C++第五天

一、静态成员（static）

所谓静态成员，是想让类中的某个成员或者某几个成员，不依附于某个类对象的空间而独立存在。
我们调用类中成员时，通常使用该类实例化出来的对象，然后通过对象调用类中后才能有，但是有时我们希望某个成员不依赖于某个类对象，并且能实现每个类对象都有该成员，此时我们就可以将该成员定义成静态成员。

1.1 静态成员变量

1> 定义格式：在定义成员变量前加关键字static那么此成员变量就是静态成员变量

2> 一般定义权限为public，必须在全局处进行声明，声明时一般给定初始值，没有给定初始值时默认为0

3> 静态成员变量，不依附于某个类对象，不占用对象的空间，在编译时已经分配空间。

4> 虽然独立于类之外，但是她也还是成员变量，每个类对象都共同拥有该成员，通过某一个类对象进行更改该成员，虽有对象的该成员都会被更改

5> 从功能上说，起到全局变量的作用，但是又属于类中的成员，其他类不能进行访问，从这个角度而言，使用静态成员变量比使用全局变量更加能体现封装性。

6> 由于静态成员不需要依附于某个类对象，所有也可以直接通过类名进行访问：

类名::静态成员

```
#include <iostream>

using namespace std;

class Stu
{
private:
    int num;            //编号
    int age;

public:
    static int flag;

    Stu() {}
    Stu(int n, int a):num(n), age(a) {}
```

```
16      ~Stu() {}
17
18      void show()
19      {
20          cout<<"num = "<<num<<"    age = "<<age<<"    flag = "
    <<flag<<endl;
21      }
22  };
23
24  int Stu::flag = 520;          //在全局处声明静态成员变量    如果不给定
    初始值，默认为0
25
26
27  int main()
28  {
29      cout<<sizeof(Stu)<<endl;          //8
30      Stu s1(9527, 30);
31      s1.show();
32
33      Stu s2(3399, 50);
34      s2.show();
35
36      s1.flag = 1314;           //通过其中一个对象更改静态成员变量
37      s1.show();           //1314
38      s2.show();            //1314
39
40      cout<<Stu::flag<<endl;    //1314
41
42      Stu::flag = 0;
43      s1.show();           //1314
44      s2.show();            //1314
45
46
47      return 0;
48  }
```

1.2 静态成员函数

1> 在成员函数前加关键字static，那么该函数就是静态成员函数
2> 跟静态成员变量一样，不隶属于某个类对象，是属于整个类的
3> 调用静态成员函数也有两种方式：类对象调用、类名直接调用
4> 静态成员函数中只允许访问静态成员变量，不允许访问非静态成员变量
5> 静态成员函数，没有this指针，但是不能与同名同参数的非静态成员函数构成重载关系

```cpp
#include <iostream>

using namespace std;

class Stu
{
private:
    int num;           //编号
    int age;

public:
    static int flag;

    Stu() {}
    Stu(int n, int a):num(n), age(a) {}
    ~Stu() {}

    void show()
    {
        cout<<"num = "<<num<<"   age = "<<age<<"    flag = "<<flag<<endl;
    }

    static void display()
    {
        //cout<<"num = "<<num<<"   age = "<<age<<endl;     //报错，在静态成员函数中不能访问非静态成员变量
        cout<<"flag = "<<flag<<endl;        //允许访问静态成员变量
    }


};

int Stu::flag = 520;          //在全局处声明静态成员变量    如果不给定初始值，默认为0


int main()
{
    cout<<sizeof(Stu)<<endl;          //8
    Stu s1(9527, 30);
    s1.show();

    Stu s2(3399, 50);
    s2.show();
```

```
43
44      s1.flag = 1314;           //通过其中一个对象更改静态成员变量
45      s1.show();              //1314
46      s2.show();               //1314
47
48      cout<<Stu::flag<<endl;     //1314
49
50      Stu::flag = 0;
51      s1.show();              //1314
52      s2.show();               //1314
53      cout<<"****************************************\n";
54
55      s1.display();             //通过类对象调用静态成员函数
56      Stu::display();           //通过类名直接访问静态成员函数
57
58
59      return 0;
60  }
```

二、类的继承（inhert）

2.1 面向对象的三大特性

封装、继承、多态

继承是描述类与类之间关系特征的

2.2 继承

概念:基于一个已有的类，去定义另一个类的过程，就叫做继承

2.3 继承的作用

1> 能实现代码的复用性
2> 是实现多态的必要条件，没有继承，就没有多态

2.4 一个类B继承类A

此时我们称A类为父类、基类
B类为子类、派生类

2.5 继承的格式

class B: 继承方式 classA //此时我们称为B类继承了A类，class可以省略

{

子类扩展的成员

};

2.6 继承方式

1> 回顾访问权限

public: 类内、子类、类外都可以被访问

protected: 类内和子类中能被访问，类外不允许访问

private:类内可以被访问，子类、类外不允许访问

2> 继承方式也有三种：public、protected、private

```
父类        public|protected|private    public|protected|private    public|protected|private
继承方式              prubic                    protected                 private
子类        public|protected|不能访问    protected|protected|不能访问    private|private|不能访问
```

3> 如果不加继承方式，默认为private

4> 常用的继承方式是：public

验证：

```
#include <iostream>

using namespace std;

class A
{
private:
    int aa;
protected:
    int bb;
public:
    int cc;
};

//class B : A       //不加访问权限默认为私有
//class B :protected A       //protected继承
```

```
class B :public A       //public继承
{
public:
    void show()
    {
        //cout<<aa<<endl;       //父类中的私有成员，子类无法访问报错
        cout<<bb<<endl;
        cout<<cc<<endl;
    }
};

int main()
{
    B t;
    cout<<t.aa<<endl;
    cout<<t.bb<<endl;
    cout<<t.cc<<endl;

    return 0;
}
```

2.7 子类中会继承父类中所有成员

子类会继承父类中的所有成员，包括私有成员，只不过私有成员不能被访问，如果非要访问父类中的私有成员，需要在父类中提供public或者protected类型的函数接口，来完成对私有成员的使用。

要完成子类从父类中继承的成员的初始化，需要在子类的构造函数的初始化列表中，显性调用父类的有参构造来完成。在这个过程中，虽然调用了父类的构造函数，但是，仅仅只是用父类的构造函数完成对子类从父类中继承下来的**子类成员的初始化**，并没有实例化父类，所以，最终也就还是只有一个对象存在。

继承关系（is a）是特殊的包含关系(has a)

友元（use a）

如果在子类的初始化列表中没有显性调用父类的构造函数，系统会默认调用父类的无参构造函数来完成对子类从父类中继承成员的初始化。当父类中显性定义了有参构造，但是没有定义无参构造，而且在子类的初始化列表中没有显性调用父类的有参构造，此时会报错。

2.8 当父子类中出现同名的成员时

首先不会报错，在子类实例化对象时，通过子类调用的该成员，用的是子类的空间，但是，如果子类中没有该同名的成员，那么会调用父类中的该成员。如果出现同名并且也还想要调用父类的该成员时，需要加上父类名和作用域限定符。

2.9 继承中构造函数和析构函数调用顺序

先构造父类，再构造子类
先析构子类，再析构父类

```
#include <iostream>

using namespace std;

class Father
{
private:
    int money;        //私房钱

protected:
    int pwd;           //银行卡密码

public:
    Father() {
        cout<<"father::无参构造"<<endl;
    }
    Father(int m, int p):money(m), pwd(p) {
        cout<<"father::有参构造"<<endl;
    }
    ~Father() {
        cout<<"father::析构函数"<<endl;
    }

    void show()
    {
        cout<<"Father::show"<<endl;
        cout<<"money = "<<money<<endl;
        cout<<"pwd = "<<pwd<<endl;
    }
};


class Son : public Father
```

```
{
private:
    string toy;         //玩具
public:
    Son()
    {
        cout<<"Son::无参构造"<<endl;
    }

    Son(int n, int p, string t):Father(n, p),toy(t)
    {

        cout<<"Son::有参构造"<<endl;
    }

    ~Son() {
        cout<<"Son::析构函数"<<endl;
    }

    //子类的show函数
    void show()
    {
        cout<<"Son::show"<<endl;
        //cout<<"money = "<<money<<endl;
        //cout<<"pwd = "<<pwd<<endl;
        Father::show();                 //调用父类的show函数

        cout<<"toy = "<<toy<<endl;
    }

};

int main()
{
    Son s1(5, 123456, "car");


    s1.Father::show();      //调用的是子类中从父类中继承下来的show函数

    return 0;
}

```

三、继承中特殊的成员函数

3.1 构造函数

1> 构造函数不会被继承
2> 需要在子类的初始化列表中，显性调用父类的有参构造来完成对子类从父类中继承下来成员的初始化
3> 如果没有在子类的初始化列表中显性调用父类的有参构造，系统会默认调用父类的无参构造
4> 如果父类没有定义无参构造，会报错
5> 调用顺序：先构造父类，再构造子类

3.2 析构函数

1> 析构函数不会被继承
2> 无论子类中是否显性调用父类的析构函数，在子类对象消亡时，系统都会调用父类的析构函数，来完成对子类从父类中继承成员的销毁
3> 析构顺序：先析构子类，再析构父类

3.3 拷贝构造函数

1> 拷贝构造函数不会被继承
2> 需要在拷贝构造函数的初始化列表中，显性调用父类的拷贝构造函数来完成对子类从父类中继承成员的初始化
3> 如果没有在拷贝构造函数的初始化列表中，显性调用父类的拷贝构造函数，系统会默认调用父类的无参构造
4> 如果父类没有显性定义无参构造，并且有其他构造函数，则会报错
5> 如果父子类中没有指针成员，使用系统提供的拷贝构造函数即可，此时是一个浅拷贝
6> 如果父子类中有指针成员，需要在初始化列表中显性调用父类拷贝构造来完成对指针成员的初始化

3.4 拷贝赋值函数

1> 拷贝赋值函数不会被继承
2> 需要在拷贝赋值函数的函数体，显性调用父类的拷贝赋值函数来完成对子类从父类中继承成员的赋值
3> 如果没有在拷贝赋值函数的函数体中，显性调用父类的拷贝赋值函数，子类从父类继承的 成员为旧值
4> 如果父类没有显性定义无参构造，并且有其他构造函数，则会报错
5> 如果父子类中没有指针成员，使用系统提供的拷贝赋值函数即可，此时是一个浅拷贝

6> 如果父子类中有指针成员，需要拷贝赋值函数体中显性调用父类拷贝赋值函数来完成对指针成员的赋值

```
#include <iostream>

using namespace std;

class Father
{
private:
    int money;        //私房钱

protected:
    int pwd;           //银行卡密码

public:
    Father() {
        cout<<"father::无参构造"<<endl;
    }
    Father(int m, int p):money(m), pwd(p) {
        cout<<"father::有参构造"<<endl;
    }
    ~Father() {
        cout<<"father::析构函数"<<endl;
    }

    //定义父类的拷贝构造函数      父类的指针或者引用可以指向子类的对象
    Father(const Father &other):money(other.money),
pwd(other.pwd) {

        cout<<"father::拷贝构造"<<endl;
    }

    //定义父类的拷贝赋值函数
    Father &operator=(const Father &other)
    {
        this->money = other.money;
        this->pwd = other.pwd;

        cout<<"father::拷贝赋值函数"<<endl;
        return *this;
    }

    void show()
    {
```

```
        cout<<"Father::show"<<endl;
        cout<<"money = "<<money<<endl;
        cout<<"pwd = "<<pwd<<endl;
    }
};


class Son : public Father
{
private:
    string toy;         //玩具
public:
    Son()
    {
        cout<<"Son::无参构造"<<endl;
    }

    Son(int n, int p, string t):Father(n, p),toy(t)
    {

        cout<<"Son::有参构造"<<endl;
    }

    ~Son() {
        cout<<"Son::析构函数"<<endl;
    }

    //子类的拷贝构造函数
    Son(const Son &other):Father(other),toy(other.toy)
    {
        cout<<"Son::拷贝构造"<<endl;
    }

    //子类的拷贝赋值函数
    Son &operator=(const Son &other)
    {
        cout<<"Son::拷贝赋值函数"<<endl;
        this->toy = other.toy;
        Father::operator=(other);         //显性调用父类拷贝赋值函数
        return *this;
    }

    //子类的show函数
    void show()
```

```
    {
        cout<<"Son::show"<<endl;
        //cout<<"money = "<<money<<endl;
        //cout<<"pwd = "<<pwd<<endl;
        Father::show();                 //调用父类的show函数

        cout<<"toy = "<<toy<<endl;
    }

};

int main()
{
    Son s1(5, 123456, "car");   //有参构造
    Son s2(s1);           //拷贝构造
    s2.show();
    Son s3;

    cout<<"**************************************\n";
    s3 = s2;          //调用子类的拷贝赋值函数
    s3.show();
    cout<<"**************************************\n";

    return 0;
}
```

四、多重继承

4.1 含义

一个类可以由多个类同时派生出来，这种继承方式叫多重继承，该子类拥有所有父类的特征

4.2 格式

```
class 子类名：继承方式 父类1， 继承方式 父类2，。。。，继承方式 父类n
{
    子类拓展成员
};
```

```
#include <iostream>

using namespace std;

```

```
class Worker
{
protected:
    int num;        //工号
public:
    Worker() {}
    Worker(int n):num(n) {
        cout<<"Worker::构造函数"<<endl;
    }
};

class Master
{
protected:
    string degree;       //学位
public:
    Master() {}
    Master(string d):degree(d) {
        cout<<"Master::构造函数"<<endl;
    }
};

//父类的构造函数调用顺序跟继承顺序有关
class Engineer:public Worker, public Master
{
private:
    string name;
public:
    Engineer(){
        cout<<"Engineer::无参构造"<<endl;
    }

    //父类的构造函数调用顺序跟调用构造函数的顺序无关
    Engineer(int n, string d, string name):Worker(n),
Master(d), name(name){
        cout<<"Engineer::无参构造"<<endl;
    }

    void show()
    {
        cout<<num<<endl;
        cout<<degree<<endl;
        cout<<name<<endl;
    }
};
```

```
49
50  int main()
51  {
52      Engineer e(9527, "博士", "zhangpp");
53      e.show();
54
55      return 0;
56  }
```

4.3 多重继承中成员同名的情况

```
1   #include <iostream>
2
3   using namespace std;
4
5   class A
6   {
7   protected:
8       int num;
9   public:
10      A() {}
11      A(int n):num(n) {}
12  };
13
14  class B
15  {
16  protected:
17      int num;
18  public:
19      B() {}
20      B(int n):num(n) {}
21  };
22
23  class C : public A, public B
24  {
25
26  public:
27      C() {}
28      C(int a, int b):A(a), B(b) {}
29
30      void show()
31      {
32          //cout<<num<<endl;      //报错，因为不知道num从哪个父类中继
    承下来的
```

```
        cout<<A::num<<endl;
        cout<<B::num<<endl;
    }
};


int main()
{
    C c(520, 1314);
    c.show();
    return 0;
}

```

五、虚继承

5.1 菱形继承(钻石继承)

一个公共基类，派生多个中间子类，又由多个中间子类共同派生出一个汇聚子类，那么在汇聚子类中，会保留多份公共基类的成员，访问起来会有歧义，该继承方式便是菱形继承

```
            A                公共基类
          /   \
         B     C             中间子类
          \   /
            D                 汇聚子类
```

```
#include <iostream>

using namespace std;

class A
{
protected:
    int value_a;
public:
    A() {cout<<"A::无参构造"<<endl;}
    A(int a):value_a(a) {cout<<"A::有参构造"<<endl;}
};

class B:public A
{
protected:
```

```
    int value_b;
public:
    B(){cout<<"B::无参构造"<<endl;}
    B(int a, int b):A(a), value_b(b){cout<<"B::有参构造"
<<endl;}
};

class C:public A
{
protected:
    int value_c;
public:
    C(int a, int c):A(a), value_c(c) {cout<<"C::有参构造"
<<endl;}
    C(){cout<<"C::无参构造"<<endl;}
};

class D:public B, public C
{
public:
    D(){}
    D(int aa, int bb, int cc):B(aa, bb), C(aa, cc){cout<<"D::
有参构造"<<endl;}

    void show()
    {
        cout<<B::value_a<<endl;
        cout<<C::value_a<<endl;
        //cout<<value_a<<endl;          //报错    因为每条路都会有一
个value_a
        cout<<value_b<<endl;
        cout<<value_c<<endl;
    }

};


int main()
{
    cout << "Hello World!" << endl;
    return 0;
}
```

如何结构菱形继承问题？需要用到虚继承，所以，虚继承是解决菱形继承问题的
能够保证在继承过程中，只保留一份公共基类的成员

5.2 虚继承格式

只需要在生成中间子类的继承方式之前加关键字virtual,那么该继承方式便是虚继承

说明：在正常的子类的构造函数中，只需要调用直接父类的有参构造即可完成对子类成员的初始化，但是在虚继承中，因为编译器不能确定公共基类从哪一条线上将成员传递给汇聚子类，索性，哪条路都不走。

此时想要完成对汇聚子类从公共基类继承下来成员的初始化，需要在汇聚子类的初始化列表中，显性调用公共基类的有参构造，来完成对汇聚子类从公共基类继承下来这一份成员的初始化，如果没有显性调用公共基类的有参构造，系统会默认调用公共基类的无参构造。

```
#include <iostream>

using namespace std;

class A
{
protected:
    int value_a;
public:
    A() {cout<<"A::无参构造"<<endl;}
    A(int a):value_a(a) {cout<<"A::有参构造"<<endl;}
};

class B:virtual public A
{
protected:
    int value_b;
public:
    B(){cout<<"B::无参构造"<<endl;}
    B(int a, int b):A(a), value_b(b){cout<<"B::有参构造"
<<endl;}
};

class C:virtual public A
{
protected:
    int value_c;
public:
    C(int a, int c):A(a), value_c(c) {cout<<"C::有参构造"
<<endl;}
    C(){cout<<"C::无参构造"<<endl;}
```

```
};

class D:public B, public C
{
public:
    D(){}
    D(int aa, int bb, int cc, int dd):A(dd),B(aa, bb), C(dd, cc){cout<<"D::有参构造"<<endl;}

    void show()
    {

        cout<<value_a<<endl;        //
        cout<<value_b<<endl;       //2
        cout<<value_c<<endl;       //3
    }

};


int main()
{
    D t(1,2,3,4);
    t.show();
    return 0;
}
```

作业

定义一个学生类（Student）：私有成员属性（姓名、年龄、分数）、成员方法（无参构造、有参构造、析构函数、show函数）

再定义一个党员类（Party）：私有成员属性（党组织活动，组织），成员方法（无参构造、有参构造、析构函数、show函数）。

由这两个类共同派生出学生干部类，私有成员属性（职位），成员方法（无参构造、有参构造、析构函数、show函数），使用学生干部类实例化一个对象，然后调用其show函数进行测试