Complex
- Viết class Binary
- *Khi đối tượng là tham số của hàm lưu ý. Nên truyền tham biến vì nếu truyền tham trị sẽ tự động sao chép ra một đối tượng khác, nếu trong class có thuộc tính là con trỏ và hàm destructor giải phóng con trỏ mà không sử dụng hàm copy constructor sẽ gây ra lỗi --> do nó huỷ luôn đối tượng truyền vào.*

THỪA KẾ (INHERITANCE)

Nội dung

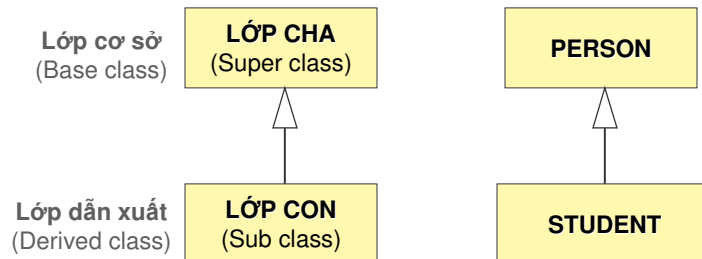
- Khái niệm
- Lớp dẫn xuất đơn giản
- Thành viên lớp được bảo vệ
- Lớp cơ sở riêng, chung và được bảo vệ
- Đa thừa kế - Sự mơ hồ
- Hàm ảo - Lớp cơ sở ảo
- Chuyển kiểu

NBT

65

Khái niệm

- Kế thừa từ các lớp có từ trước.
- Ích lợi: có thể tận dụng lại
 - Các thuộc tính chung
 - Các hàm có thao tác tương tự



NBT

66

Ví dụ minh họa

Ký hiệu
composition



```

#include <iostream.h>
#include <string.h>
class Date {
private:
    int day;
    int month;
    int year;
public:
    Date (const int d, const int m, const int y):
        day(d), month(m),year(y){}
    ~Date ();
    const void Print() {
        cout<<day<<"/"<<month<<"/"<<year<<endl;
    }
};
    
```

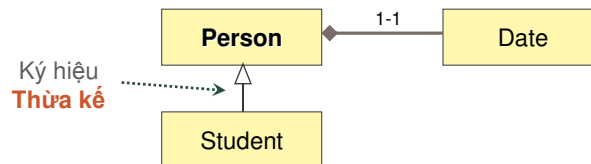
```

class Person{
public:
    Date date;
    char *name;
    char *place;
public:
    Person(){}
    Person(Date d, char* n, char* p);
    void Print();
};
Person::Person(Date d, char *n, char *p)
: date(d), name(n), place(p){}
void Person::Print() {
    cout<<"Ho ten: "<<name<<endl;
    cout<<"Ngày sinh: "; date.Print();
    cout<<"Nơi sinh:"<<place<<endl;
}
    
```

NBT

67

Ví dụ minh họa (tt)



Ký hiệu
Thừa kế

NBT

68

```

class Student:public Person{
    char *maso;
    int diem1, diem2, diem3;
    float tb;

public:
    Student(char*, Date , char*, int, int, int);
    void TinhTB();
    void Print();
};

Student::Student(char *n, Date d, char *p, int d1, int d2, int d3)
    :Person(d,n,p){
    diem1 = d1; diem2 = d2; diem3 = d3;
}
void TinhTB(){
    tb = (diem1+diem2+diem3)/(float)3;
}
void Print(){
    Person::Print();
    cout<<"diem trung binh: "<< tb<<endl;
}

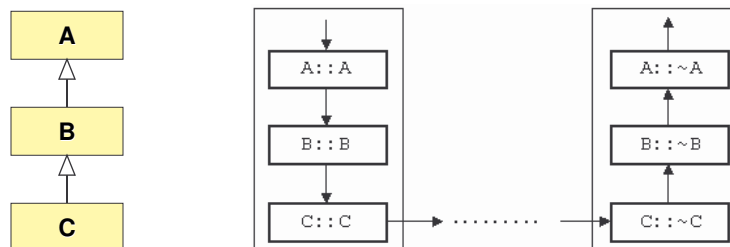
```

NBT

69

Hàm khởi tạo và hàm hủy

- Trong thừa kế, khi khởi tạo đối tượng:
 - Hàm khởi tạo của **lớp cha** sẽ được gọi trước
 - Sau đó mới là hàm khởi tạo của **lớp con**.
- Trong thừa kế, khi hủy bỏ đối tượng:
 - Hàm hủy của **lớp con** sẽ được gọi trước
 - Sau đó mới là hàm hủy của **lớp cha**.



NBT

70

Hàm xây dựng và hàm hủy (tt)

```
class Student:public Person{
    char *maso;
    int diem1, diem2, diem3;
    float tb;
public:
    Student::Student(char *n, Date d, char *p, int d1, int d2, int d3)
        :Person(d,n,p){
        diem1 = d1;
        diem2 = d2;
        diem3 = d3;
    }
    Student(const Student&st):Person(st){}
    ...
};
```

Gọi hàm
khởi tạo
của lớp cha

NBT

71

Thành viên lớp được bảo vệ

- Thừa kế:
 - Có tất cả các dữ liệu và hàm thành viên.
 - Không được truy xuất đến thành viên **private**.
- Thuộc tính truy cập **protected**:
 - Cho phép lớp con truy xuất.

```
class Person{
    Date date;
    char *name;
    char *place;
protected:
    Person(){}
    Person(Date d, char* n, char*p);
    void Print();
};
```

```
class Foo {
    public:
        // các thành viên chung...
    private:
        // các thành viên riêng...
    protected:
        // các thành viên được bảo vệ...
    public:
        // các thành viên chung...
    protected:
        // các thành viên được bảo vệ...
};
```

NBT

72

Lớp cơ sở private, public và protected

Lớp cơ sở	Thừa kế public	Thừa kế private	Thừa kế protected
private	—	—	—
public	public	private	protected
protected	protected	private	protected

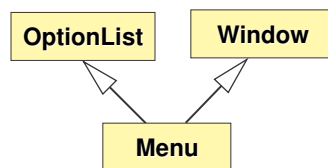
```
class A {
    private:
        int x;
        void Fx (void);
    public:
        int y;
        void Fy (void);
    protected:
        int z;
        void Fz (void);
};
```

```
class B : A { // Thừa kế dạng private
    .....
};
class C : private A { // A là lớp cơ sở riêng của C
    .....
};
class D : public A { // A là lớp cơ sở chung của D
    .....
};
class E : protected A { // A: lớp cơ sở được bảo vệ
    .....
};
```

NBT

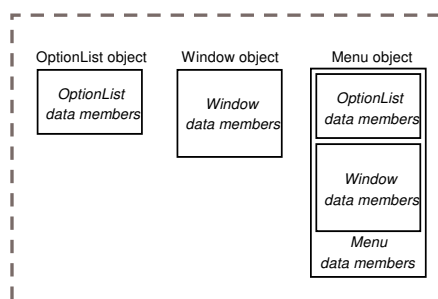
73

Đa thừa kế



```
class OptionList {
    public:
        OptionList (int n);
        ~OptionList ();
        //...
};
```

```
class Window {
    public:
        Window (Rect &);
        ~Window (void);
        //...
};
```



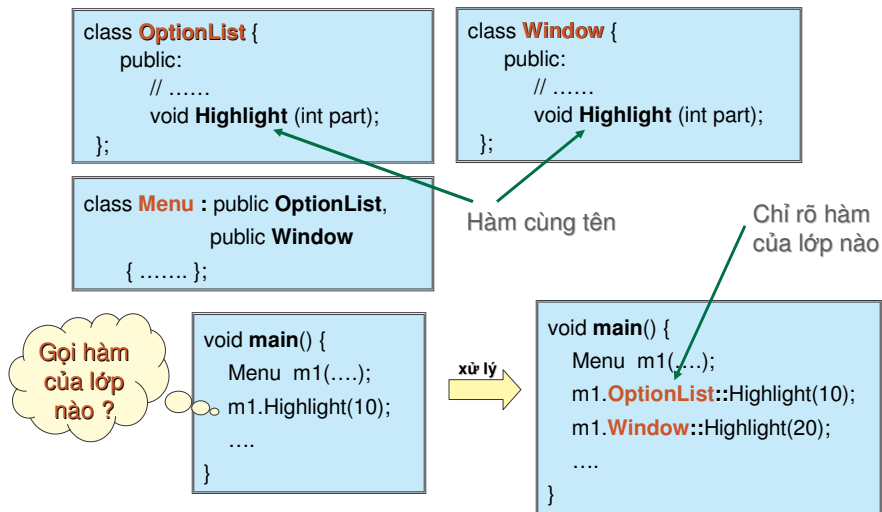
```
class Menu
    : public OptionList, public Window {
    public:
        Menu (int n, Rect &bounds);
        ~Menu (void);
        //...
};

Menu::Menu (int n, Rect &bounds) :
    OptionList(n), Window(bounds)
{ /* ... */ }
```

NBT

74

Sự mơ hồ trong đa thừa kế

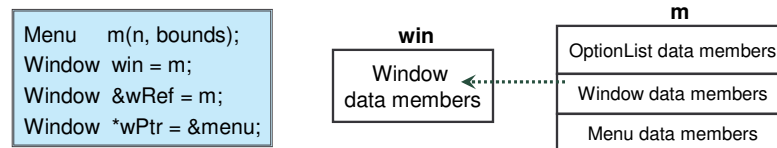


NBT

75

Chuyển kiểu

- Có sẵn 1 phép chuyển kiểu không tường minh:
 - Đối tượng lớp cha = Đối tượng lớp con;
 - Áp dụng cho cả đối tượng, tham chiếu và con trỏ.



- Không được thực hiện phép gán ngược:

- Đối tượng lớp con = Đối tượng lớp cha; // **SAI**

Nếu muốn thực hiện phải tự định nghĩa phép ép kiểu

```

class Menu : public OptionList, public Window {
public:
    //...
    Menu (Window&);
};
    
```

NBT

76

Liên kết tĩnh (*static binding*):

- ❑ Xác định khi biên dịch chương trình.
- ❑ Dùng **hàm thành viên**.
- ❑ Gọi hàm của **lớp cơ sở (lớp cha)**.

Liên kết tĩnh (*static binding*):

```
class Timer {
protected:
    int hrs, mins, secs;
public:
    Timer(int h, int m, int s):hrs(h), mins(m), secs(s){}
    void WriteTime() {
        cout.fill('0'); cout.width(2); cout<<hrs<<":";
        cout.fill('0'); cout.width(2); cout<<mins<<":";
        cout.fill('0'); cout.width(2); cout<<secs<<endl;
    }
};
class ExtTimer : public Timer {
    char *zone;
public:
    ExtTimer(int d, int m, int y, char *z):Timer(d, m, y) {
        zone = z;
    }
    void WriteTime() {
        cout.fill('0'); cout.width(2); cout<<hrs<<":";
        cout.fill('0'); cout.width(2); cout<<mins<<":";
        cout.fill('0'); cout.width(2); cout<<secs<<":"<<zone<<endl;
    }
};
```

Liên kết tĩnh (*static binding*):

```
void Print (Timer someTime){
    cout << "Time is " ;
    someTime.Write ( ) ;
    cout << endl ;
}
void main(){
    Timer startTime ( 8, 30, 0 ) ;
    ExtTime endTime (10, 45, 0, "CST") ;

    Print ( startTime ) ;
    Print ( endTime ) ;           // Timer::WriteTime()
}
```

Gọi hàm
nào ?

OUTPUT

Time is 08:30:00
Time is 10:45:00

NBT

79

Liên kết động (*dynamic binding*)

- ❑ Xác định khi thực thi chương trình.
- ❑ Dùng **hàm ảo** (virtual function).
- ❑ Gọi hàm của **lớp dẫn xuất (lớp con)**.
- ❑ Thể hiện tính **đa hình** của OOP.

NBT

80

Liên kết động (*dynamic binding*)

```
class Timer {
protected:
    int hrs, mins, secs;
public:
    Timer(int h, int m, int s):hrs(h), mins(m), secs(s){}
    virtual void WriteTime() {
        cout.fill('0'); cout.width(2); cout<<hrs<<":";
        cout.fill('0'); cout.width(2); cout<<mins<<":";
        cout.fill('0'); cout.width(2); cout<<secs<<endl;
    }
};
class ExtTimer : public Timer {
    char *zone;
public:
    ExtTimer(int d, int m, int y, char *z):Timer(d, m, y) {
        zone = z;
    }
    void WriteTime() {
        cout.fill('0'); cout.width(2); cout<<hrs<<":";
        cout.fill('0'); cout.width(2); cout<<mins<<":";
        cout.fill('0'); cout.width(2); cout<<secs<<":"<<zone<<endl;
    }
};
```

Liên kết động (*dynamic binding*)

```
void Print (Timer* someTime) {
    cout << "Time is " ;
    someTime->Write ( ) ;
    cout << endl ;
}
void main(){
    Timer startTime ( 8, 30, 0 ) ;
    ExtTime endTime (10, 45, 0, "CST") ;

    Timer * ptr;
    ptr = & startTime;
    Print ( startTime ) ;           // Timer::WriteTime()
    ptr = & endTime;
    Print ( endTime ) ;            // ExtTimer::WriteTime()
}
```

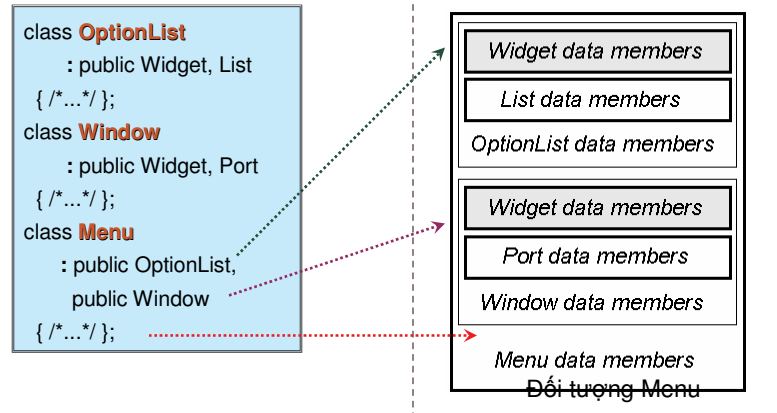
Gọi hàm
nào ?

OUTPUT

Time is 08:30:00
Time is 10:45:00:CST

Lớp cơ sở ảo

■ Sự mơ hồ - dư thừa dữ liệu

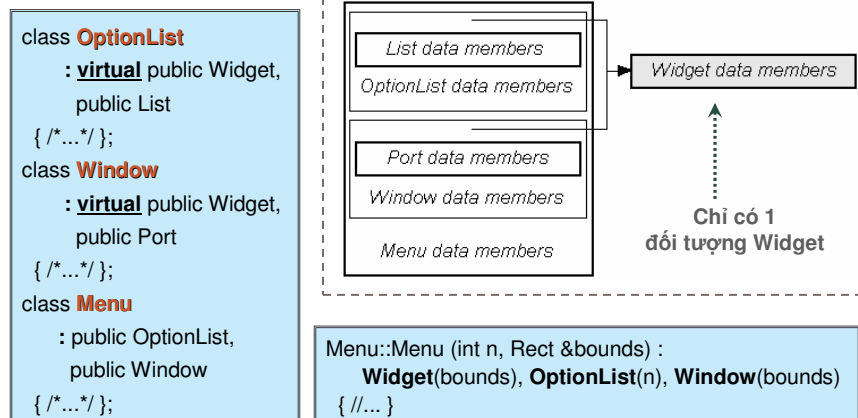


NBT

83

Lớp cơ sở ảo (tt)

■ Cách xử lý: dùng lớp cơ sở ảo.



NBT

84

Tái định nghĩa toán tử

- Định nghĩa các phép toán trên đối tượng.
- Các phép toán có thể tái định nghĩa:

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

- Các phép toán không thể tái định nghĩa:

. .* :: ?: sizeof

NBT

85

Tái định nghĩa toán tử (tt)

- Bảng hàm thành viên

```
class Point {
public:
    Point (int x, int y)    { Point::x = x; Point::y = y; }
    Point operator + (Point &p) { return Point(x + p.x, y + p.y); }
    Point operator - (Point &p) { return Point(x - p.x, y - p.y); }
private:
    int x, y;
};
```

```
void main() {
    Point p1(10,20), p2(10,20);
    Point p3 = p1 + p2;      Point p4 = p1 - p2;
    Point p5 = p3.operator + (p4); Point p6 = p3.operator - (p4);
};
```

NBT

86

Tái định nghĩa toán tử (tt)

- **Bảng hàm độc lập:** thường khai báo **friend**

```
class Point {
public:
    Point (int x, int y) { Point::x = x; Point::y = y; }
    friend Point operator + (Point &p, Point &q)
        {return Point(p.x + q.x, p.y + q.y); }
    friend Point operator - (Point &p, Point &q)
        {return Point(p.x - q.x, p.y - q.y); }
private:
    int x, y;
};
```

Có 2 tham số
(Nếu là toán tử nhị hạng)

```
void main() {
    Point p1(10,20), p2(10,20);
    Point p3 = p1 + p2;      Point p4 = p1 - p2;
    Point p5 = operator + (p3, p4); Point p6 = operator - (p3, p4);
};
```

NBT

87

Tái định nghĩa toán tử (tt)

- **Lớp tập hợp (Set):**

```
#include <iostream.h>
class Set {
public:
    Set(void) { num = 100; elems = new int[100]; }
    friend int operator & (const int, Set&); // thành viên ?
    friend int operator == (Set&, Set&); // bang ?
    friend int operator != (Set&, Set&); // không bang ?
    friend Set operator * (Set&, Set&); // giao
    friend Set operator + (Set&, Set&); // hợp
    //...
    void AddElem(const int elem);
    void Copy (Set &set);
    void Print (void);
private:
    int *elems;
    int num;
};
```

// Định nghĩa các toán tử

```
.....
int main (void)
{ Set s1, s2, s3;
  s1.AddElem(10); s1.AddElem(20);
  s1.AddElem(30); s1.AddElem(40);
  s2.AddElem(30); s2.AddElem(50);
  s2.AddElem(10); s2.AddElem(60);
  cout << "s1 = "; s1.Print();
  cout << "s2 = "; s2.Print();
  if (20 & s1) cout << "20 thuộc s1\n";
  cout << "s1 giao s2 = "; (s1 * s2).Print();
  cout << "s1 hợp s2 = "; (s1 + s2).Print();
  if (s1 != s2) cout << "s1 != s2\n";
  return 0;
}
```

NBT

88

Chuyển kiểu

- Muốn thực hiện các phép cộng:

```
void main() {  
    Point p1(10,20), p2(30,40), p3, p4, p5;  
    p3 = p1 + p2;  
    p4 = p1 + 5; p5 = 5 + p1;  
};
```

⇒ Có thể định nghĩa thêm 2 toán tử:

```
class Point {  
    //...  
    friend Point operator + (Point, Point);  
    friend Point operator + (int, Point);  
    friend Point operator + (Point, int);  
};
```

NBT

89

Chuyển kiểu (tt)

- Chuyển đổi kiểu: ngôn ngữ định nghĩa sẵn.

```
void main() {  
    Point p1(10,20), p2(30,40), p3, p4, p5;  
    p3 = p1 + p2;  
    p4 = p1 + 5; // tương đương p1 + Point(5)  
    p5 = 5 + p1; // tương đương Point(5) + p1  
}
```

⇒ Định nghĩa phép chuyển đổi kiểu

```
class Point {  
    //...  
    Point (int x) { Point::x = Point::y = x; }  
    friend Point operator + (Point, Point);  
};
```

Chuyển kiểu
5 ⇔ Point(5)

NBT

90

Chuyển kiểu (tt)

- Chuyển đổi kiểu: ngôn ngữ định nghĩa sẵn.

```
void main() {  
    Point p1(10,20), p2(30,40), p3, p4, p5;  
    p3 = p1 + p2;  
    p4 = p1 + 5; // tương đương p1 + Point(5)  
    p5 = 5 + p1; // tương đương Point(5) + p1  
}
```



Định nghĩa phép chuyển đổi kiểu

```
class Point {  
    //...  
    Point(int x) { Point::x = Point::y = x; }  
    friend Point operator + (Point, Point);  
};
```

Chuyển kiểu
5 ⇔ Point(5)

NBT

91

Ví dụ

```
class Point  
{  
private: int xVal, yVal;  
public:  
    void Print(){cout<<" "<<xVal<<" "<<yVal<<"\n";}   
    Point(int x, int y):xVal(x),yVal(y){}  
    Point(int x){Point::xVal = Point::yVal = x;} // constructor chuyển kiểu  
  
    friend Point operator + (Point, Point);  
};  
  
Point operator + (Point p1, Point p2){  
    return Point(p1.xVal+p2.xVal, p1.yVal+p2.yVal);  
}  
  
void main(){  
    Point p1(3,4), p2(2,5);  
    p1.Print();  
    Point p = p1+3;  
    p.Print();  
}
```

NBT

92

Tái định nghĩa toán tử xuất <<

- Định nghĩa hàm toàn cục:

ostream& **operator** << (ostream&, **Class**&);

```
class Point {  
public:  
    Point (int x=0, int y=0)  
    { Point::x = x; Point::y = y; }  
    friend ostream& operator <<  
        (ostream& os, Point& p)  
        { os<< "(" << p.x << "," << p.y << ")"; }  
    // .....  
private:  
    int x, y;  
};
```

```
void main() {  
    Point p1(10,20), p2;  
    cout<<"Diem P1: "<< p1 << endl;  
    cout<<"Diem P2: "<< p2 << endl;  
}
```

Kết quả
trên
màn hình ?

NBT

93

Tái định nghĩa toán tử nhập >>

- Định nghĩa hàm toàn cục:

istream& **operator** >> (istream&, **Class**&);

```
class Point {  
public:  
    Point (int x=0, int y=0)  
    { Point::x = x; Point::y = y; }  
    friend istream& operator >>  
        (istream& is, Point& p)  
        { cout<<"Nhập x: "; is>>p.x;  
          cout<<"Nhập y: "; is>>p.y;  
          }  
    // .....  
private:  
    int x, y;  
};
```

```
void main() {  
    Point p1, p2;  
    cout<<"Nhập thông tin cho P1: \n";  
    cin>>p1;  
    cout<<"Nhập thông tin cho P2: \n";  
    cin>>p2;  
}
```

NBT

94

Tái định nghĩa toán tử []

- Thông thường để xuất ra giá trị của 1 phần tử tại vị trí cho trước trong đối tượng.
- Định nghĩa là hàm thành viên.

```
class StringVec {  
public:  
    StringVec (const int dim);  
    ~StringVec ();  
    char* operator [] (int);  
    int add(char* );  
    // .....  
private:  
    char **elems; // các phần tử  
    int dim; // kích thước của vectơ  
    int used; // vị trí hiện tại  
};
```

```
char* StringVec::operator [] (int i) {  
    if ( i >= 0 && i < used) return elems[i];  
    return "";  
}  
  
void main() {  
    StringVec sv1(100);  
    sv1.add("PTPhi");sv1.add("BQThai");  
    sv1.add("LVLam"); sv1.add("NCHuy");  
    cout<< sv1[2]<<endl;  
    cout<<sv1[0];  
}
```

NBT

95

Tái định nghĩa toán tử ()

- Định nghĩa là hàm thành viên.

```
class Matrix {  
public:  
    Matrix (const short rows, const short cols);  
    ~Matrix (void) {delete elems;}  
    double& operator () (const short row,  
                        const short col);  
    friend ostream& operator << (ostream&, Matrix&);  
    friend Matrix operator + (Matrix&, Matrix&);  
    friend Matrix operator - (Matrix&, Matrix&);  
    friend Matrix operator * (Matrix&, Matrix&);  
private:  
    const short rows; // số hàng  
    const short cols; // số cột  
    double *elems; // các phần tử  
};
```

```
double& Matrix::operator ()  
(const short row, const short col)  
{  
    static double dummy = 0.0;  
    return (row >= 1 && row <= rows  
            && col >= 1 && col <= cols)  
        ? elems[(row - 1)*cols  
                + (col - 1)]  
        : dummy;  
}  
  
void main() {  
    Matrix m(3,2);  
    m(1,1) = 10; m(1,2) = 20;  
    m(2,1) = 30; m(2,2) = 40;  
    m(3,1) = 50; m(3,2) = 60;  
    cout<<m<<endl;  
}
```

NBT

96

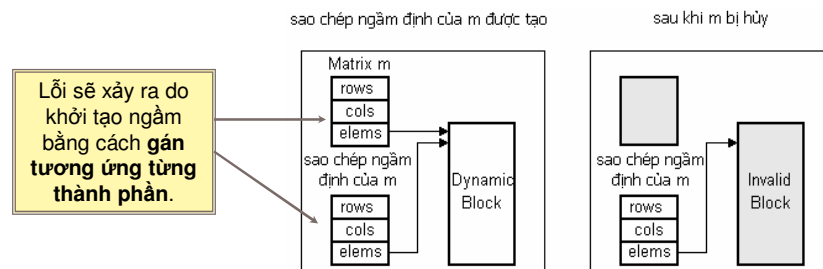
Khởi tạo ngầm định

- Được định nghĩa sẵn trong ngôn ngữ:

VD: Point p1(10,20); Point p2 = p1;

- Sẽ gây ra lỗi (kết quả SAI) khi bên trong đối tượng có *thành phần dữ liệu là con trỏ*.

VD: Matrix m(5,6); Matrix n = m;



NBT

97

Khởi tạo ngầm định (tt)



Khi lớp có *thành phần dữ liệu con trỏ*, phải định nghĩa hàm **khởi tạo sao chép**

```
class Point {
    int x, y;
public:
    Point (int =0; int =0 );
    // Không cần thiết DN
    Point (const Point& p) {
        x = p.x;
        y = p.y;
    }
    // .....
};
// .....
```

```
class Matrix {
    //....
    Matrix(const Matrix&);
};
Matrix::Matrix (const Matrix &m)
    : rows(m.rows), cols(m.cols)
{
    int n = rows * cols;
    elems = new double[n]; // cùng kích thước
    for (register i = 0; i < n; ++i) // sao chép phần tử
        elems[i] = m.elems[i];
}
```

NBT

98

Gán ngầm định

- Được định nghĩa sẵn trong ngôn ngữ:
 - Gán tương ứng từng thành phần.
 - **Đúng** khi đối tượng không có dữ liệu con trỏ.
- Khi thành phần dữ liệu có con trỏ, bắt buộc phải định nghĩa phép gán = cho lớp.

Hàm thành viên

```
class Matrix {  
    //...  
    Matrix& operator = (const Matrix &m) {  
        if (rows == m.rows && cols == m.cols) { // phải khớp  
            int n = rows * cols;  
            for (int i = 0; i < n; ++i) // sao chép các phần tử  
                elems[i] = m.elems[i];  
        }  
        return *this;  
    }  
};
```

NBT

99

Tái định nghĩa toán tử ++ & --

- Toán tử ++ (hoặc toán tử --) có 2 loại:
 - Tiền tố: ++n
 - Hậu tố: n++

```
class PhanSo {  
    int tuso, mau so;  
public:  
    // .....  
    PhanSo(int=0, int=1);  
    friend PhanSo operator ++ (PhanSo&);  
    friend PhanSo operator ++ (PhanSo&, int);  
};  
PhanSo operator ++ (PhanSo& p) {  
    return (p = PhanSo(tuso+mauso, mauso));  
}  
PhanSo operator ++ (PhanSo& p, int x) {  
    PhanSo p2 = PhanSo(tuso+mauso, mauso);  
    return p2;  
}
```

```
void main() {  
    PhanSo p1(3,4), p2;  
    cout<< p1++;  
    cout<< ++p2;  
    cout<< ++(p1++) + (++p2)++;  
}
```

Kết quả trên màn hình ?

NBT

100

Tái định nghĩa new & delete

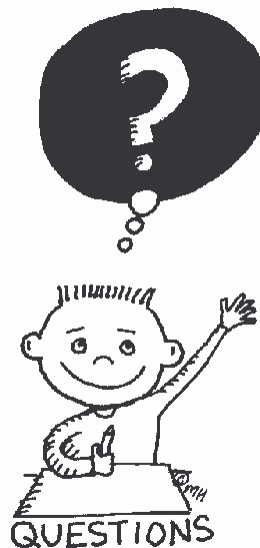
- Hàm **new** và **delete** mặc định của ngôn ngữ:
 - Nếu đối tượng kích thước nhỏ, có thể sẽ gây ra quá nhiều khối nhỏ => chậm.
 - Không đáng kể khi đối tượng có kích thước lớn.=> Toán tử new và delete ít được tái định nghĩa.
- Định nghĩa theo dạng hàm thành viên:

```
class Point {  
public:  
    //...  
    void* operator new (size_t bytes);  
    void operator delete (void *ptr, size_t bytes);  
private:  
    int xVal, yVal;  
};
```

```
void main() {  
    Point *p = new Point(10,20);  
    Point *ds = new Point[30];  
    //.....  
    delete p;  
    delete []ds;  
}
```

NBT

101



NBT

102

EXCEPTION

NBT

103

Nội dung

- Bẫy và bắt lỗi
- Cơ chế bắt lỗi
- Dừng liên kết động xử lý lỗi

NBT

104

Ví dụ

```
float GiaTri(int tu, int mau){  
    return (float)tu/mau;  
}  
void main(){  
    int ts,ms;  
    cout<<"Nhap tu so: "; cin>>ts;  
    cout<<"Nhap mau so:"; cin>>ms;  
    cout<<"gia tri cua ps:"<<GiaTri(ts,ms);  
}
```

Lỗi chia cho 0

NBT

105

Giải pháp

```
class XLLOI{};  
  
float GiaTri(int tu, int mau){  
    if(mau==0) throw XLLOI();  
    return (float)tu/mau;  
}  
void main(){  
    int ts,ms;  
    cout<<"Nhap tu so: "; cin>>ts;  
    cout<<"Nhap mau so:"; cin>>ms;  
    try{  
        cout<<"gia tri cua ps:"<<GiaTri(ts,ms);  
    }catch(XLLOI)  
    {  
        cout<<"Loi chia 0";  
    }  
}
```

NBT

106

Xử lý nhiều lỗi

```
class A{ };
class B{ }

void SinhLoi(int a){
    if(a>0) throw A();
    throw B();
}

void main(){
    int n;
    cout<<"Nhập số nguyên: "; cin>>n;

    try{
        SinhLoi(n);
    }catch(A){
        cout<<"Lỗi A";
    }catch(B){
        cout<<"Lỗi B";
    }
}
```

```
class A{ };
class B{ }

void SinhLoi(int a){
    if(a>0) throw A();
    throw B();
}

void main(){
    int n;
    cout<<"Nhập số nguyên: "; cin>>n;

    try{
        SinhLoi(n);
    }catch(...){ // bắt tất cả các lỗi
        cout<<"Xử lý lỗi";
    }
}
```

NBT

107

Xử lý nhiều lỗi

```
class A{ };
class B{ }

void SinhLoi(int a){
    if(a>0) throw A();
    throw B();
}

void main(){
    int n;
    cout<<"Nhập số nguyên: "; cin>>n;

    try{
        SinhLoi(n);
    }catch(...){
        cout<<"Lỗi";
    }catch(B){
        cout<<"Lỗi B";
    }
}
```



```
class A{ };
class B{ }

void SinhLoi(int a){
    if(a>0) throw A();
    throw B();
}

void main(){
    int n;
    cout<<"Nhập số nguyên: "; cin>>n;

    try{
        SinhLoi(n);
    } catch(B){
        cout<<"Lỗi B";
    }catch(...){ // bắt tất cả các lỗi
        cout<<"Xử lý lỗi";
    }
}
```

NBT

108

Ví dụ

```
#include <iostream.h>
class Loi_Chia_0{
public:
    int ts;
    Loi_Chia_0(int t): ts(t) {}
};
float GiaTriPS(int ts, int ms){
    if( ms==0) throw(Loi_Chia_0(ts) );

    return float(ts)/ms;
}
```

```
void main(){
    int ts, ms;
    cout << "Tính gia tri phan so\n";
    cout << "TS = "; cin >> ts;
    cout << "MS = "; cin >> ms;
    try {
        float gt = GiaTriPS(ts, ms);
        cout << "Gia tri PS la: " << gt;
    }
    catch ( Loi_Chia_0 loi )
    {
        cout << "Loi chia " << loi.ts <<
            " cho 0";
    }
}
```

NBT

109

Cơ chế bắt lỗi

- Khi 1 lỗi bị bắt thì chương trình vẫn tiếp tục và sẽ xử lý lỗi theo mã lệnh được bắt.
- Trong xly, ta có thể duy trì lỗi bằng từ khoá throw

NBT

110

Ví dụ

```
#include <iostream.h>
class Loi {};
void PhatLoi(){
    throw ( Loi );
}
void BatLoi(){
    try {
        PhatLoi();
    }
    catch ( Loi ) {
        cout << "Loi bi bat lần 1";
        throw; // duy trì lỗi
    }
}
```

```
void main() {
    try {
        BatLoi();
    }
    catch ( Loi ) {
        cout << "Loi bi bat lan hai";
    }
}
```

NBT

111

Dùng liên kết động xử lý lỗi

```
#include <iostream.h>
#include <stdlib.h>
class Loi{
public:
    virtual void InLoi();
};
class Loi_KT : public Loi {
public:
    int num;
    Loi_KT(int n): num(n) {}
    void InLoi() {
        cout << "Loi khoi tao voi " << num << "phan tu\n";
    }
};
```

NBT

112

Dùng liên kết động xử lý lỗi

```
// loi truy cap
class Loi_TC : public Loi {
public:
    int cs;
    Loi_TC(int i): cs(i) {}
    void InLoi() {
        cout << "Loi truy cap chi so " << cs << "\n";
    }
};
```

NBT

113

Dùng liên kết động xử lý lỗi

```
class Array
{
    int num;
    int *elems;
public:
    Array(int n): num(n) {
        if ( num <= 0 ) throw( Loi_KT(num) );
        elems = new int[num];
    }
    ~Array() { delete elems; }
    int phantu(int i){
        if ( i<0 || i>=num ) throw( Loi_TC(i) );
        return elems[i];
    }
};
```

NBT

114

Dùng liên kết động xử lý lỗi

```
void main(){  
    try {  
        Array a(-3);  
        a[5] = 10;  
    }  
    catch ( Loi& l ) {  
        l.InLoi();  
    }  
}
```

NBT

115

Input / Output Stream

- Giới thiệu
- Khái niệm
- Kiến trúc dòng nhập xuất
- Dòng nhập
- Dòng xuất
- Nhập xuất với tập tin

NBT

116

Giới thiệu

■ Xét ví dụ

cout<<n;

- Chỉ thị này gọi đến toán tử "<<" và cung cấp cho nó hai toán hạng, một là "luồng xuất - output stream"(**cout**), hai là biểu thức mà ta muốn xuất (**n**).

cin>>x

- Chỉ thị này gọi tới toán tử ">>" và cung cấp cho nó 2 toán hạng, một là "luồng nhập - input stream"(**cin**), hai là biến mà ta lưu giá trị.

NBT

117

Khái niệm luồng?

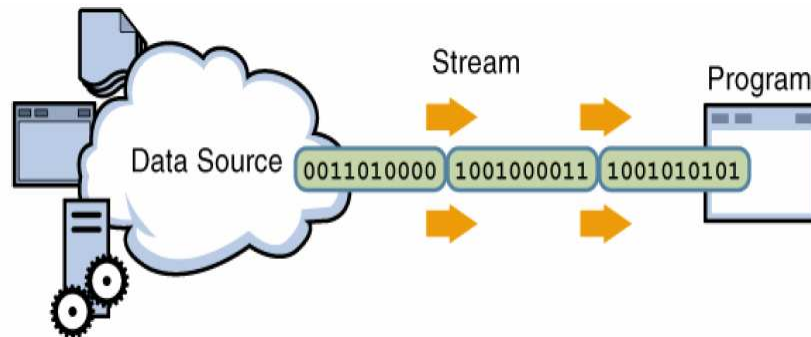
- Luồng là một "dòng chảy" của dữ liệu được gắn với các thiết bị vào ra.
- Hai loại luồng:
 - Luồng nhập: Gắn với các thiết bị nhập như bàn phím, máy scan, file...
 - Luồng xuất: Gắn với các thiết bị xuất như màn hình, máy in, file...

NBT

118

Luồng nhập - Input Stream

- Chương trình sử dụng input stream để đọc dữ liệu từ nguồn.

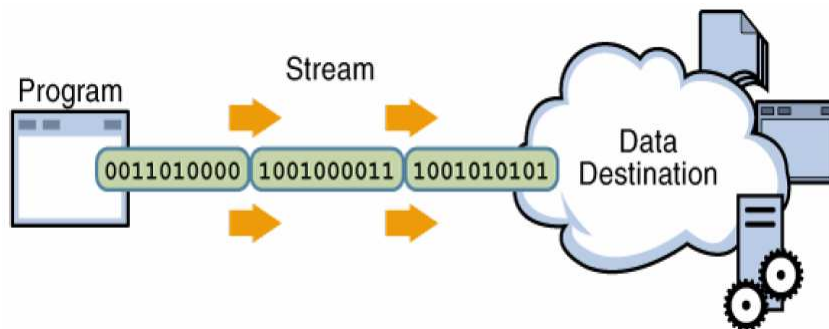


NBT

119

Luồng xuất - Output Stream

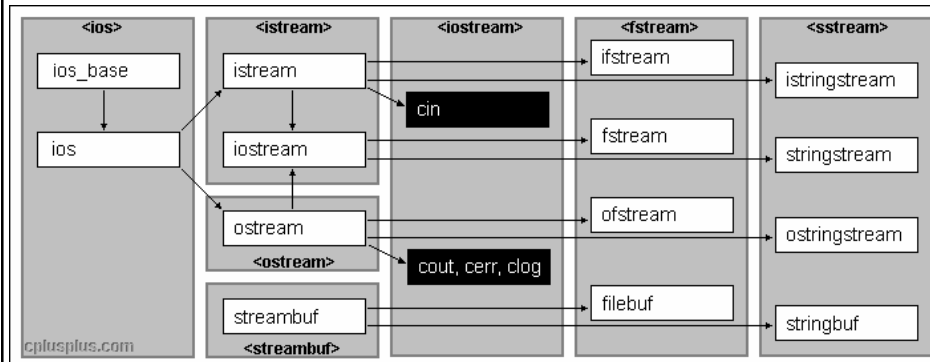
- Chương trình sử dụng output stream để ghi dữ liệu xuống đích.



NBT

120

Kiến trúc dòng nhập xuất



NBT

121

OSTREAM

- **Toán tử <<**
`ostream &operator<<(expression)`
 Ví dụ: `cout<<5;`
- **Hàm put:** xuất ra một tham số ký tự.
`cout.put(c);`
 Vì hàm `put` trả về một đối tượng là `ostream` nên nó có thể được viết liên tục như sau: **`cout.put(c1).put(c2).put(c3);`** để xuất 3 ký tự `c1, c2, c3`
- **Hàm write:** xuất ra luồng 1 chuỗi ký tự với chiều dài đã cho.
 Hàm trả về 1 đối tượng `ostream`
`char t[] = "hello";`
`cout.write(t, 4);`

NBT

122

OSTREAM - Định dạng

- Chọn cơ số thể hiện: cơ số ngầm định là 10

```
#include <iostream.h>
#include <conio.h>
void main() {
    clrscr();
    int n = 12000;
    cout<<"Ngam dinh " <<n<<endl;
    cout<<"Dui he 16 "<<hex<<n<<endl;
    cout<<"Dui he 10 "<<dec<<n<<endl;
    cout<<"Dui he 8 "<<oct<<n<<endl;
    cout<<"va ... " <<n<<endl;
    getch();
}
```

```
Ngam dinh 12000
Dui he 16 2ee0
Dui he 10 12000
Dui he 8 27340
va ... 27340
```

NBT

123

OSTREAM - Định dạng

- width: đặt độ rộng
- precision: xác định chữ số thập phân được in
- fill: chèn ký tự đệm

```
int x = 10;
cout.fill('0');
cout.width(5);
cout<<x;
```

```
00010
```

```
float pi=3.1415927f;
int orig_prec = cout.precision(2);
cout<<pi<<endl;
cout.precision(orig_prec);
cout<<pi;
```

```
3.1
3.14159
```

<http://www.cplusplus.com/reference/iostream/ostream/>

NBT

124

ISTREAM

- Toán tử >>
istream & operator >>(&base_type)
ví dụ: cin>>n;
- Hàm get: lấy một ký tự từ luồng nhập
istream & get(char &);

```
char c;
while (cin.get(c)){ //lấy các ký tự từ luồng nhập vào c
    cout.put(c);    // đưa c vào luồng xuất
    if(c=='\n') break;
}
```

NBT

125

ISTREAM

- Hàm getline: đọc các ký tự và đưa vào ch
*istream & getline(char * ch, int size, char delim='\n')*
 - khi gặp ký tự phân cách delim thì hàm ngắt
 - đọc đủ kích thước size thì hàm ngắt
- Hàm gcount ????
- Hàm read: đọc một dãy ký tự với chiều dài
char t[10];
cin.read(t,5);
- Một số hàm khác
putback(char c) : trả lại luồng nhập 1 ký tự
peek() : đưa ra ký tự kế tiếp, nhưng không lấy ra khỏi luồng

NBT

126

ISTREAM

```
// istream putback
#include <iostream>
using namespace std;
int main () {
    char c;
    int n;
    char str[256];
    cout << "Enter a number or a word: ";
    c = cin.get();
    if ( (c >= '0') && (c <= '9') ) {
        cin.putback(c);
        cin >> n;
        cout << "You have entered number "
              << n << endl;
    } else {
        cin.putback(c);
        cin >> str;
        cout << " You have entered word " <<
              str << endl;
    }
    return 0;
}
```

```
// istream putback
#include <iostream>
using namespace std;
int main () {
    char c;
    int n;
    char str[256];
    cout << "Enter a number or a word: ";
    c = cin.peek();
    if ( (c >= '0') && (c <= '9') ) {
        cin >> n;
        cout << "You have entered number "
              << n << endl;
    } else {
        cin >> str;
        cout << " You have entered word " <<
              str << endl;
    }
    return 0;
}
```

<http://www.cplusplus.com/reference/istream/istream/>

NBT

127

Xuất ra tập tin- ofstream

- Kế thừa ostream và được định nghĩa trong fstream.h
- Hàm constructor
*ofstream (const char * filename, ios_base::openmode mode =
ios_base::out);*
 - *filename*
 - chuỗi chứa tên tập tin cần mở
 - *mode*
 - app:
 - ate: thiết lập vị trí ở cuối luồng khi mở file
 - binary:
 - in:
 - out:
 - trunc: Nếu file đã tồn tại thì ghi lên file ghi xóa nd cũ, ghi nd mới
- ví dụ:
ofstream output("abc.txt",ios::out);

NBT

128

Các hàm thành viên của ofstream

- is_open
- open
- close
- operator<<
- put
- write
- flush
- eof
-

NBT

129

```
#include <stdlib.h>
#include <iostream.h>
#include <fstream.h>
#include <iomanip.h>
#include <conio.h>
void main() {
    char filename[100];
    int n;
    cout<<"Ten tap tin : "; cin>>filename;
    ofstream output(filename,ios::out);
    if (!output) {
        cout<<"Khong the tao duoc tap tin\n";
        exit(1);
    }
    do {
        cin >>n;
        if (n>0) output<<n<<' ';
    }while(n>0);
    output<<endl;
    output.close();
}
```

NBT

130

Nhập từ tập tin - ifstream

- Kế thừa từ istream và được định nghĩa trong fstream.h
- Hàm constructor

```
ifstream ( const char * filename, ios_base::openmode mode =  
ios_base::in );
```

- *filename*
 - chuỗi chứa tên tập tin cần mở
- *mode*
 - app:
 - ate:
 - binary:
 - in:
 - out:
 - trunc:

NBT

131

Các hàm thành viên

- is_open
- open
- close
- operator>>
- get
- getline
- read
- eof
-

NBT

132

```

#include <stdlib.h>
#include <iostream.h>
#include <fstream.h>
#include <iomanip.h>
void main() {
    char filename[100];
    int n;
    cout<<"Ten tap tin : "; cin>>filename;
    ifstream input(filename,ios::in);
    if (!input) {
        cout<<"Khong the mo duoc tap tin\n";
        exit(1);
    }
    while(input) {
        input>>n;
        cout<<n<<endl;
    }
    input.close();
}

```

NBT

133

Khuôn hình(Template)

- Nội dung:
 - Khuôn hình hàm
 - Khuôn hình lớp
 - Cụ thể khuôn hình hàm
 - Cụ thể khuôn hình lớp
 - Danh sách tham số kiểu
 - Danh sách tham số biểu thức
 - Khai báo friend cho hàm và lớp trong khuôn hình.

NBT


134

Khuôn hình(Template)

Ví dụ

```
int min (int a, int b) {  
    if (a < b) return a;  
    else return b;  
}
```

```
float min (float a, float b) {  
    if (a < b) return a;  
    else return b;  
}
```



Giải pháp 1 hàm
cho nhiều loại
dữ liệu?

NBT

135

Khung hình hàm

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

```
template <class T> T min(T a, T b) {  
    if (a < b) return a;  
    else return b;  
}
```

```
void main() {  
    int n = 4, p = 12;  
    float x = 2.5, y = 3.25;  
    cout<<"min (n, p) = "<<min (n, p)<<"\n"; //int min(int, int)  
    cout<<"min (x, y) = "<<min (x, y)<<"\n"; //float min(float, float)  
    getch();  
}
```

NBT

136

Xét ví dụ sau

```
void main() {  
    char * adr1 = "DHHS";  
    char * adr2 = "CDHS";  
    cout << "min (adr1, adr2) ="<<min (adr1,adr2);  
}
```

min (adr1, adr2) = DHHS



Tại sao ?

NBT

137

Giải quyết

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

min (adr1, adr2) = CDHS

```
template <class T> T min(T a, T b) {  
    if (a < b) return a;  
    else return b;  
}
```

```
char * min (char *cha, char *chb) {  
    if (strcmp(cha, chb) < 0) return cha;  
    else return chb;  
}
```

Cụ thể một khuôn hình hàm

```
void main() {  
    char * adr1 = "DHHS";  
    char * adr2 = "CDHS";  
    cout << "min (adr1, adr2) ="<<min (adr1,adr2);  
}
```

```

#include <iostream.h>
template <class T> T min( T a, T b) {
    if (a < b) return a;
    else return b;
}

class PhanSo {
    int tu, mau;
public:
    PhanSo(int t=0, int m=1) { tu = t; mau = m;}
    void Print() { cout <<tu<<"/"<<mau<<endl; }
    friend int operator < (PhanSo , PhanSo);
};
int operator < (PhanSo a, PhanSo b) {
    return a.tu*b.mau < a.mau*b.tu;
}

void main() {
    PhanSo p1(3,4), p2(4, 5);
    cout<<"min (p1, p2) = ";
    min(p1, p2).Print();
}

```

NBT

139

Khai báo nhiều template

```

template <class T, class U> int Func (T a, T b, U c) {...}

```

```

#include <iostream.h>
#include <conio.h>
template <class T, class U> T Func(T x, U y, T z) {
    return x + y + z;
}
void main() {
    int n= 1, p = 2, q = 3;
    float x =2.5, y = 5.0;
    cout <<Func ( n, x, p)<<"\n"; // (int) 5
    cout <<Func (x, n, y)<<"\n"; // (float)8.5
    cout <<Func (n, p, q)<<"\n"; // (int) 6
}

```

NBT

140

Nguyên tắc thực hiện template

Xét lại ví dụ sau:

```
template <class T> T min(T a, T b) {  
    if (a < b) return a;  
    else return b;  
}
```

với khai báo:

int n = 9;

char c= 'A';

thì kết quả của min(n,c) là ?

Lỗi

NBT

141

Nguyên tắc thực hiện template

- (1) Đầu tiên xét các hàm thông thường. Nếu có sự tương ứng chính xác thì hàm thông thường được chọn. Nếu không ta xét hàm khuôn mẫu.
- (2) Xét hàm khuôn mẫu nếu có sự tương ứng chính xác giữa kiểu của tham số hình thức và kiểu của tham số thực được truyền vào hàm, thì hàm được sản sinh.
- (3) Nếu có nhiều hàm tương ứng → lỗi
- (4) Không cho phép chuyển kiểu thông thường như T thành const T, T[] thành T*. Những trường hợp này hoàn toàn được phép trong định nghĩa chồng hàm

NBT

142

Khuôn hình lớp

■ Ví dụ:

```
class point {  
    int x, y;  
    public:  
        point (int abs =0, int ord =0);  
        void display();  
        //...  
};
```

- Lớp point thể hiện tọa độ số thực, số nguyên. !?

NBT

143

Hàm thành phần được định nghĩa bên trong định nghĩa lớp

```
#include <iostream.h>  
#include <conio.h>  
template <class T> class point {  
    T x, y;  
    public:  
        point(T abs = 0, T ord = 0) {  
            x = abs; y = ord;  
        }  
        void display() {  
            cout<<" "<<x<<" "<<y<<"\n";  
        }  
};
```

```
void main() {  
    clrscr();  
    point<int> ai(3,5);  
    ai.display();  
    point<char> ac('d','y');  
    ac.display();  
    point<double> ad(3.5, 2.3);  
    ad.display();  
    getch();  
}
```

(3,5)
(d,y)
(3.5,2.3)

NBT

144

Hàm thành phần được định nghĩa bên ngoài định nghĩa lớp

```
#include <iostream.h>
#include <conio.h>
template <class T> class point {
    T x, y;
public:
    point(T abs = 0, T ord = 0) {
        x = abs; y = ord;
    }
    void display();
}
```

```
template <class T> void point<T>::display() {
    cout<<"Toa do: "<<x<<" "<<y<<"\n";
}
void main() {
    clrscr();
    point<int> ai(3,5);
    ai.display();
    point<char> ac('d','y'); ac.display();
    point<double> ad(3.5, 2.3); ad.display();
    getch();
}
```

NBT

145

Danh sách tham số kiểu

```
template <class T, class U, class V> //ds các tham số kiểu
class try {
    T x;
    U t[5];
    ...
    V function (int, U);
    ...
};
```

```
try <int, float, int> // lớp thể hiện với 3 tham số int, float, int
try <int, int *, double> // lớp thể hiện với 3 tham số int, int *, double
try <float, point<int>, double>
try <point<int>, point<float>, char *>
```

NBT

146

Danh sách tham số biểu thức

```
#include <iostream.h>
#include <conio.h>
template <class T, int n> class table
{
    T data[n];
public:
    table() {
        cout<<"Tao bang\n";
        for(int i=0; i<n; i++)
            data[i] = rand()%10;
    }
    T & operator[](int i){
        return data[i];
    }
};
```

```
class point {
    int x, y;
public:
    point (int abs = 1, int ord = 1) {
        x = abs; y = ord;
    }
    void display() {
        cout<<"<<x<<","<<y<<"\n";
    }
};
```

NBT

147

Danh sách tham số biểu thức

```
void main() {
    table<int, 4> ti;
    for(int i = 0; i < 4; i++)        ti[i] = i;

    cout<<"ti: ";
    for(i = 0; i < 4; i++) cout <<ti[i]<<" ";
    cout<<"\n";

    table <point, 3> tp;
    for(i = 0; i < 3; i++) tp[i].display();
}
```

Hằng số

NBT

148

Cụ thể hoá hàm thành phần của 1 lớp

```
#include <iostream.h>
#include <conio.h>
template <class T> class point {
    T x, y;
public:
    point(T abs = 0, T ord = 0) {
        x = abs; y = ord;
    }
    void display();
};
template <class T> void point<T>::display() {
    cout<<"Toa do: "<<x<<" "<<y<<"\n";
}
void point<char>::display() {
    cout<<"Toa do: "<<(int)x<<" "<<(int)y<<"\n";
}
```

```
void main() {
    point <int> ai(3,5);
    ai.display();
    point <char> ac('d','y');
    ac.display();
    point <double> ad(3.5, 2.3);
    ad.display();
}
```

```
Toa do: 3 5
Toa do: 100 121
Toa do: 3.5 2.3
```

NBT

149

Cụ thể hoá một lớp

```
class point<char> {
    T x, y;
public:
    point(T abs = 0, T ord = 0) {
        x = abs; y = ord;
    }
    void display();
};
void point<char>::display() {
    cout<<"Toa do: "<<(int)x<<" "<<(int)y<<"\n";
}
```

NBT

150

Lệnh gán giữa 2 đối tượng

```
table <int, 12> t1;  
table <float, 12> t2;  
ta không được viết:  
    t2 = t1; //không tương thích hai tham số đầu
```

```
table <int, 12> ta;  
table <int, 20> tb;  
ta không được viết:  
    ta = tb; //không tương thích hai tham số sau
```

Nếu ta định nghĩa toán tử gán trong table thì phép gán trên được thực hiện bình thường

NBT

151

Khai báo **friend** trong khuôn hình hàm và khuôn hình lớp

```
template <class T> class Example {  
    int x; public:  
    friend class point;  
    friend int function(float);  
    ...  
};
```

Lớp **point** và hàm **function** thông thường là bạn của lớp Example

NBT

152

Khai báo **friend** trong khuôn hình hàm và khuôn hình lớp

```
template <class T> class point {...};  
template <class T> int function (T) {...};
```

```
template <class T> class Example {  
    int x; public:  
    friend class point<int>;  
    friend int function(double);  
    ...  
};
```

Lớp **point** và hàm **function** cụ thể là bạn của lớp Example

NBT

153

Khai báo **friend** trong khuôn hình hàm và khuôn hình lớp

```
template <class T> class point {...};  
template <class T> int function (T) {...};
```

```
template <class T, class U> class Example {  
    int x; public:  
    friend class point<T>;  
    friend int function(U);  
    ...  
};
```

Lớp **point** và hàm **function** không xác định thể hiện là bạn của lớp **Example**. Các thể hiện cụ thể sẽ được xác định khi tạo ra thể hiện **Example**.

NBT

154

Khai báo **friend** trong khuôn hình hàm và khuôn hình lớp

```
template <class T> class point {...};  
template <class T> int function (T) {...};
```

```
template <class T, class U> class Example {  
    int x; public:  
    template <class X> friend class point<X>;  
    template <class X> friend int function(X);  
    ...  
};
```

Tất cả các thể hiện của lớp **point** và hàm **function** là bạn của lớp **Example**.

NBT

155

Ví dụ về quản lý mảng 2 chiều

```
#include <iostream.h>  
#include <conio.h>  
template <class T, int n> class table {  
    T arr[n];  
    int limit;  
    public:  
        table (int init = 0);  
        T & operator[] (int i) {  
            if (i < 0 || i > limit)  
                cout<<"Tran chi so "<<i<<"\n";  
            else return arr[i];  
        }  
};
```

NBT

156

Ví dụ về quản lý mảng 2 chiều

```
template <class T, int n> table<T,n>::table(int init = 0) {
    int i;
    for (i = 0; i < n; i++) arr[i] = init;
    limit = n - 1;
    cout<<"Tao bang kích thước "<<n<<" init = "<<init<<"\n";
}

void main() {
    table <table<int,3>,2>ti; //khởi tạo ngẫu nhiên
    table <table<float,4>,2> td(10); //khởi tạo bằng 10
    ti[1][6] = 15;
    ti[8][-1] = 20;
    cout<<ti[1][2]<<"\n"; cout<<td[1][0]<<"\n";
}
```

NBT

157

Bài tập

- Viết chương trình khai báo khuôn hình để mô phỏng hoạt động của stack và queue cho các kiểu dữ liệu khác nhau.

NBT

158

- Tham khảo từ nhiều nguồn khác nhau.