KỸ THUẬT LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG VỚI C++

Tạ Quang Chiểu
quangchieu.ta@gmail.com

Chương 6 – Tính đa hình

Xem xét ví dụ 1 sau:

```
#include <iostream>
using namespace std;
int dt=0;
class Hinh {
   protected:
      int chieurong, chieucao;
   public:
      Hinh( int a=0, int b=0)
      {
         chieurong = a;
         chieucao = b;
      }
      int dientich()
      {
         cout << "Dien tich cua lop cha:" <<endl;
         return 0;
      }
};
```

Chương 6 – Tính đa hình

Xem xét ví dụ sau:

```
class HinhChuNhat: public Hinh{
   public:
      HinhChuNhat( int a=0, int b=0):Hinh(a, b) { }
      int dientich ()
      {
         dt= (chieurong * chieucao);
         cout << "Dien tich cua lop HinhChuNhat la: " << dt <<
endl;

      }
};
class TamGiac: public Hinh{
   public:
      TamGiac( int a=0, int b=0):Hinh(a, b) { }
      int dientich ()
      {
         dt= (chieurong * chieucao / 2);
         cout << "Dien tich cua lop TamGiac la: " << dt << endl;
      }
};
```

Chương 6 – Tính đa hình

Xem xét ví dụ sau:

```
// ham main cua chuong trinh
int main( )
{
   Hinh *hinh;
   HinhChuNhat hcn(13,6);
   TamGiac  tag(8, 9);

   // luu giu dia chi cua HinhChuNhat
   hinh = &hcn;
   // goi dien tich cua hinh chu nhat.
   hinh->dientich();

   // luu giu dia chi cua TamGiac
   hinh = &tag;
   // goi dien tich cua tam giac.
   hinh->dientich();

   return 0;
}
```

Chương 6 – Tính đa hình

Biên dịch và chạy chương trình C++ trên sẽ cho kết quả sau:

```
Dien tich cua lop cha:
Dien tich cua lop cha:
```

Chương 6 – Tính đa hình

Chúng ta sửa chương trình trên một chút và đặt trước dòng **`int dientich()`** trong lớp *`Hinh`* với từ khóa **virtual** như sau:

```
#include <iostream>
using namespace std;
int dt=0;
class Hinh {
   protected:
      int chieurong, chieucao;
   public:
      Hinh( int a=0, int b=0)
      {
         chieurong = a;
         chieucao = b;
      }
      virtual int dientich()
      {
         cout << "Dien tich cua lop cha:" <<endl;
         return 0;
      }
};
```

Chương 6 – Tính đa hình

Sau khi sửa đổi, chạy lại chương trình C++ trên sẽ cho kết quả sau:

```
Dien tich cua lop HinhChuNhat la: 78
Dien tich cua lop TamGiac la: 36
```

Chương 6 – Tính đa hình

❖ **TÍNH ĐA HÌNH (Polymorphism)**

Phương thức của lớp cha khi thực hiện sẽ được thay thế bằng một phương thức của lớp con thì phương thức này gọi là có tính đa hình. Tính đa hình giúp cho việc lập trình đơn giản và dễ mở rộng. Để cài đặt phương thức có tính đa hình ta dùng phương thức ảo và phương thức thuần ảo.

❖ **PHƯƠNG THỨC ẢO (virtual method)**

Phương thức ảo là phương thức được định nghĩa ở lớp cơ sở (lớp cha) mà các lớp dẫn xuất (lớp con) muốn sử dụng phải định nghĩa lại. Dùng từ khoá **virtual** để khai báo phương thức ảo:

Chương 6 – Tính đa hình

Cú pháp:

```
virtual <kiểu trả về> <tên phương thức >(<d/s tham số>)
{
      …
}
```

- ✓ Phương thức khởi tạo không được là phương thức ảo nhưng phương thức hủy bỏ có thể là phương thức ảo.
- ✓ Dùng phương thức ảo chậm hơn phương thức thông thường vì khi thực hiện mới được xác định cụ thể.
- ✓ Phương thức (hàm ảo) là hàm thành phần của lớp
- ✓ Được khai báo trong lớp cơ sở và định nghĩa lại trong lớp dẫn xuất.
- ✓ Hàm ảo sẽ được gọi thực hiện từ đối tượng của lớp dẫn xuất nhưng mô tả chúng trong lớp cơ sở.

Chương 6 – Tính đa hình

Ví dụ 2

```
#include <iostream.h>
#include <conio.h>
class A
{
  public:
  virtual void Chao() //phuong thuc ao
 {
  cout<<"\nA chao cac ban";
 }
};

// class B
class B:public A
{
  public:
  void Chao()
   {
     cout<<"\nB chao cac ban";
   }
 };
```

Chương 6 – Tính đa hình

Ví dụ 2

```
// class C
class C:public A
{
  public:
  void Chao()
  {
    cout<<"\nC chao cac ban";
  }
};
// ham main
void main()
{
   A a;
   A *pa= new A; pa->Chao(); //goi chao cua A
   B b;
   pa=&b; pa->Chao(); //goi chao cua B
   C c;
   pa=&c; pa->Chao(); //goi chao cua C
   getch();
}
```

Chương 6 – Tính đa hình

Nhận xét:

- ✓ Phương thức `Chao()` có tính đa hình: cùng lời gọi `pa->chao()` nhưng lần 1 gọi `chao cua A`, lan 2 gọi `chao cua B`, lan 3 goi `chao cua C`. Nếu trong lớp B, C không định nghĩa lại phương thức chào thì cả ba lần đều gọi chào của A.
- ✓ Nếu phương thức `chao()` trong lớp A không khai báo virtual thì phương thức `chao()` sẽ không có tính đa hình, khi đó cả ba lần đều gọi chào của A.
- ✓ Có thể gán địa chỉ của đt thuộc lớp con vào biến con trỏ, trỏ tới đt thuộc lớp cha nhưng không thể làm ngược lại (áp dụng nguyên tắc "con gán vào cha" đối với biến kiểu đối tượng hoặc biến kiểu con trỏ, trỏ tới đối tượng)

ví dụ 3: Giả sử có 3 lớp A, B và C được xây dựng như sau

```
class A
  {
          public:
      void xuat()
       {
         cout <<"\n Lop A";
       }
  };
  class B : public A
  {
       public:
       void xuat()
       {
              cout <<"\n Lop B";
       }
  };
```

```
class C : public B
          {
           public:
                   void xuat()
                  {
                   cout <<"\n Lop C";
                  }
          };
```

- Cả 3 lớp này đều có hàm thành phần là xuat().
- Lớp C có hai lớp cơ sở là A, B và C kế thừa các hàm thành phần của A và B.
- Do đó một đối tượng của C sẽ có 3 hàm xuat().

- ✓ Xem các câu lệnh sau:

```
C  ob; //  ob là đối tượng kiểu C
ob.xuat();      // Gọi tới hàm thành phần xuat() của lớp C
ob.B::xuat() ;  // Gọi tới hàm thành phần xuat() của lớp B
ob.A::xuat() ;  // Gọi tới hàm thành phần xuat() của lớp A
```

- ✓ Các lời gọi hàm thành phần trong ví dụ trên đều xuất phát từ đối tượng `ob` và mọi lời gọi đều xác định rõ hàm cần gọi.

- Ta xét tiếp tình huống các lời gọi không phải từ một biến đối tượng mà từ một con trỏ đối tượng. Xét các câu lệnh:

```
A   *p,  *q, *r;   // p,q,r là các con trỏ kiểu A
A  a;              // a là đối tượng kiểu A
B  b;              // b là đối tượng kiểu B
C  c;              // c là đối tượng kiểu C
```

- Bởi vì con trỏ của lớp cơ sở có thể dùng để chứa địa chỉ các đối tượng của lớp dẫn xuất, nên cả 3 phép gán sau đều hợp lệ:

```
p = &a;     q = &b;r = &c;
```

- Ta xét các lời gọi hàm thành phần từ các con trỏ p, q, r:

```
p->xuat();          q->xuat();     r->xuat();
```

- Cả 3 câu lệnh trên đều gọi tới hàm thành phần `xuat()` của lớp A, bởi vì các con trỏ p, q, r đều có kiểu lớp A.
- Sở dĩ như vậy là vì một lời gọi (xuất phát từ một đối tượng hay con trỏ) tới hàm thành phần luôn luôn liên kết với một hàm thành phần cố định và sự liên kết này xác định trong quá trình biên dịch chương trình. Ta bảo đây là **sự liên kết tĩnh**.

- Có thể tóm lược cách thức gọi các hàm thành phần như sau:
  - Nếu lời gọi xuất phát từ một đối tượng của lớp nào đó, thì hàm thành phần của lớp đó sẽ được gọi.
  - Nếu lời gọi xuất phát từ một con trỏ kiểu lớp, thì hàm thành phần của lớp đó sẽ được gọi bất kể con trỏ chứa địa chỉ của đối tượng nào.
- **Vấn đề đặt ra là**: Ta muốn tại thời điểm con trỏ đang trỏ đến đối tượng nào đó thì lời gọi hàm phải liên kết đúng hàm thành phần của lớp mà đối tượng đó thuộc vào chứ không phụ thuộc vào kiểu lớp của con trỏ.
- C++ giải quyết vấn đề này bằng cách dùng khái niệm **hàm ảo**.

Ví dụ 4:

```
#include <iostream.h>
#include <conio.h>
class A
{     public:
      virtual  void hienthi() { cout<<"Lop co so A"; }
};
class B : public A
{     public:
      void hienthi() { cout<<"Lop dan xuat B"; }
};
class C : public A
{     public:
      void hienthi() { cout<<"Lop dan xuat C"; }
};
```

```
void main()
{     clrscr();
      A  a, *p;
      p = &a;  p->hienthi(); cout<<"\n";
      B  b;
      p = &b;     p -> hienthi(); cout<<"\n";
      C c;
      p = &c;     p -> hienthi();
      getch();
}
```

Kết quả:

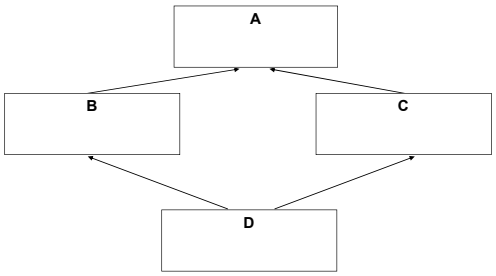
Lop co so A
Lop dan xuat B
Lop dan xuat C

Một số chú ý:

- Định nghĩa các hàm ảo giống như các hàm thông thường
- Sử dụng con trỏ để truy cập tới hàm ảo
- Định nghĩa trong lớp cơ sở ngay cả khi nó không được sử dụng
- Không có hàm khởi tạo ảo, nhưng có thể có hàm huỷ ảo
- Con trỏ của lớp cơ sở có thể chứa địa chỉ của đối tượng lớp dẫn xuất

Lớp cơ sở ảo

Xét ví dụ A là lớp cơ sở của lớp B và lớp C, D là lớp dẫn xuất của lớp B và lớp C theo sơ đồ phân cấp sau:




```
#include <iostream.h>
class   A
{   public:
       int   i;
};
class  B : public  A
{    public:
       float   f;
};
class  C : public  A
{    public:
       double   d;
};
```

```
class  D : public  B , public  C
{    public:
       char   c;
};
void   main()
{ D   d;
  d.i = 0; //Nhập nhằng: Member is ambiguous: 'A::i' and
  'A::i'
  d.f = 3.141593;  d.d = 1.5;     d.c = 'a';
  cout<<"i = "<<d.i<<endl; //Nhập nhằng: Member is ambiguous
  cout<<"f = "<<d.f<<endl;
  cout<<"d = "<<d.d<<endl;
  cout<<"c = "<<d.c;
}
```

Giải quyết:

Coi A là lớp cơ sở ảo của cả B và C. Khi đó trong D chỉ có một sự thể hiện của A

Khai báo:

```
class <tên lớp dẫn xuất>: virtual <kiểu dẫn xuất><lớp cơ sở>
```

VD:

```
class  B: virtual  public A { … } ;
class  C: virtual  public A { … } ;
```

```
#include <iostream.h>
#include <conio.h>
class   A
{   public:
       int   i;
};
class  B : virtual  public  A
{   public:
       float   f;
};
class  C : virtual  public  A
{   public:
       double   d;
};
```

```
class  D : public  B, public  C
{   public:
       char  c;
};
void  main()
{ D   d;
  d.f = 3.141593; d.i = 0;  d.c = 'a'; d.d = 1.5;
  cout<<"i = "<< d.i<<endl;
  cout<<"f = "<<d.f<<endl;
  cout<<"d = "<<d.d<<endl;
  cout<<"c = "<<d.c;
  getch();
}
```

Lớp trừu tượng và hàm ảo thuần tuý

Mục đích:

- Tránh lãng phí bộ nhớ

-Cung cấp một phương thức thống nhất làm giao diện chung.

Khai báo hàm ảo thuần túy:

```
virtual <kiểu trả về> <tên phương thức>([các tham số]) = 0;
```

Đặc điểm:

- Không bắt buộc định nghĩa trong lớp cơ sở
- Không thể khai báo đối tượng thuộc lớp có hàm ảo thuần tuý
- Lớp có hàm ảo thuần tuý và chỉ làm lớp cơ sở cho lớp khác gọi là lớp cơ sở trừu tượng
- Lớp dẫn xuất kế thừa lớp cơ sở trừu tượng mà không định nghĩa lại phương thức ảo thuần tuý → nó trở thành lớp cơ sở trừu tượng

```
#include  <iostream.h>
#include  <conio.h>
#define   PI   3.141593
class  Hinh
{   public:
      virtual   float  Dien_tich() = 0; // Hàm ảo thuần túy
};
class  Tron : public  Hinh
{       float   r;
    public:
       Tron(float   rr=0) { r = rr; }
       //Định nghĩa lại
       float  Dien_tich() { return PI*r*r; }
};
```

```
class Chu_nhat: public Hinh
{       float  dai, rong;
   public:
      Chu_nhat(float  d = 0, float  r = 0)
       { dai = d; rong = r; }
      float  Dien_tich() { return   dai*rong;}  //Định nghĩa
  lại
};
```

```
void main()
{
   Hinh *hinh;
   Tron  tron(5); //Khai bao hinh tron ban kinh  5
   Chu_nhat  ch_nhat(4,3); //Kh/bao chu nhat voi kich thuoc 4
  va 3
   hinh = &tron;
   cout << hinh -> Dien_tich() << endl;
   hinh = &ch_nhat;
   cout << hinh -> Dien_tich() << endl;
   getch();
}
```

Bài tập chương 6

Bài 1. Xây dựng chương trình thực hiện các công việc sau:

- Cài đặt lớp người Nguoi gồm có:
  - Dữ liệu: họ tên, phương thức nhập, phương thức ảo in ra, phương thức ảo được khen thưởng
- Cài đặt lớp sinh viên SinhVien kế thừa lớp Nguoi và bổ sung:
  - Dữ liệu: điểm trung bình, phương thức: định nghĩa lại phương thức nhập, phương thức ảo in, phương thức ảo được khen thưởng nếu điểm trung bình từ 9 trở lên
- Cài đặt lớp giảng viên GiangVien kế thừa lớp Nguoi và bổ sung:
  - Dữ liệu: số bài báo, phương thức: định nghĩa lại phương thức nhập, phương thức ảo in, phương thức ảo được khen thưởng nếu có số bài báo từ 5 trở lên
- Chương trình chính: nhập mảng các n người (n < 100), in ra danh sách này.