

Lập trình hướng đối tượng và C++

Bài 6: Hàm tạo và hàm hủy

TS. Nguyễn Hiếu Cường

Bộ môn CNPM, Khoa CNTT, Trường Đại học GTVT

Email: cuonggt@gmail.com

Nội dung chính

1. Giới thiệu môn học
2. Các khái niệm cơ bản
3. Hàm trong C++
4. Lớp và đối tượng
5. Định nghĩa chồng toán tử
- 6. Hàm tạo và hàm huỷ**
7. Dẫn xuất và thừa kế
8. Tương ứng bội
9. Khuôn hình (templates)

Ví dụ

```
class DT
{
    int n;
    float *a;    // co con tro
public:
    DT() { }
    DT(int n1);
    ~DT() { delete a; }
    void hien();
    DT& operator=(DT& d); // Khai bao
};

DT::DT(int n1)
{
    n= n1;
    a= new float[n+1];
    for (int i=0; i<=n; ++i)
    {
        cout<<"a["<<i<<"]= ";
        cin>>a[i];
    }
}
```

```
void DT::hien() {
    for (int i=0; i<n; ++i)
        cout<<a[i]<<" ";
}

DT& DT::operator=(DT& d)
{
    n= d.n;
    //a= new float[n+1];
    for (int i=0; i<=n; ++i)
        a[i]= d.a[i];
    return (*this);
}

void main()
{
    DT y(3);
    cout<<endl<<"y:"<<endl; y.hien();
    DT x;
    x = y; // Su dung toan tu gan
    cout<< endl <<"x:"<< endl;
    x.hien();
}
```

Hàm tạo (Constructor)

- Tác dụng của hàm tạo là gì?
- Đối tượng được tạo bằng cách nào?
- Nếu lớp không xây dựng hàm tạo thì sao?

Cách xây dựng hàm tạo

- Tương tự như phương thức thông thường của lớp
- Khác phương thức thông thường:
 - Tên của hàm tạo: phải trùng với tên lớp
 - Kiểu của hàm tạo: không có
 - Đối của hàm tạo: có thể có hoặc không
 - Trong một lớp có thể có nhiều hàm tạo
 - Tùy theo cách khai báo khi tạo đối tượng mà hàm tạo tương ứng sẽ được gọi

Ví dụ (hàm tạo)

```
class DIEM_DH
{
private:
    int x, y, m ;
public:
    DIEM_DH() { }
    DIEM_DH(int x1, int y1, int m1=15) {
        x= x1; y= y1; m= m1;
    }
} ;

...

DIEM_DH d;                //
DIEM_DH u(200,100,4);      //  u.x=200, u.y=100, u.m=4
DIEM_DH v(300,250);        //  v.x=300, v.y=250, v.m=15
DIEM_DH p[10];             //  Gọi hàm tạo không đối 10 lần
```

Ví dụ

```
class DIEM_DH
{
private:
    int x, y, m ;
public:
    DIEM_DH() ;
    DIEM_DH(int x1, int y1, int m1=15) ;
} ;

...
DIEM_DH *q = new DIEM_DH(50,40,6);    // q->x=50, q->y=40, q->m=6
DIEM_DH *r = new DIEM_DH ;           // r->x=0, r->y= 0, r->m=1
int n=20;
DIEM_DH *s = new DIEM_DH[n];          // Gọi hàm tạo 20 lần
```

Ví dụ (hàm tạo có đối)

```
class DIEM_DH
{
    int x,y,m;
public:
    void in() { cout <<"\n " << x << " " << y << " " << m ; }
    DIEM_DH(int x1,int y1,int m1)
    {
        x=x1; y=y1; m=m1;
    }
};

void main()
{
    DIEM_DH d1(200,200,10);    // Gọi hàm tạo có đối
    DIEM_DH d2;                // Gọi hàm tạo không đối, lỗi!
    d2= DIEM_DH(300,300,8);    // Gọi hàm tạo có đối
    d1.in();
    d2.in();
}
```

Khắc phục lỗi này?

Ví dụ (lớp có nhiều hàm tạo)

```
class DIEM_DH
{
    int x,y,m;
public:
    void in() { cout <<"\n " << x << " " << y << " " << m ; }
    DIEM_DH() { }          // Ham tao khong doi
    DIEM_DH(int x1,int y1,int m1)
    {
        x=x1; y=y1; m=m1;
    }
};

void main()
{
    DIEM_DH d1(200,200,10);
    DIEM_DH d2;              // OK!
    d2= DIEM_DH(300,300,8);
    d1.in();
    d2.in();
}
```

Hàm hủy (Destructor)

- Tác dụng của hàm hủy là gì?
 - Tự động thực hiện khi một đối tượng bị hủy
 - Làm các thao tác “dọn dẹp” sau khi đối tượng bị hủy
- Đối tượng bị hủy bằng cách nào?
 - Tĩnh : Khi kết thúc phạm vi của biến đối tượng
 - Động : Dùng toán tử `delete`
- Nếu lớp không xây dựng hàm hủy thì sao?
 - Khi không xây dựng hàm tạo thì hàm hủy mặc định sẽ được sử dụng
 - Mọi lớp đều có hàm hủy mặc định (default destructor)

Cách xây dựng hàm hủy

- Tên của hàm hủy: trùng tên với lớp và được đặt sau dấu ~
- Kiểu của hàm hủy: không có
- Đối của hàm hủy: không có
 - Trong mỗi lớp có duy nhất 1 hàm hủy!
- Khi nào cần xây dựng hàm hủy?

Ví dụ

```
class DT
{
    int n;
    float *a;    // co con tro nen phai dinh nghia toan tu gan
public:
    DT(int n1); // Constructor
    ~DT() { delete a; } // Destructor
};

DT::DT(int n1)
{
    n= n1;
    a= new float[n+1];
    for (int i=0; i<=n; ++i)
    {
        cout<<"a["<<i<<"]= ";
        cin>>a[i];
    }
}
```

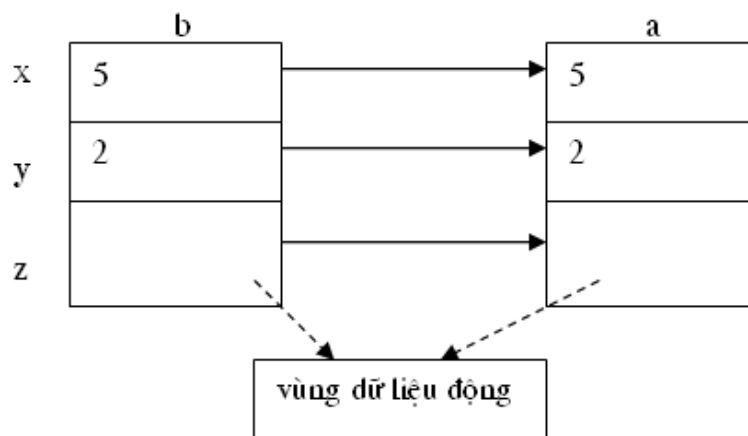
Hàm tạo sao chép (copy constructor)

- Tác dụng của hàm tạo sao chép
 - Tạo một đối tượng mới giống hệt một đối tượng đã có
 - Mỗi lớp đều có hàm tạo sao chép mặc định
 - Tạo một đối tượng đích
 - Sao chép từng bit của đối tượng nguồn sang đối tượng đích
- Khi nào cần định nghĩa hàm tạo sao chép?
 - Khi trong các thành phần dữ liệu của lớp có con trỏ hoặc tham chiếu

```
class C {  
    int x,y;  
    int *z;  
    ...  
};  
  
C b;          // Tạo b - gọi hàm tạo mặc định  
C a(b);       // Tạo a giống b - gọi hàm tạo sao chép mặc định: OK?
```

Ví dụ (hàm tạo sao chép)

```
class C {  
    int x,y;  
    int *z; // Thanh phần con tro  
    ...  
};  
C b;      // Tạo b - gọi hàm tạo mặc định  
C a(b);   // Tạo a giống b - gọi hàm tạo sao chép mặc định
```



Cách xây dựng hàm tạo sao chép

- Có các đặc điểm như một hàm tạo
 - Tên trùng tên lớp
 - Không có kiểu trả về
- Khác hàm tạo thông thường?
 - Có duy nhất một đối để tham chiếu đến đối tượng “nguồn”
 - Sao chép đối tượng “nguồn” vào “đích”
- Cú pháp:

```
Tên_lớp (const Tên_lớp &DT)
{
    // Các câu lệnh
}
```

Ví dụ

```
class DT
{
    int bac;
    float *a;
public:
    DT(int n= 0);
    DT(const DT& d);
    ~DT();
    friend istream& operator>>(istream&
                               is, DT& d);
    friend ostream& operator<<(ostream&
                               os, DT& d);
};

DT::DT(int n) { a= new float[n+1]; }
DT::~~DT() { delete [] a; }
DT::DT(const DT& d)
{
    bac= d.bac;
    a= new float[d.bac+1];
    for (int i=0; i<=bac; ++i)
        a[i]= d.a[i];
}
```

```
istream& operator>>(istream& is,DT& d)
{
    for (int i=0; i<=d.bac; ++i) {
        cout<<"He so bac "<<i<<" ";
        is>>d.a[i];
    }
    return is;
}

ostream& operator<<(ostream& os,DT& d)
{
    for (int i=0; i<=d.bac; ++i)
        os<<d.a[i]<<" ";
    return os;
}

void main()
{
    DT d1(3), d3;
    cin>>d1;
    DT d2(d1);
    cout<<"Da thuc d1:"<<endl<<d1<<endl;
    cout<<"Da thuc d2:"<<endl<<d2;
}
```


So sánh hàm tạo sao chép với toán tử =

- Giống nhau
 - Cần xây dựng khi trong thành phần dữ liệu của lớp có biến con trỏ
- Khác nhau
 - Toán tử gán là gán hai đối tượng đã tồn tại, gán A cho B:
 $B = A$
 - Hàm tạo sao chép tạo ra đối tượng mới B từ đối tượng đã có A:
 $B(A)$

Ví dụ (toán tử gán)

```
// Xây dựng lớp Đa thức
class DT {
    int bac;
    float *a;
public:
    DT() { bac=0; a=NULL; }
    DT(int n) {bac=n; a=new float[n+1];}
    ~DT() { delete a; }
    DT(const DT& dt);
    DT& operator=(DT& dt);
    friend ostream& operator<<(ostream&
                               os, DT& dt);
    friend istream& operator>>(istream&
                               is, DT& dt);
};

// Định nghĩa hàm tạo sao chép
DT::DT(const DT& dt) {
    bac = dt.bac;
    a = new float[bac];
    for(int i= 0; i<= bac; ++i)
        a[i] = dt.a[i];
}
```

```
// Định nghĩa toán tử gán
DT& DT::operator=(DT& dt) {
    bac = dt.bac;
    // a = new float[bac];
    for(int i= 0; i<= bac; ++i)
        a[i] = dt.a[i];
    return dt;
}
...
int main()
{
    DT d1(5), d2(5);
    cout<<"Nhập đa thức d1\n"; cin>>d1;
    cout<<"Nhập đa thức d2\n"; cin>>d2;
    cout<<endl<<"d1 là:"<<endl<<d1;
    cout<<endl<<"d2 là:"<<endl<<d2;
    d2 = d1;
    cout<<endl<<"d2 sau khi gán bằng
                  d1:"<<endl<<d2;

    DT d3(d1);
    cout<<endl<<"d3 được tạo mới và
                  giống d1:"<<endl<<d3;
}
```

Ví dụ

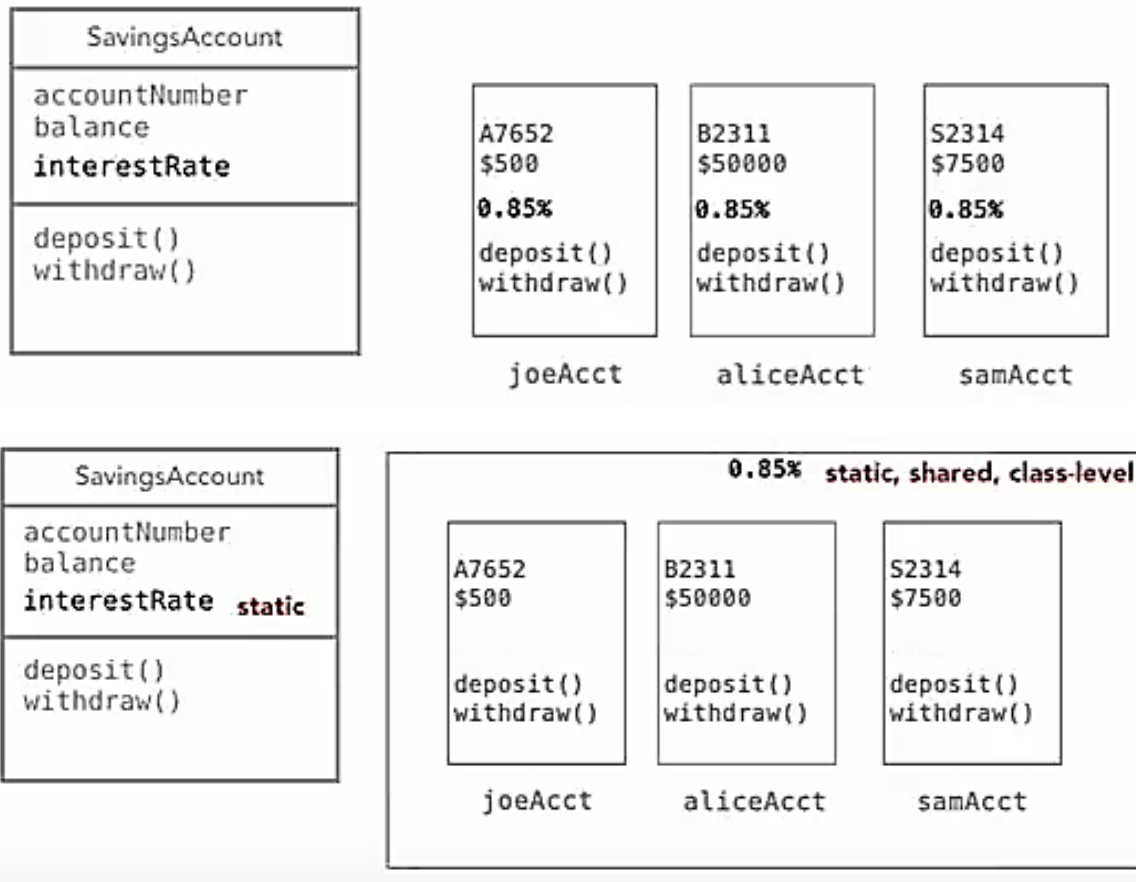
Xây dựng lớp Điểm để biểu diễn điểm trong không gian 2 chiều.

Lớp có:

1. một hàm tạo không đối;
2. một hàm tạo có 2 đối (để khởi gán cho hoành độ, tung độ);
3. hàm tạo sao chép
4. hàm thành phần tính khoảng cách từ một điểm đến gốc tọa độ
5. hàm thành phần tính khoảng cách giữa hai điểm;
6. toán tử `==` để xác định hai điểm có trùng nhau không.
7. toán tử `*` để "nhân" hai điểm. Ví dụ điểm d1 có tọa độ (2,3), điểm d2 có tọa độ (4,7) thì điểm d1*d2 sẽ có tọa độ (8,21).
8. Định nghĩa toán tử `>>` và `<<` để nhập và xuất điểm.

Viết hàm `main()` để sử dụng.

Biến tĩnh



Khai báo biến tĩnh

- Khái niệm
 - Là biến chung của lớp, không của riêng đối tượng nào
 - Tồn tại ngay cả khi chưa khai báo đối tượng nào
- Khai báo và truy nhập
 - Tương tự biến thông thường, thêm `static` ở trước
 - Truy nhập thông qua: tên lớp hoặc tên đối tượng của lớp
- Khởi gán
 - Tường minh, ở bên ngoài lớp

```
Kiểu  Lớp::Biến = Giá_trị;
```
 - Ví dụ:

```
float SavingAccount::InterestRate = 0.0085;
```

Ví dụ (biến tĩnh)

```
#include <iostream>
using namespace std;
class Dummy {
public:
    static int n;
    Dummy () { n++; };
    ~Dummy () { n--; };
};
int Dummy::n = 0;      // khởi gán
int main () {
    Dummy a, b[5];
    Dummy * c = new Dummy;
    cout << a.n << '\n';
    delete c;
    cout << Dummy::n << '\n';
}
```

Phương thức tĩnh

- Phương thức tĩnh
 - Truy nhập các thành phần tĩnh mà không cần đối tượng
 - Không thể truy nhập vào các thành phần riêng của lớp
 - Khi một lớp có biến tĩnh thì cũng thường có phương thức tĩnh
- Cách viết phương thức tĩnh
 - Cách 1: Định nghĩa trong lớp với từ khóa static đặt trước

```
static void print()
{
    cout<<"count= "<<count<<endl;
}
```
 - Cách 2: Khai báo trong lớp với từ khóa static đặt trước, và định nghĩa bên ngoài lớp thì không cần có từ khóa static

Ví dụ (phương thức tĩnh)

```
class A {
    static int count;
public:
    A() { count++; }
    ~A() { count --; }
    static void print() { cout<<"  count= "<<count<<endl; }
};

int A::count=0;          // khởi gán biến tĩnh

void main() {
    cout<<"Result:"<<endl;
    A::print();
    A a1;
    a1.print();
    A* pa= new A;
    a1.print();
    delete pa;           // ~A()
    a1.print();
    A a2= a1;             // gọi hàm tạo sao chép
    a2.print();
}
```


Ví dụ (Cho biết kết quả)

```
class A
{
private:
    static int count;
public:
    A() { count++; }
    ~A(){ count--; }
    static void print()
    {
        cout<<"count= ";
        cout<<count<<endl;
    }
};
int A::count = 0;
```

```
int main()
{
    cout<<"Results:\n";
    A::print();
    A a1;
    a1.print();
    A* pa = new A;
    a1.print();
    delete(pa);
    a1.print();
    A a2 = a1;
    a2.print();
}
```

Bài tập

- Xác định kết quả của chương trình sau:

```
class Demo {
    static int X;
    static int Y;
public:
    static void Print(){
        cout <<"Value of X: " << X << endl;
        cout <<"Value of Y: " << Y << endl;
    }
};

int Demo :: X =10;
int Demo :: Y =20;
int main() {
    Demo OB;
    cout<<"Printing through object name:"<<endl;
    OB.Print();
    cout<<"Printing through class name:"<<endl;
    Demo::Print();
}
```

Bài tập

1. Viết lớp Số phức có các thuộc tính là phần thực, phần ảo, một hàm tạo không đối, một hàm tạo có 2 đối trong đó đối thứ hai mặc định bằng 0, hàm thành phần tính tổng hai số phức. Viết hàm main() để sử dụng.
2. Viết một lớp Hình chữ nhật, gồm có các thuộc tính (private) là cạnh dài, rộng; các phương thức (public): nhập, tính diện tích của hình chữ nhật. Viết hàm main() để: Nhập một dãy n hình chữ nhật, tìm hình có diện tích lớn nhất.
3. Xây dựng lớp Đa thức, trong đó định nghĩa: các hàm tạo có đối, và các phương thức toán tử $<<$, $>>$, $+$, $=$