8-1

多态性指同样的消息被不同类型的对象接收时导致不同的行为。

c++通过编译和运行来实现多态，包括重载多态，强制多态，包含多态和参数多态。

8-2

带有纯虚函数的类是抽象类。

抽象类的主要作用是通过它为一个类族建立一个公共的接口，使他们能够更有效地发挥多态特性。

抽象类的派生类若给出所有纯虚函数的实现，这个派生类就可以定义自己的对象，因而不再是抽象类；而如果没有给出全部纯虚函数的实现，这是派生类仍是一个抽象类。

8-3

在类的定义中用 virtual 关键字来限定成员函数，即声明虚函数。

在 c++中不能声明虚构造函数，但能声明虚析构函数。

通过基类指针调用对象的析构函数，就需要让基类的析构函数成为虚函数，否则会造成内存泄漏。

8-4

```
class counter{
  public:
    counter operator + (const counter &c)const{
      return counter(number+c.number);
    }
  private:
    int number;
};
```

8-5

```
#include <iostream>
using namespace std;
class mammal{
public:
    mammal(int x):m(x){}
    virtual void speak(){
        cout<<"mammal"<<endl;
    }
    ~mammal(){}
private:
    int m;
};
class dog:public mammal{
public:
    dog(int x,int y):mammal(x),n(y){}
    void speak(){
        cout<<"dog"<<endl;
    }
    ~dog(){}
private:
    int n;
```

```
};

int main(){

    dog dd(1,2);

    dd.speak();

}
```

调用的是 dog 里的 speak 函数

8-6

```
#include <iostream>

using namespace std;

class shape{

public:

    shape(){}

    ~shape(){}

};

class rectangle:public shape{

public:

    rectangle(int x,int y):l(x),w(y){}

    void getarea(){

        int s=l*w;

        cout<<s<<endl;

    }

    void getperim(){
```

```
        int c=2*(l+w);
        cout<<c<<endl;
    }
    ~rectangle(){}
private:
    int l;
    int w;
};
class circle:public shape{
public:
    circle(int x):r(x){}
    void getarea(){
        double s=3.14*r*r;
        cout<<s<<endl;
    }
    void getperim(){
        double c=2*3.14*r;
        cout<<c<<endl;
    }
    ~circle(){}
private:
    float r;
```

```
};

int main(){

    rectangle rr(2,3);

    circle cc(3);

    rr.getarea();

    cc.getperim();

}
```

8-7

```
#include <iostream>

using namespace std;

class point{

public:

  point(int a,int b):x(a),y(b){}

  point &operator ++ (){

    x=x+1;

    y=y+1;

   return *this;

  }

  point operator ++ (int){

    point pp = *this;

   ++(*this);
```

```
    return pp;
  }
  point &operator -- (){
    x=x-1;
    y=y-1;
    return *this;
  }
  point operator --(int){
    point pp=*this;
    --(*this);
    return pp;
  }
  void show(){
    cout<<x<<" "<<y<<endl;
  }
private:
  int x,y;
};
int main(){
    point p(1,1);
    ++p;
    p.show();
```

```
    p++;
    p.show();
    p--;
    p.show();
    --p;
    p.show();
}
```

8-8

```
#include <iostream>
using namespace std;
class baseclass{
public:
    baseclass(){}
    virtual void fn1(){
        cout<<"base fn1"<<endl;
    }
    void fn2(){
        cout<<"base fn2"<<endl;
    }
    ~baseclass(){}
```

```
};
class derivedclass:public baseclass{
public:
    derivedclass(){}
    void fn1(){
        cout<<"derive fn1"<<endl;
    }
    void fn2(){
        cout<<"derived fn2"<<endl;
    }
    ~derivedclass(){}
};
int main(){
    derivedclass d;
    baseclass *p=&d;
    derivedclass *pp=&d;
    p->fn1();
    p->fn2();
    pp->fn1();
    pp->fn2();
}
```

分析：派生类的虚函数覆盖了基类的虚函数，用 baseclass 指针 p 调用虚函数时发生动态绑定

```
or-bqinfjcv.54n   --pid=Microsoft-MIEngine-Pid-frpdraud.don   --db
xe=D:\mingw64\bin\gdb.exe' '--interpreter=mi'
derive fn1
base fn2
derive fn1
derived fn2
```