# 一、虚函数

定义: 在类的定义中, 前面带有 virtual 关键字的成员函数就是虚函数。如下:

```
1 class Student{
2  public:
3     virtual int get();
4 };
5 int Student::get(){}
```

virtual 只在声明函数时使用,定义时则不用加

# 二、多态的表现形式1.0

- 派生类的指针可以赋给基类指针。
- 通过基类指针调用基类和派生类中的同名虚函数时
  - 1. 若该指针指向一个基类的对象,那么被调用是基类的虚函数。
  - 2. 若该指针指向一个派生类的对象,那么被调用的是派生类的虚函数

这种机制就叫做"多态"

```
1 class CBase{
    public:
     virtual void SomeVirtualFunction(){}
 5 class CDerived:public CBase{
      public:
 7
     virtual void SomeVirtualFunction(){}
8
   };
9 int main()
10 {
     CDerived ODerived;
11
     CBase * p = \&ODerived;
12
13
     p -> SomeVirtualFunction(); //调用那个函数取决与p指向的那种类型的对象
14
     return 0;
15 }
```

## 三、多态的表现形式2.0

- 派生类的对象可以赋给基类使用。
- 通过基类引用调用基类和派生类中的同名虚函数时:
  - 1. 若这个引用引用的是一个基类的对象,那么被调用的就是基类的虚函数。
  - 2. 若这个引用**引用的是一个派生类的对象**,那么被调用的就是**派生类的虚函数**。

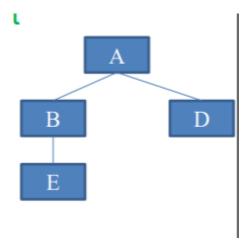
#### 这种机制也是多态

```
1 class CBase{
2  public:
3  virtual void SomeVirtualFunction(){}
4 };
```

```
5 class CDerived:public CBase{
  6
        public:
  7
        virtual void SomeVirtualFunction(){}
  8
    };
 9
    int main()
 10
     CDerived ODerived;
 11
 12
       CBase & p = ODerived;
 13
        p.SomeVirtualFunction(); //调用那个函数取决与p是那种类型的引用
 14
        return 0;
 15 }
```

#### 多态的举例:

```
1 #include<bits/std++.h>
   using namespace std;
 3
   class A{
 4
      public:
 5
           virtual void Print(){
 6
               cout << "A.Print" << endl;</pre>
 7
 8
   };
 9
    class B : public A{
10
     public:
11
        virtual void Print(){
12
               cout << "B.Print" << endl;</pre>
13
       }
14
    };
15
   class D : public A{
16
      public:
17
            virtual void Print(){
               cout << "D.Print" << endl;</pre>
18
19
            }
20
   };
21 class E : public B{
      public:
22
23
           virtual void Print(){
24
                cout << "E.Print" << endl;</pre>
25
            }
26 };
27
   int main()
28
29
        A a; B b; E e; D d;
30
      A * pa = &a;
31
        B * pb = \&b;
32
        D * pd = &d;
33
        E * pe = \&e;
34
        pa->Print();
35
        pa = pb;
36
        pa-> Print();
37
        pa = pd;
38
        pa-> Print();
39
        pa = pe;
40
        pa-> Print();
41
        return 0;
42
   }
43 //输出结果:
```



### 四、例题

几何形体处理程序: 输入若干个几何形体的参数,要求按面积排序输出。输出时要指明形状。 Input: 第一行是几何形体数目n(不超过100).下面有n行,每行以一个字母c开头. 若 c 是 'R',则代表一个矩形,本行后面跟着两个整数,分别是矩形的宽和高; 若 c 是 'C',则代表一个圆,本行后面跟着一个整数代表其半径 若 c 是 'T',则代表一个三角形,本行后面跟着三个整数,代表三条边的长度

Output: 按面积从小到大依次输出每个几何形体的种类及面积。每行一个几何形体。

```
1 | Sample Input:
2
   3
3 R 3 5
4
   C 9
5
   T 3 4 5
6
7
   Sample Output
8
9
   Triangle:6
10 Rectangle:15
11 | Circle:254.34
```

#### AC代码:

```
1 #include<iostream>
 2
   #include<cstring>
 3
   #include<cstdio>
   #include<cmath>
 5
   using namespace std;
    class shape{
 6
 7
      public:
            virtual double area() = 0;
 8
9
            virtual void print() = 0;
10
   };
11
    shape * p[50];
    class rectangle : public shape{
12
13
        public:
            int w,h;
14
```

```
15
      virtual double area();
16
           virtual void print();
17
    };
18
   double rectangle :: area(){
19
       return w * h;
20 }
21
    void rectangle :: print(){
22
        cout << "Rectangle:" << area() << endl;</pre>
23 }
24
   class circle : public shape{
25
      public:
26
           int r;
27
           virtual double area();
28
           virtual void print();
29
   };
   double circle :: area(){
30
31
      return r * r * 3.14;
32
   }
   void circle :: print(){
33
34
       cout << "Circle:" << area() << endl;</pre>
35 }
   class triangle : public shape{
36
      public:
37
38
           int a,b,c;
39
           virtual double area();
           virtual void print();
40
41
   };
42
   double triangle :: area(){
      double p = (a + b + c)/2.0;
43
        return sqrt(p * (p - a) * (p - b) * (p - c));
45
    void triangle :: print(){
46
        cout << "Triangle:" << area() << endl;</pre>
47
48 }
49
   int main()
50
51
       int n;
52
      rectangle * pr;
53
       circle * pc;
54
        triangle * pt;
55
        cin >> n;
56
        for(int i = 1; i <= n; i++){
57
            char c;
58
            cin >> c;
59
            switch(c){
60
                case'R':
61
                    pr = new rectangle();
62
                    cin >> pr->w >> pr->h;
63
                    p[i] = pr;
64
                    break;
                case'C':
65
66
                    pc = new circle();
67
                    cin >> pc->r;
68
                    p[i] = pc;
69
                    break;
                case'T':
70
71
                    pt = new triangle();
72
                    cin >> pt->a >> pt->b >> pt->c;
```

```
73
                       p[i] = pt;
74
                       break;
75
              }
76
         }
77
         for(int i = 1; i \le n; i++){
78
              p[i]->print();
79
         for(int i = 1; i < n; i++){
80
81
              if(p[i]\rightarrow area() > p[i + 1]\rightarrow area()) cout << 1;
82
              else cout << 0 << ' ';
         }
83
84
         return 0;
85 }
```

注意:用基类数组存放指向各种派生类对象的指针,然后遍历数组,就能对各个派生类对象进行各种操作,这是一种很常见的做法

# 五、其他例子及其说明

```
#include<bits/stdc++.h>
 2
   using namespace std;
 3
   class Base {
 4
        public:
            void fun1() { fun2(); }//{this->fun2()} this是基类指针, fun2是虚函数,
    所以是一个多态
           virtual void fun2() { cout << "Base::fun2()" << endl; }</pre>
 6
 7
    };
    class Derived:public Base {
 8
9
      public:
10
            virtual void fun2() { cout << "Derived:fun2()" << endl; }</pre>
11 | };
12
   int main()
13
14
        Derived d;
15
        Base * pBase = & d;
16
        pBase->fun1();
17
        return 0;
18 }
```

注意:在构造函数和析构函数中调用虚函数,不是多态。编译时即可确定,调用的函数是自己的类或基 类中定义的函数,不会等到运行时才决定调用自己的还是派生类的函数。

```
1 #include<bits/stdc++.h>
    using namespace std;
 2
 3
    class myclass{
        public:
 4
 5
             virtual void hello(){
                 cout << "hello from myclass" << endl;</pre>
 6
 7
 8
            virtual void bye(){
 9
                 cout << "bye from myclass" << endl;</pre>
10
             }
11
    };
12
    class son : public myclass{
13
        public:
```

```
void hello(){
14
15
                 cout << "hello from son" << endl;</pre>
             }
16
17
             son(){
18
                 hello();
19
             }
20
             ~son(){
21
                 bye();
22
             }
23
24
    class grandson : public myclass{
25
        public:
26
             void hello(){
27
                 cout << "hello from grandson" << endl;</pre>
28
29
             void bye() { cout << "bye from grandson"<<endl;}</pre>
30
             grandson(){cout<<"constructing grandson"<<endl;}</pre>
31
             ~grandson(){cout<<"destructing grandson"<<endl;};
32 };
33
   int main()
34
35
        grandson gson;
36
        myclass * pson;
37
        pson = &gson;
38
        pson->hello();//多态
39
        return 0;
40
    }
```

#### 注意:派生类中和基类中虚函数同名同参数表的函数,不加virtual也自动成为虚函数

```
class Base {
2
        private:
 3
            virtual void fun2() { cout << "Base::fun2()" << endl; }</pre>
4
   };
5
   class Derived:public Base {
6
        public:
7
            virtual void fun2() { cout << "Derived:fun2()" << endl; }</pre>
8
    };
9
        Derived d;
        Base * pBase = & d;
10
11
        pBase -> fun2();//error
```

编译出错是因为 fun2() 是 Base 的私有成员。即使运行到此时实际上调用的应该是 Derived 的公有成员 fun2() 也不行,因为语法检查是不考虑运行结果的。

如果将 Base 中的 private 换成 public ,即使 Derived 中的 fun2() 是 private 的,编译依然能通过,也能正确调用 Derived::fun2()。

可以理解为:对于派生类的私有成员我们可以通过多态访问,但是对于基类的私有成员则不可以