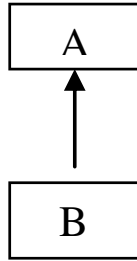


## CHƯƠNG 5

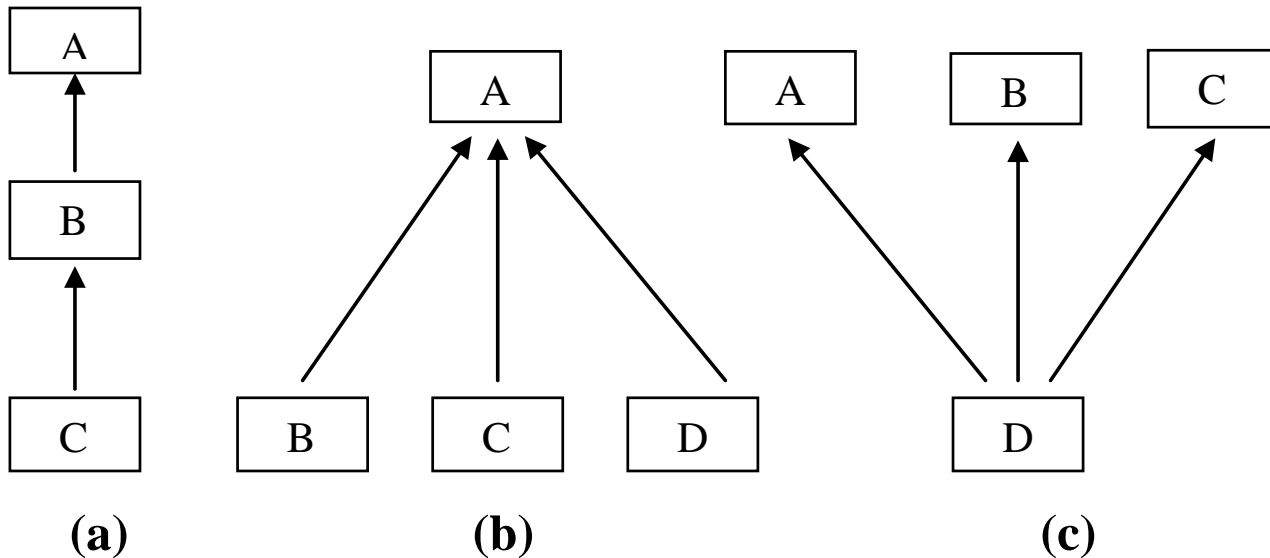
# KẾ THỪA VÀ TÍNH ĐA HÌNH

## 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 //lớp A là lớp cơ sở
```

```
{
```

```
    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 //lớp B là lớp dẫn xuất
```

```
{
```

```
    private:
```

```
        // Khai báo các thuộc tính, phương thức
```

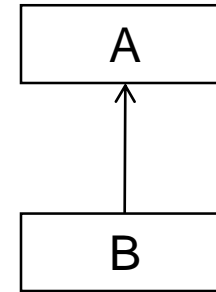
```
    protected:
```

```
        // 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.

**Chú ý: Vùng private của lớp cơ sở không được kế thừa**

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 trở thành thành phần private của lớp dẫn xuất
    public:
        void showdata()
        {
            cout<<"x không thể truy cập"<< endl;
            cout<<"giá trị của y là "<<y<< endl;
            cout<<"giá trị của z là "<<z<< endl;
        }
};
```

```
int main()
{
    derive a;
    a.showdata();
    //a.y = 5; //lỗi truy cập vùng riêng
    //a.z = 10; //lỗi truy cập vùng riêng
    // a.x = 1; lỗi : thành phần private không thể truy cập bên ngoài lớp
    // a.y = 2; lỗi : y lúc này là thành phần private của lớp đang xuất
    // a.z = 3; lỗi : lúc này z cũng là thành phần private của lớp đang xuất
    return 0;
}
```

## // Ví 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 trở thành thành phần protected của lớp dẫn xuất */
    public:
        void showdata()
        {
            cout<<"x không thể truy cập"<< endl;
            cout<<"giá trị của y là "<<y<<endl;
            cout<<"giá trị của z là "<<z<<endl;
        }
};
```



```

int main()
{
    derive a;
    a.showdata();
    /* a.x = 1; loi : thành phần private không thể truy
        cap bên ngoài lớp */
    //a.y = 2; // loi : y bây giờ cũng là
    //thành phần protected của lớp dẫn xuất */
    //a.z = 3; // loi : z bây giờ cũng là
    //thành phần protected của lớp dẫn xuất */
    return 0;
}

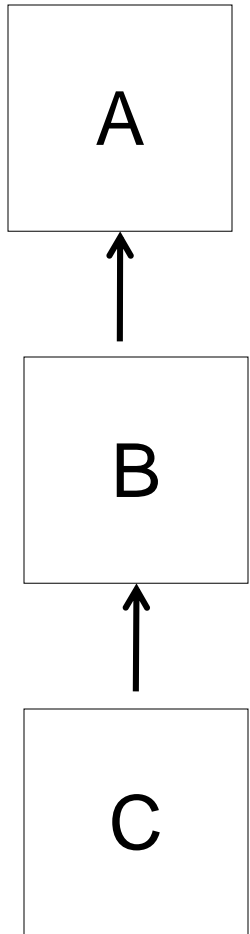
```

### //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 trở thành thành phần protected và z trở thành thành phần public của lớp dẫn xuất*/
    public:
        void showdata()
        { cout<<"x không thể truy cập"<< endl;
          cout<<"giá trị của y là "<<y<<endl;
          cout<<"giá trị của z là "<<z<<endl;  }
};
```

```
int main()
{
    derive a;
    a.showdata();
    // a.x = 1;   loi : thành phần private không thể
                  truy cập bên ngoài lớp */
    //a.y = 2; //   loi : y bây giờ là thành viên
                  //protected của lớp dẫn xuất
    a.z = 3;   //không loi vì z thuộc vùng public khi
               //kế thừa
    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: // ...
};
```

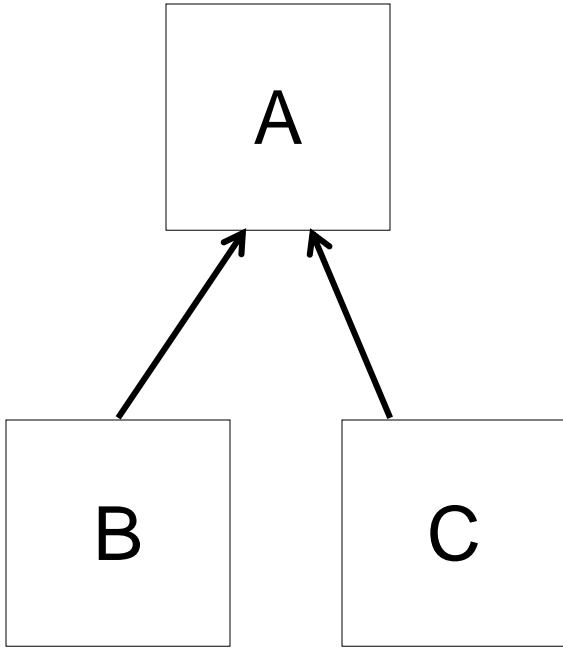
trong đó mode có thể là **private**, **public** hoặc **protected**. Ý nghĩa của kiểu dẫn xuất này giống như trường hợp đơn kế thừa.

## 2. Kế thừa phân cấp

class A  
{...};

class B : mode A  
{...};

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



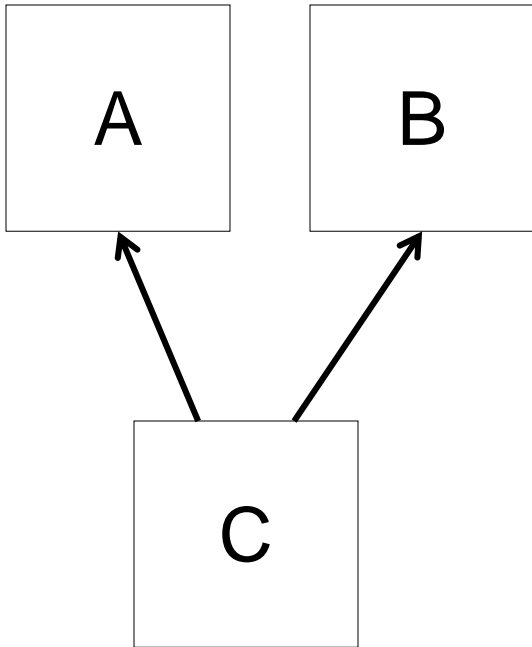
### 3. Kế thừa bội

```
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;
        }
};
```



```

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 ba
    a.readdata(); //goi ham lop
    a.indata();
    a.Prod();
    return 0;
}

```

## //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;
}
```

### 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:

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

## **5.1.6. Một số ví dụ về đa kế thừa**

**Ví dụ 5.11 (Xem giáo trình)**

**Ví dụ 5.12 (Xem giáo trình)**

**Ví dụ 5.13**

**Ví dụ 5.11.** Chương trình sau minh họa việc quản lý kết quả thi của một lớp không quá 100 sinh viên. Chương trình gồm 3 lớp: lớp cơ sở sinh viên (Sinhvien) chỉ lưu họ tên và số báo danh, lớp điểm thi (Diemthi) kế thừa lớp sinh viên và lưu kết quả môn thi 1 và môn thi 2. Lớp kết quả (Ketqua) lưu tổng số điểm đạt được của sinh viên.

```

#include <iostream>
#include <stdio.h>
using namespace std;
class Sinhvien
{
private:
    int Sbd;
    char Hoten[25];
public:
    void Nhap()
    {
        cout<<"So bao danh :";cin>>Sbd;
        cout<<"\nHo ten :";fflush(stdin);
        gets(Hoten);
    }
    void Xuat()
    { cout<<"So bao danh : "<<Sbd<<endl;
      cout<<"Ho va ten sinh vien : "
        <<Hoten<<endl;
    }
};

```

```

class Diemthi : public Sinhvien
{ protected :
    float D1,D2;
public :
    void Nhap_diem()
    {
        cout<<"Nhap diem hai mon thi : \n";
        cin>>D1>>D2;
    }
    void Xuat_diem()
    {
        cout<<"Diem mon 1 :"<<D1<<endl;
        cout<<"Diem mon 2 :"<<D2<<endl;
    }
};

class ketqua : public Diemthi
{
private:
    float Tong;
public :
    void Xuat_kq()
    { Tong = D1+D2;
      Xuat();
      Xuat_diem();
      cout<<"Tong so diem :"<<Tong<<endl;
    }
};

```

```
int main()
{
    int i,n; ketqua sv[100];
    cout<<"\n Nhap so sinh vien : ";
    cin>>n;
    for(i=0;i<n;++i)
        { sv[i].Nhap();
          sv[i].Nhap_diem();
        }
    for(i=0;i<n;++i)
        sv[i].Xuat_kq();
    return 0;
}
```

**Ví dụ 5.12.** Chương trình sau là sự mở rộng của chương trình ở trên, trong đó ngoài kết quả hai thi, mỗi sinh viên còn có thể có điểm ưu tiên. Chương trình mở rộng thêm một lớp ưu tiên (Uutien) để lưu điểm ưu tiên, cấu trúc phân cấp lớp được minh họa như hình sau:

**Ví dụ 5.13.** Giả sử cuối năm học ở một trường đại học cần trao phần thưởng cho các sinh viên giỏi, các giáo viên có tham gia nghiên cứu khoa học. Điều kiện khen thưởng của sinh viên là có điểm trung bình lớn hơn 8. Điều kiện khen thưởng của giáo viên là có hơn một bài báo nghiên cứu khoa học. Sinh viên có các thuộc tính: mã sinh viên, họ tên, điểm trung bình. Giáo viên có các thuộc tính: mã giáo viên, họ tên, số bài báo nghiên cứu khoa học. Chương trình sau minh họa việc nhập dữ liệu của không quá 100 sinh viên và không quá 20 giáo viên và in ra danh sách các sinh viên và giáo viên được khen thưởng.



```

#include <iostream>
#include <stdio.h>
using namespace std;
class Nguoi
{
    char Ma[10],Hoten[25];
public :
    void Nhap()
    {
        cout<<"\n Nhap Ma: ";
        gets(Ma);
        cout<<"\n Ho ten: ";
        gets(Hoten);
    }
    void Xuat()
    {
        cout<<"\n Ma: "<<Ma;
        cout<<"\n Ho ten: "<<Hoten;
    }
};

```

```

class Sinhvien : public Nguoi
{
    float Dtb;
public:
    void Nhap_SV()
    {
        cout<<"\n Nhap diem trung binh: ";
        cin>>Dtb;
        cin.ignore(1);
    }
    void Xuat_SV()
        { cout<<"\n Diem trung binh: "<<Dtb;}
    int khenthuong()
    {
        return Dtb > 8;
    }
};

```

```

class Giaovien : public Nguoi
{
    float sbb;
public :
    void Nhap_GV()
    {
        cout<<"\nNhap so bai bao: ";
        cin>>sbb;
        cin.ignore(1);
    }
    void Xuat_GV()
    {
        cout<<"\n So bai bao: "<<sbb;
    }
    int khenthuong()
    {
        return sbb > 1;
    }
};

```

```

int main()
{
    Sinhvien sv[100];
    Giaovien gv[20];
    int m,n;
    cout<<"\n Nhap so sinh vien: ";
    cin>>n;
    cin.ignore(1);
    for(int i=0;i<n;++i)
    {
        sv[i].Nhap();
        sv[i].Nhap_SV();
    }
    cout<<"\n Danh sach SV duoc khen thuong :";
    for(int i=0;i<n;++i)
        if (sv[i].khenthuong())
        { sv[i].Xuat();
          sv[i].Xuat_SV();
        }
}

```

```

cout<<"\n Nhap so giao vien : ";
cin>>m;
cin.ignore(1);
for(int i=0;i<m;++i)
{
    gv[i].Nhap();
    gv[i].Nhap_GV();
}
cout<<"\n Danh sách GV duoc khen thuong :";
for(int i=0;i<n;++i)
    if (gv[i].khenthuong())
    {
        gv[i].Xuat();
        gv[i].Xuat_GV();
    }
return 0;
}

```

Thảo luận:

1. Viết lại chương trình trong ví dụ 5.13, trong đó việc xử lý nhập dữ liệu, khen thưởng được thực hiện ở một lớp có tên là Quanly. Hàm main chỉ gọi để thực thi.
2. Xét bài tập của Chương 3 về các chuyến xe nội thành, ngoại thành. Viết Chương trình minh họa theo kế thừa.