

8.1

多态的字面意思就是多种状态，在面向对象的程序设计中，一个接口，多种实现即为多态。c++的多态性具体体现在编译和运行两个阶段。编译时多态是静态多态，在编译时就可以确定使用的接口。运行时多态是动态多态，具体引用的接口在运行时才能确定。

8.2

含有纯虚函数的类是抽象类。抽象类作为可从中派生更具体的类的一般概念的表达式。一定要给出，因为抽象类本身的纯虚函数不含实现，不给出实现无法使用。

8.3

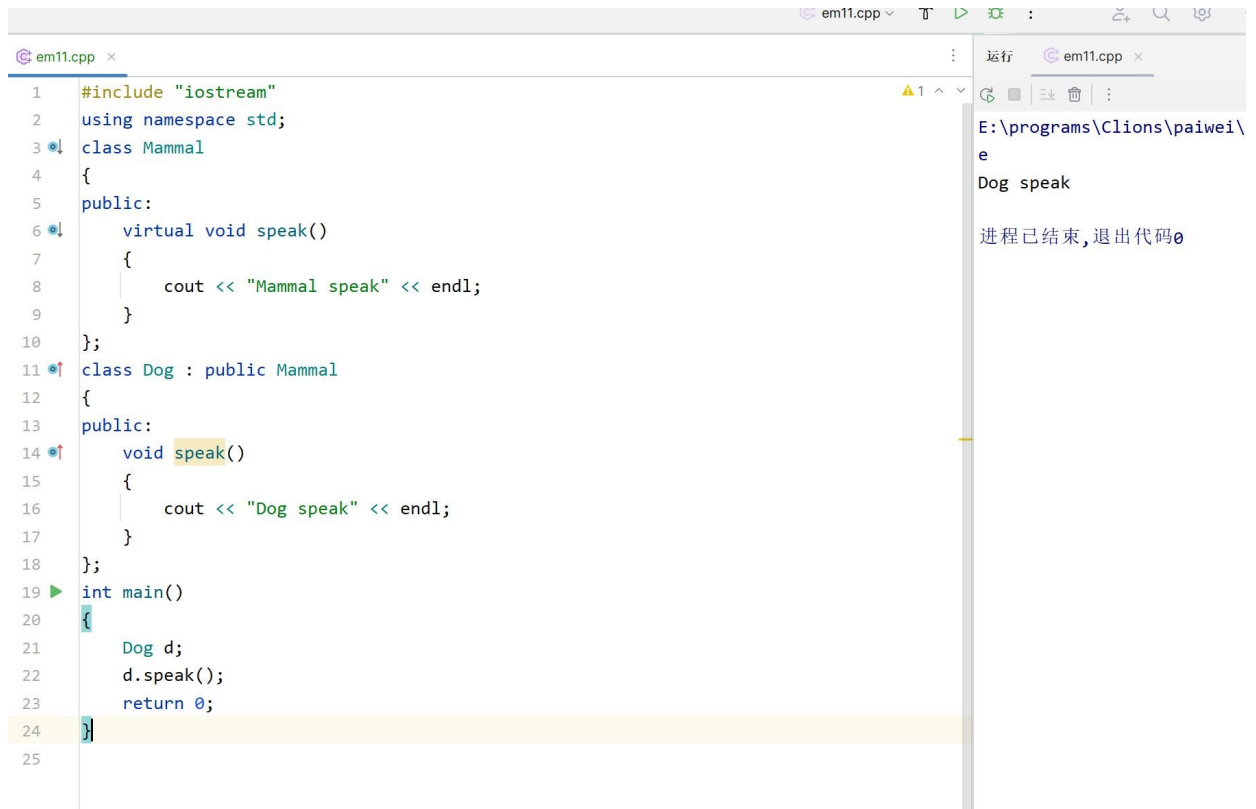
不能声明虚构造函数，构造函数必须为非虚。虚函数的作用在于通过父类的指针或者引用，在调用它的时候能够通过动态链调用子类重写的虚成员函数。而构造函数是在创建对象时是系统自动调用的，不可能通过父类或者引用去调用，因此就规定构造函数不能是虚函数。

可以（一般都会）声明虚析构函数，这样使用父类指针或引用指向派生类时，析构可以直接析构掉派生类，而不是只析构基类的部分，造成内存泄露。

8.4

```
1  class counter
2  {
3  private:
4      int count;
5
6  public:
7      counter() { count = 0; }
8      counter(int c) { count = c; }
9      counter operator+(counter c)
10     {
11         counter temp;
12         temp.count = count + c.count;
13         return temp;
14     }
15     counter operator+(int c)
16     {
17         counter temp;
18         temp.count = count + c;
19         return temp;
20     }
21     int getCount() { return count; }
22 };
```

8.5



The screenshot shows a C++ IDE with a file named `em11.cpp`. The code defines a `Mammal` class with a `virtual void speak()` method that prints "Mammal speak". A `Dog` class inherits from `Mammal` and overrides the `speak()` method to print "Dog speak". The `main` function creates a `Dog` object `d` and calls `d.speak()`. The output window on the right shows the execution path: `E:\programs\Clions\paiwei\em11.cpp`, `Dog speak`, and `进程已结束,退出代码0` (Process ended, exit code 0).

```
1 #include "iostream"
2 using namespace std;
3 class Mammal
4 {
5 public:
6     virtual void speak()
7     {
8         cout << "Mammal speak" << endl;
9     }
10 };
11 class Dog : public Mammal
12 {
13 public:
14     void speak()
15     {
16         cout << "Dog speak" << endl;
17     }
18 };
19 int main()
20 {
21     Dog d;
22     d.speak();
23     return 0;
24 }
```

8.6

```
1 class Shape
2 {
3 public:
4     virtual double getarea() = 0;
5     virtual double getperim() = 0;
6     virtual ~Shape(){};
7 };
8 class Rectangle : public Shape
9 {
10 private:
11     double len;
12     double wid;
13
14 public:
15     Rectangle(double l, double w) : len(l), wid(w) {}
16     double getarea() { return len * wid; }
17     double getperim() { return 2 * (len + wid); }
18     ~Rectangle() {}
19 };
20 class Circle : public Shape
21 {
22 private:
23     double r;
24     const double pi = 3.1415926;
```

```

25
26 public:
27     Circle(double r) : r(r) {}
28     double getarea() { return pi * r * r; }
29     double getperim() { return 2 * pi * r; }
30     ~Circle() {}
31 };

```

8.7

```

1  class Point
2  {
3  private:
4      int x, y;
5
6  public:
7      Point(int x = 0, int y = 0) : x(x), y(y) {}
8      Point &operator++();
9      Point operator++(int);
10     Point &operator--();
11     Point operator--(int);
12     int getX() const { return x; }
13     int getY() const { return y; }
14 };
15 Point &Point::operator++()
16 {
17     x++;
18     y++;
19     return *this;
20 }
21 Point Point::operator++(int)
22 {
23     Point old = *this;
24     ++(*this);
25     return old;
26 }
27 Point &Point::operator--()
28 {
29     x--;
30     y--;
31     return *this;
32 }
33 Point Point::operator--(int)
34 {
35     Point old = *this;
36     --(*this);
37     return old;
38 }
39

```

8.8

```
1  #include "iostream"
2  using namespace std;
3
4  class BaseClass
5  {
6  public:
7      virtual void fn1() { cout << "BaseClass::fn1()" << endl; }
8      void fn2() { cout << "BaseClass::fn2()" << endl; }
9  };
10
11 class DerivedClass : public BaseClass
12 {
13 public:
14     void fn1() { cout << "DerivedClass::fn1()" << endl; }
15     void fn2() { cout << "DerivedClass::fn2()" << endl; }
16 };
17
18 int main()
19 {
20     DerivedClass d;
21     BaseClass *p1 = &d;
22     DerivedClass *p2 = &d;
23     p1->fn1();
24     p1->fn2();
25     p2->fn1();
26     p2->fn2();
27     return 0;
28 }
```



```
运行 em11.cpp ×
E:\programs\Clions\paiwei\em11.exe
DerivedClass::fn1()
BaseClass::fn2()
DerivedClass::fn1()
DerivedClass::fn2()
进程已结束,退出代码0
```