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。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编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

```cpp
class counter
{
private:
    int count;

public:
    counter() { count = 0; }
    counter(int c) { count = c; }
    counter operator+(counter c)
    {
        counter temp;
        temp.count = count + c.count;
        return temp;
    }
    counter operator+(int c)
    {
        counter temp;
        temp.count = count + c;
        return temp;
    }
    int getCount() { return count; }
};
```

8.5

em11.cpp

```cpp
#include "iostream"
using namespace std;
class Mammal
{
public:
    virtual void speak()
    {
        cout << "Mammal speak" << endl;
    }
};
class Dog : public Mammal
{
public:
    void speak()
    {
        cout << "Dog speak" << endl;
    }
};
int main()
{
    Dog d;
    d.speak();
    return 0;
}
```

运行 em11.cpp

```
E:\programs\Clions\paiwei\
e
Dog speak

进程已结束,退出代码0
```

8.6

```cpp
class Shape
{
public:
    virtual double getarea() = 0;
    virtual double getperim() = 0;
    virtual ~Shape(){};
};
class Rectangle : public Shape
{
private:
    double len;
    double wid;

public:
    Rectangle(double l, double w) : len(l), wid(w) {}
    double getarea() { return len * wid; }
    double getperim() { return 2 * (len + wid); }
    ~Rectangle() {}
};
class Circle : public Shape
{
private:
    double r;
    const double pi = 3.1415926;
```

```

public:
    Circle(double r) : r(r) {}
    double getarea() { return pi * r * r; }
    double getperim() { return 2 * pi * r; }
    ~Circle() {}
};
```

8.7

```
class Point
{
private:
    int x, y;

public:
    Point(int x = 0, int y = 0) : x(x), y(y) {}
    Point &operator++();
    Point operator++(int);
    Point &operator--();
    Point operator--(int);
    int getX() const { return x; }
    int getY() const { return y; }
};
Point &Point::operator++()
{
    x++;
    y++;
    return *this;
}
Point Point::operator++(int)
{
    Point old = *this;
    ++(*this);
    return old;
}
Point &Point::operator--()
{
    x--;
    y--;
    return *this;
}
Point Point::operator--(int)
{
    Point old = *this;
    --(*this);
    return old;
}

```

8.8

```cpp
#include "iostream"
using namespace std;

class BaseClass
{
public:
    virtual void fn1() { cout << "BaseClass::fn1()" << endl; }
    void fn2() { cout << "BaseClass::fn2()" << endl; }
};

class DerivedClass : public BaseClass
{
public:
    void fn1() { cout << "DerivedClass::fn1()" << endl; }
    void fn2() { cout << "DerivedClass::fn2()" << endl; }
};

int main()
{
    DerivedClass d;
    BaseClass *p1 = &d;
    DerivedClass *p2 = &d;
    p1->fn1();
    p1->fn2();
    p2->fn1();
    p2->fn2();
    return 0;
}
```