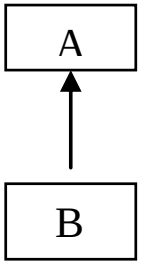
CHƯƠNG 5

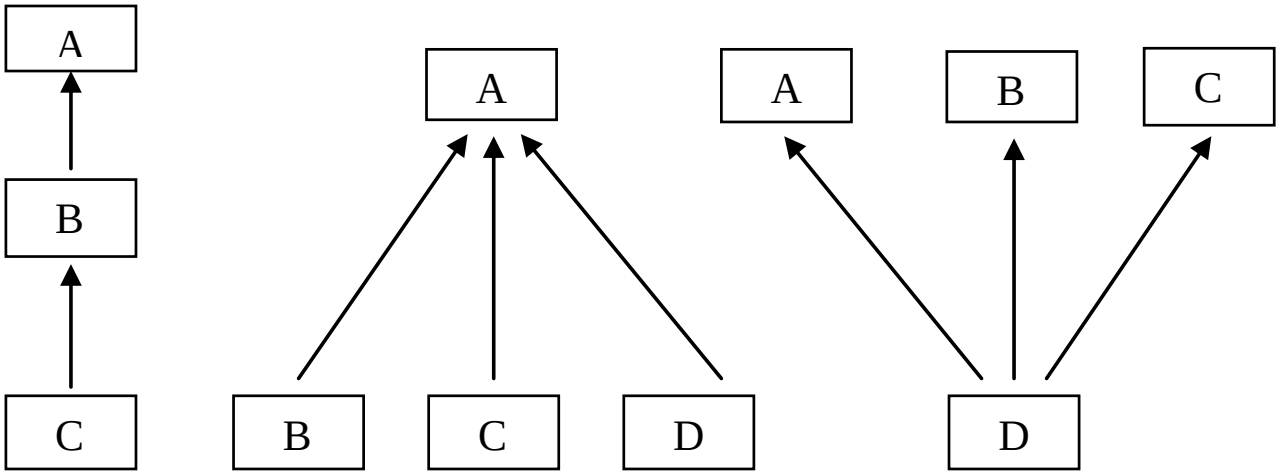
KẾ THỪA VÀ TÍNH ĐA HÌNH

5.1. KẾ THỪA

C++ Có hai loại kế thừa là: đơn kế thừa và đa kế thừa, có thể minh họa qua các hình vẽ sau đây:



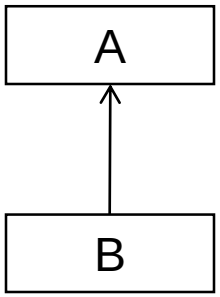
Hình 5.1. Đơn kế thừa, lớp A là lớp cơ sở của lớp B



Hình 5.2. Đa kế thừa

5.1.1. Đơn kế thừa

Cú pháp:




```
class  A   //lop A la lop co so
   {
        private:
                // Khai báo các thuộc tính, phương thức
        protected:
                // Khai báo các thuộc tính, phương thức
        public:
                // Khai báo các thuộc tính và phương thức
   };

class  B: mode A    //lop B la lop dan xuat
   {
        private:
                // Khai báo các thuộc tính, phương thức
         protected:
                // Khai báo các thuộc tính, phương thức
        public:
                 // Khai báo các thuộc tính, phương thức
   };
```

- Trong đó, mode là kiểu kế thừa, có thể là **public, protected** hoặc **private** với ý nghĩa như sau:
- - Kế thừa theo kiểu **public** thì tất cả các thành phần **public** của lớp cơ sở trở thành thành phần **public** của lớp dẫn xuất và các thành phần **protected** của lớp cơ sở trở thành thành phần **protected** của lớp dẫn xuất.
- - Kế thừa theo kiểu **protected** thì tất cả các thành phần **public** và **protected** của lớp cơ sở sẽ trở thành các thành phần **protected** của lớp dẫn xuất.
- - Kế thừa theo kiểu **private** thì tất cả các thành phần **public** và **protected** của lớp cơ sở sẽ trở thành các thành phần **private** của lớp dẫn xuất.

Ví dụ 5.1

Ví dụ 5.2

Ví dụ 5.3

```
//Vi du 5.1
#include <iostream>
using namespace std;
class base
{
        private:     int x;
        protected:   int y;
        public:
            int z;
            base()
            {
              x = 1;    y = 2;   z = 3;
            }
};
class derive: private base
{     // y và z tro thành thành phan private cua lop dan xuat
        public:
            void showdata()
            {
               cout<<"x khong the truy cap"<< endl;
              cout<<"gia tri cua y la "<<y<< endl;
              cout<<"gia tri cua z la "<<z<< endl;
            }
};
```

```
int main()
{
    derive a;
    a.showdata();
    //a.y = 5;  //loi truy ca vung rieng
    //a.z = 10; //loi truy cap vung rieng
    // a.x = 1; loi : thành phan private không the truy cap bên ngoài lop
    // a.y = 2; loi : y lúc này là thành phan private cua lop dan xuat
    //  a.z = 3; loi : lúc này z cung là thành phan private cua lop dan xuat
    return 0;
}
```

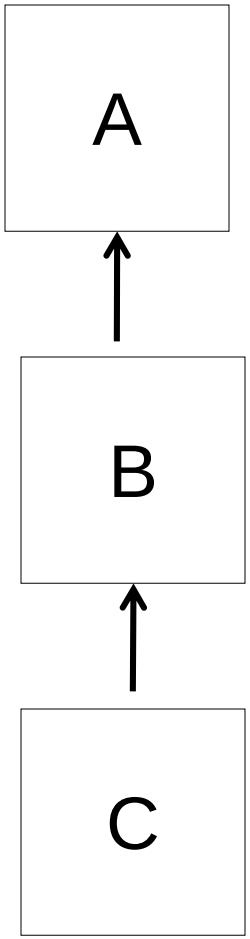
```
// Vi dụ 5.2
#include <iostream>
using namespace std;
class base
{
        private:
           int x;
        protected:
           int y;
        public:
           int z;
           base()
           {
             x = 1;     y = 2;     z = 3;    }
};
class derive: protected base
{       /* y và z tr? thành thành ph?n protected     C?a l?p d?n xu?t */
        public:
           void showdata()
           {
              cout<<"x khong the truy cap"<< endl;
             cout<<"gia tri cua y la "<<y<<endl;
             cout<<"gia tri cua z la "<<z<<endl;
           }
};
```

```
int main()
{
    derive a;
    a.showdata();
    /* a.x = 1; loi : thành phan private  không th? truy
                        cap bên ngoài lop */
    //a.y = 2; //  loi : y bây gio cung là
    //thành ph?n private cua lop dan xuat */
     //a.z = 3; //  loi : z bây gio cung là
      //thành phan private cua lop dan xuat */
    return 0;
}
```

//Ví dụ 5.3

```
#include <iostream>
using namespace std;
class base
{
        private:
           int x;
        protected:
           int y;
        public:
           int z;
           base()
           {
             x = 1;   y = 2;   z = 3;    }
};
class derive: public base
{
    /* y tro thành thành phan protected và z  tr? thành thành ph?n public c?a l?p   d?n xu?t*/
        public:
           void showdata()
```

```
int main()
{
    derive a;
    a.showdata();
    // a.x = 1;   loi : thành phan private không the
                      truy cap bên ngoài lop */
     //a.y = 2; //   loi : y bây gio là thành viên
                    //protected cua lop dan xuat
     a.z = 3;    //khong loi vi z thuoc vung public khi
                      //ke thua
    return 0;
}
```

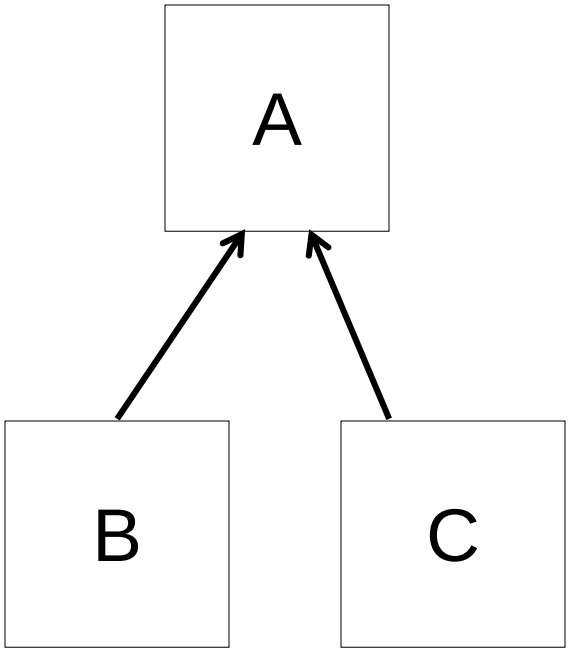


5.1.2. Đa kế thừa

1. Kế thừa đa mức

```
class A
{…};
class B : mode A
{…};
class  C: mode B
 {
    private:  // …
    public:  // …
};
```

trong đó mode có thể là **private, public hoặc protected**. Ý nghĩa của kiểu dẫn xuất này giống như trường hợp đơn kế thừa.

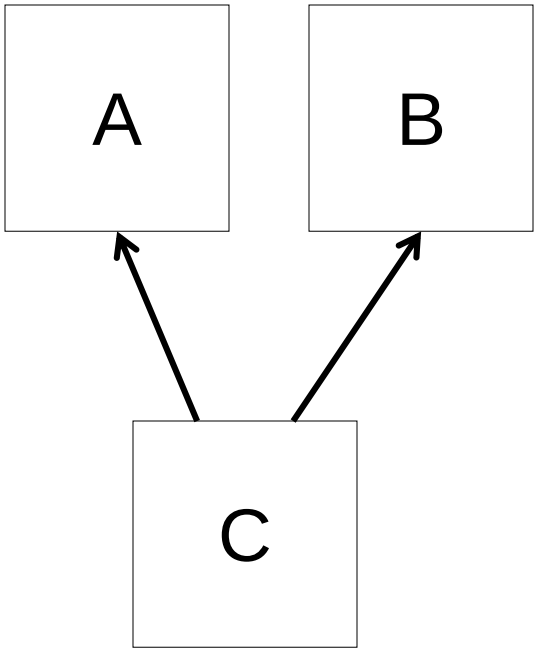


2. Kế thừa phân cấp

```
class A
{…};

class B : mode A
{…};

class  C:  mode A
 {
    private:  // …
    public:  // …
};
.
```



3. Kế thừa bội

```
class A
{…};

class B
{…};

class  C: mode A, mode B
 {
    private:  // …
    public:  // …
};
.
```

Ví dụ 5.4
Ví dụ 5.5
Ví dụ 5.6

```
//Vi du 54
#include <iostream>
using namespace std;
class base
{
        protected: int x;
        public:

          void read()
          {
      cout<<"Nhap x = ";
      cin>>x;
          }
};
```

```
class derive1 : public base
{
        protected: int y;
        public:
          void readdata()
          {
            cout<<"\nNhap y
= ";
      cin>>y;
          }
};
```

```
//Vi du 55
#include <iostream>
using namespace std;
class A
{
    protected: int x, y;
            public:
                    void read()
                    {
                cout<<"\nNhap gia tri cua x va y:\n";
        cin>>x>>y;
            }
};
```

```
class B : public A
{
    public:
            void Prod()
            {
                cout<<"\nTich = "<<x * y;
             }
};
```

```
 class C : public A
 {
     public:
                        void sum()
                        {
          cout<<"\nTong = "<<x + y;
                        }
 };
```

```
int main()
{
    B ob1;
    C ob2;
    ob1.read();
    ob1.Prod();
    ob2.read();
    ob2.sum();
    return 0;
}
```

//Vi du 56

```
#include <iostream>
using namespace std;
class A
{
        protected: int x;
        public:
        void read_x()
         {
          cout<<"Nhap gia tri cua x: ";
                cin>>x;
        }
};
class B
{
        protected: int y;
        public:

        void read_y()
        {
          cout<<"Nhap gia tri cua y: ";
          cin >> y;
        }
};
```

```
class C : public A, public B
{
   public:
      void sum()
     {
       cout<<"Tong = "<< x + y;
     }
};
```

```
int main()
{           C ob1;
            ob1.read_x();
            ob1.read_y();
            ob1.sum();
            return 0;
}
```

```
class derive2 : public derive1
{
        private:
           int z;
        public:
           void indata()
           {
             cout<<"\nNhap z= ";
              cin>>z;
           }
        void Prod()
        {
            cout<<"\nTich = "<<x * y * z;
        }
};
```

```
int main()
{
    derive2 a;
    a.read();  //goi ham lop base
    a.readdata(); //goi ham lop derive2
    a.indata();
    a.Prod();
    return 0;
}
```

5.1.3. Truy cập các thành phần trong lớp dẫn xuất

Thành phần của lớp dẫn xuất bao gồm: các thành phần khai báo trong lớp dẫn xuất và các thành phần mà lớp dẫn xuất thừa kế từ các lớp cơ sở.

Chú ý: Một đối tượng của lớp dẫn xuất có thể gọi đến phương thức của lớp cơ sở theo mẫu:

Tên đối tượng.Tên_lớp::phương thức()

Chú ý:

1. Để sử dụng phương thức của lớp dẫn xuất, có thể không dùng tên lớp, dùng tên phương thức. Khi đó chương trình dịch phải tự phán đoán để biết phương thức đó thuộc lớp nào:

   Trước tiên xem phương thức đang xét có trùng tên với phương thức nào của lớp dẫn xuất không?

- Nếu trùng tên thì đó là phương thức của lớp dẫn xuất.
- Nếu không trùng tên thì tiếp tục xét các lớp cơ sở theo thứ tự: các lớp có quan hệ gần với lớp dẫn xuất sẽ được xét trước, các lớp quan hệ xa hơn xét sau.

Chú ý:

2. Trong lớp dẫn xuất, ta có thể định nghĩa lại phương thức của lớp cơ sở. Như vậy có hai phiên bản khác nhau của phương thức trong lớp dẫn xuất. Trong phạm vi lớp dẫn xuất, hàm định nghĩa lại "che khuất" hàm được định nghĩa. Việc sử dụng hàm nào cần tuân theo quy định ở trên.

5.1.4. Hàm tạo đối với tính kế thừa

Các hàm tạo của lớp cơ sở là không được kế thừa. Một đối tượng của lớp dẫn xuất có thể xem là một đối tượng của lớp cơ sở, vì vậy việc gọi hàm tạo lớp dẫn xuất sẽ kéo theo việc gọi đến một hàm tạo của lớp cơ sở.

Thứ tự thực hiện của các hàm tạo sẽ là: hàm tạo cho lớp cơ sở, rồi đến hàm tạo cho lớp dẫn xuất.

C++ thực hiện điều này bằng cách: trong định nghĩa của hàm tạo lớp dẫn xuất, ta mô tả một lời gọi tới hàm tạo trong lớp cơ sở.

Cú pháp để truyền đối số từ lớp dẫn xuất đến lớp cơ sở:

```
lớpdẫnxuất(danh sách đối):lớpcơsở (danh sách đối)
        {
          //thân hàm tạo của lớp dẫn xuất
       } ;
```

Ví dụ 5.8

5.1.6. Một số ví dụ về đa kế thừa

Ví dụ 5.11

Ví dụ 5.12

Ví dụ 5.13