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

Hà Nội, 2019

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?
  - Hàm tạo được tự động thực hiện khi tạo đối tượng
- Đối tượng được tạo bằng cách nào?
  - Tĩnh : Khai báo biến đối tượng
  - Động: Dùng toán tử new
- Nếu lớp không xây dựng hàm tạo thì sao?
  - Khi không xây dựng hàm tạo thì hàm tạo mặc định sẽ được sử dụng
  - Mọi lớp đều có hàm tạo mặc định (default constructor)
- Khi nào cần xây dựng hàm tạo?

Cách xây dựng hàm tạo

- Tương tự như phương thức thông thường của lớp
- Khác phương thức thông thường:
  - **Tên** hàm tạo phải trùng với tên lớp
  - **Kiểu** trả về không có
  - **Đối** có thể có hoặc không
    - Một lớp có thể có nhiều hàm tạo
- Khi nào cần xây dựng hàm tạo?

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 constructor)
- Khi nào cần xây dựng hàm hủy?

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

- Khi nào cần xây dựng hàm hủy?
  - Khi lớp có con trỏ (đối tượng được cấp phát bộ nhớ động)

[Nếu không xây dựng hàm hủy trong trường hợp này để delete thì những vùng nhớ đã được cấp mà không còn sử dụng sẽ thành "rác" (garbage) trong bộ nhớ]

Ví dụ

```
class DT
{
    int n;
    float *a;   // co con tro nen phai dinh nghia toan tu gan
public:
    DT(int n1); // Constructor
    ~DT()  { this->n = 0; 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];
    }
}
```

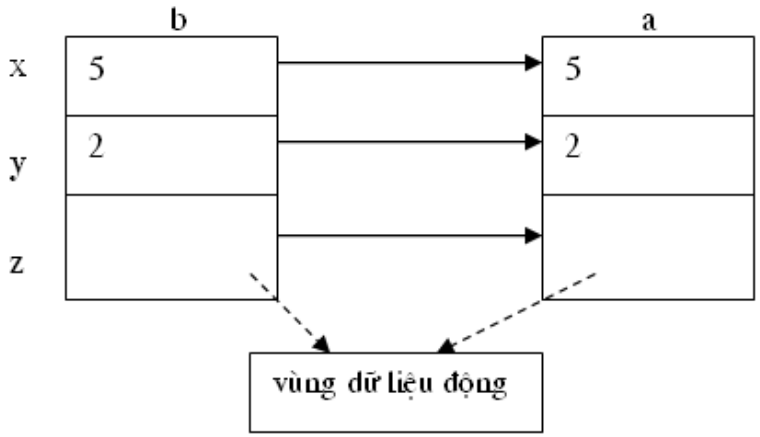
Hàm tạo sao chép (copy constructor)

- 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 phan 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:

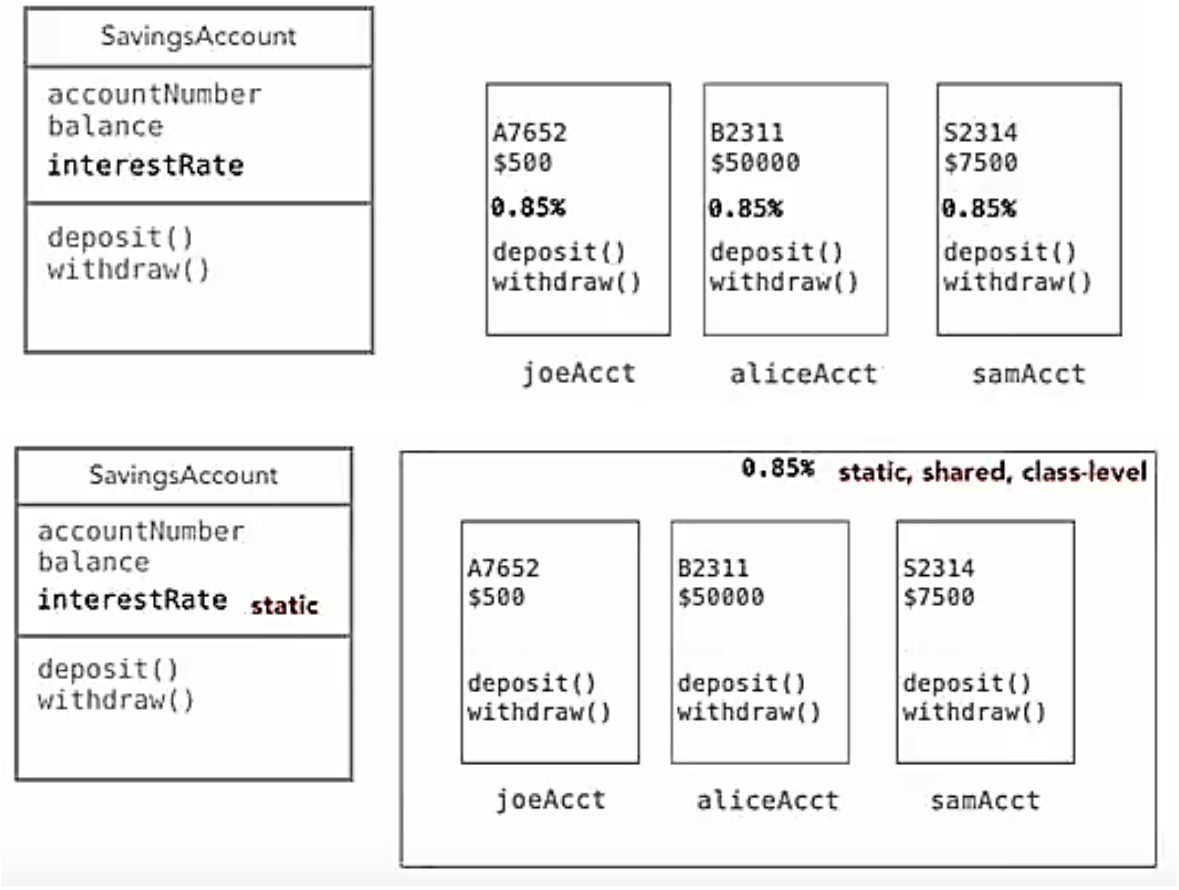
    B(A)

Ví dụ (toán tử gán)

```
// Xay dung lop Da thuc
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);
};
// Dinh nghia ham tao sao chep
DT::DT(const DT& dt) {
  bac = dt.bac;
  a = new float[bac];
  for(int i= 0; i<= bac; ++i)
    a[i] = dt.a[i];
}
```

```
// Dinh nghia toan tu gan
DT& DT::operator=(DT& dt) {
  bac = dt.bac;
  for(int i= 0; i<= bac; ++i)
    a[i] = dt.a[i];
  return dt;
}
...
int main()
{
  DT d1(5), d2(5);
  cout<<"Nhap da thuc d1\n"; cin>>d1;
  cout<<"Nhap da thuc d2\n"; cin>>d2;
  cout<<endl<<"d1 la:"<<endl<<d1;
  cout<<endl<<"d2 la:"<<endl<<d2;
  d2 = d1;
  cout<<endl<<"d2 sau khi gan bang
                    d1:"<<endl<<d2;
  DT d3(d1);
  cout<<endl<<"d3 duoc tao moi va
              giong d1:"<<endl<<d3;
}
```

Biến tĩnh


SavingsAccount
accountNumber
balance
interestRate
deposit()
withdraw()
A7652
$500
0.85%
deposit()
withdraw()
B2311
$50000
0.85%
deposit()
withdraw()
S2314
$7500
0.85%
deposit()
withdraw()
joeAcct
aliceAcct
samAcct
SavingsAccount
accountNumber
balance
interestRate static
deposit()
withdraw()
0.85% static, shared, class-level
A7652
$500
deposit()
withdraw()
B2311
$50000
deposit()
withdraw()
S2314
$7500
deposit()
withdraw()
joeAcct
aliceAcct
samAcct

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
  - Tương tự biến thông thường, thêm **`static`** ở trước
- Khởi gán
  - Tường minh, ở bên ngoài lớp

```
Kiểu  Lớp::Biến = Giá_trị;
```

- Truy nhập vào thành phần tĩnh
  - Qua tên đối tượng
  - Qua tên lớp

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';
}
```

7
6

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();
}
```

```
Result:
     count= 0
     count= 1
     count= 2
     count= 1
     count= 1
```

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;
```

```
void 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();
}
```

```
Result:
     count= 0
     count= 1
     count= 2
     count= 1
     count= 1
```

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, có thuộc tính (riêng) là cạnh dài, rộng, **hàm tạo** có 2 đối để khởi gán giá trị và một hàm thành phần trả về diện tích của hình chữ nhật. Viết hàm main() để sử dụng.
3. Xây dựng lớp Điểm biểu diễn điểm trong không gian 2 chiều. Lớp có một **hàm tạo không đối** và một **hàm tạo có đối**, hàm thành phần tính khoảng cách giữa hai điểm, toán tử **==** để xác định hai điểm có trùng nhau không. Viết hàm main() để sử dụng.
   [B5_Diem2.cpp]
4. Xây dựng lớp Đa thức, trong đó định nghĩa các hàm tạo, và các phương thức toán tử <<, >>, +, = [B4_dathuc.cpp]