专题十二 多态与虚函数、模板

【内容预览】

【知识清单】

12.1、多态与虚函数

一、多态的定义与分类

多态按字面的意思就是多种形态。在 C++中，多态分为两种：

1.编译时的多态：通过函数重载和运算符重载来实现，是静态的。

2.运行时的多态：通过类继承关系和虚函数来实现，是动态的。

本专题讨论运行时的多态。

例 12-1

```
#include <iostream>
using namespace std;
class Shape {
   protected:
      int width, height;
   public:
      Shape( int a=0, int b=0){
         width = a;
         height = b;
      }
      int area(){
         cout << "Parent class area :" <<endl;
         return 0;
      }
};
class Rectangle: public Shape{
   public:
```

```
    Rectangle( int a=0, int b=0):Shape(a, b) { }
    int area (){
       cout << "Rectangle class area :" <<endl;
       return (width * height);
    }
};
class Triangle: public Shape{
   public:
    Triangle( int a=0, int b=0):Shape(a, b) { }
    int area (){
       cout << "Triangle class area :" <<endl;
       return (width * height / 2);
    }
};
int main( ){
   Shape *shape;
   Rectangle rec(10,7);
   Triangle tri(10,5);
   shape = &rec;
   shape->area();
   shape = &tri;
   shape->area();
   return 0;
}
```

对于这个程序，我们期望的输出是：

```
Rectangle class area
Triangle class area
```

但实际的输出却是：

```
Parent class area
Parent class area
```

这是因为，调用函数 area() 被编译器设置为基类中的版本，即函数调用在程序执行前就准备好了。为了得到正确的输出，我们在 Shape 类的 area()函数前加上一个关键字 virtual，即将其声明为虚函数：

```
    class Rectangle: public Shape{
       public:
          ……
          virtual int area (){
             ……
          }
    };
```

再编译运行就可以得到正确的结果了。

二、虚函数

定义虚函数的一般形式为：

```
virtual 返回类型 函数名(参数表){
                ……
}
```

定义虚函数需注意以下几点：

1.无论虚函数是在类内定义还是在类内声明类外定义，都只用在声明时将其定义为虚函数。

2.虚函数具有继承性。基类中声明了虚函数，派生类中自动成为虚函数。

3.在派生类中重新定义虚函数时不仅要重名，参数表和返回类型也必须和基类中的虚函数完全相同，否则会被认为是重载而不是虚函数。

4.虚函数仅适用于有继承关系的类对象，只有类的成员函数才能说明为虚函数。类的静态成员函数为一个类的对象所共有，因此不能声明为虚函数。

5.因为在调用构造函数时对象还没有完成创建，所以构造函数不能定义为虚函数；但析构函数可以定义为虚函数。

三、纯虚函数与抽象类

有时在基类中无法对虚函数给出有意义的实现，则可将其定义为纯虚函数。定义纯虚函数的一般形式为：

```
virtual 返回类型 函数名(参数表) = 0;
```

含有纯虚函数的类无法被实例化，称为抽象类；如果派生类不重新定义虚函数，则派生类也是抽象类。

【解题技巧】

例12-2以下程序的执行结果为：

```
#include<iostream>
class a{
   public:
virtual void print(){
   cout << "a prog..." << endl;
}
};
class b:public a
{};
class c:public b{
   public:
void print(){
   cout << "c prog..." << endl;
}
};
void show(a *p){
   (*p).print();
}
int main(){
   a a;
   b b;
   c c;
   show(&a);
   show(&b);
   show(&c);
```

```
    return 0;
}
```

正解：a prog...
　　a prog...
　　c prog...
分析：考查多态。a 类对象调用本身的虚函数，b 类因为没有覆写 print，所以仍然调用基类的虚函数。而 c 类重新定义 print 虚函数，所以调用 c 类的 print。

例 12-3 下列关于纯虚函数和抽象类的描述中，不正确的是（ ）

A. 纯虚函数是一个没有具体实现的虚函数
B. 抽象类是包括纯虚函数的类
C. 抽象类只能作为基类，其纯虚函数的实现在派生类中给出
D. 可以定义一个抽象类的对象

正解：D
分析：抽象类不能实例化。

12.2、模板

一、模板的定义

模板是泛型编程的基础，泛型编程即以一种独立于任何特定类型的方式编写代码。例如一个函数 add()：

```
int add(int a, int b){
    return a + b;
}
```

只能实现 int 类型变量的加法，如果不知道输入变量类型的话则需要重复写很多个功能相同的函数重载，会极大地增加工作量；而使用模板就可以很好的减轻工作量。

二、函数模板

定义模板函数的一般形式为：

```
template <typename T> 返回类型 函数名(参数列表){
                ……
}
```

其中 T 是类型的占位符名称，它可以替代为任意一种类型。

例 12-4

```
#include <iostream>
#include <string>
using namespace std;
template <typename T>T Max (T a, T b) {
    return a < b ? b : a;
}
int main (){
    int i = 39;
    int j = 20;
    cout << "Max(i, j): " << Max(i, j) << endl;
```

```
    double f1 = 13.5;
    double f2 = 20.7;
    cout << "Max(f1, f2): " << Max(f1, f2) << endl;
    string s1 = "Hello";
    string s2 = "World";
    cout << "Max(s1, s2): " << Max(s1, s2) << endl;
    return 0;
}
```

则输出结果为：

```
Max(i, j): 39
Max(f1, f2): 20.7
Max(s1, s2): World
```

三、类模板

和定义模板函数一样，我们也可以定义一个模板类：

```
template <class T> class 类名{
        ……
};
```

例 12-5 栈的类模板

```
template <class T>
class Stack{
    public:
        Stack(int MaxStackSize = 100);
        Stack<T>& Push(const T&x);
        Stack<T>& Pop(T&x);
    private:
        int m_top;
        int m_MaxSize;
        T *m_stack;
};
```

【解题技巧】

例12-6在下面程序横线处填上适当字句，以使该程序执行结果为：

50 4 34 21 10

0 7.1 8.1 9.1 10.1 11.1

```
#include <iostream>
using namespace std;
template <class T>
void f (___________)
{___________;
for (int i=0;i<n/2;i++)
t=a［i］, a［i］=a［n-1-i］, a［n-1-i］=t;
}
void main ()
{int a［5］={10,21,34,4,50};
double d［6］={11.1,10.1,9.1,8.1,7.1};
```

```
f(a,5);f(d,6);
for (int i=0;i<5;i++)
cout <<a［i］<< "";
cout <<endl;
for (i=0;i<6;i++)
cout << d［i］  << "";
cout << endl;
}
```

正解：T a[], int n
　　T t=0;
分析：不同的数据类型的调用，使用了模板。f 函数增加 t 变量，因为实参类型不同，所以 t 的应该是 T 类型。