C++程序设计

主讲：王老师

尚德机构

学习是一种信仰

教材介绍

C++程序设计

（2008年版）

编著：刘振安

机械工业出版社

考试题型

单选题　　1分×20题 = 20分

填空题　　1分×20题 = 20分

程序改错题　4分×5题 = 20分

完成程序题　4分×5题 = 20分

程序分析题　5分×2题 = 10分

程序设计题 10分×1题 = 10分

第八章　多态性和虚函数

目录

第8章
多态性和虚函数

§8.1 多态性

静态联编所支持的多态性称为**编译时的多态性**，当调用重载函数时，编译器可以根据调用时所使用的实参在**编译时**就确定下来应调用哪个函数；

动态联编所支持的多态性称为**运行时的多态性**，由**虚函数**实现。虚函数类似于重载函数，但与重载函数的实现策略不同，即对虚函数的调用使用动态联编。

一、静态联编中的赋值兼容性及名字支配规律

例8.1 静态联编时的名字支配规律及赋值兼容性

```
#include <iostream>
using namespace std;
const double PI=3.14159;
class Point {
private:
        double x,y;
public:
        Point(double i,double j) {x=i;y=j;}
        double area() {return 0;}
};
class Circle:public Point {
private:
        double radius;
public:
        Circle(double a,double b,double r):Point(a,b){radius=r;}
        double area(){return PI*radius*radius;}
};
```

```
void main(){
        Point a(1.5,6.7);
        Circle c(1.5,6.7,2.5);
        cout<<"点面积为"<<a.area()<<endl;  //输出"点面积为0"
        cout<<"圆面积为"<<c.area()<<endl;  //输出"圆面积为19.6349"
        Point * p=&c;  //派生类对象的地址赋给指向基类的指针
        cout<<"圆面积为"<<p->area()<<endl;  //调用基类的area函数, 输出"圆面积为0"
        Point &rc=c;    //派生类对象初始化基类的引用
        cout<<"圆面积为"<<rc.area()<<endl;  //调用基类的area函数, 输出"圆面积为0"
        Circle * p1=&c;  //派生类对象的地址赋给指向派生类的指针
        cout<<"圆面积为"<<p1->area()<<endl;  //调用派生类area函数, 输出"圆面积为19.6349"
        Circle &rc1=c;  //派生类对象初始化派生类的引用
        cout<<"圆面积为"<<rc1.area()<<endl; //调用派生类area函数, 输出"圆面积为19.6349“
}
```

```
点面积为0
圆面积为19.6349
圆面积为0
圆面积为0
圆面积为19.6349
圆面积为19.6349
```

二、动态联编的多态性

如果让编译器动态联编，即编译语句“Pint * p=&c;”时，只根据兼容性规则检查它的合理性，也就是只检查是否符合“派生类对象的地址可以赋给基类的指针”这一条件，至于“p->area()”调用哪个函数，**等程序运行到该位置时再决定，则需要通过虚函数来实现，这就是动态联编。**

如上例中，如想让类Point的指针指向派生类函数area的地址，通过虚函数实现的过程为：使用关键字virtual将Point类的area函数声明为虚函数，语句为“virtual double area(){return 0;}”。当编译系统编译含有虚函数的类时，将为它建立一个虚函数表，表中的每一个元素都指向一个虚函数的地址，此外，编译器也将为类增加一个数据成员，这个数据成员是一个指向该虚函数表的指针，通常称为vptr。

二、动态联编的多态性

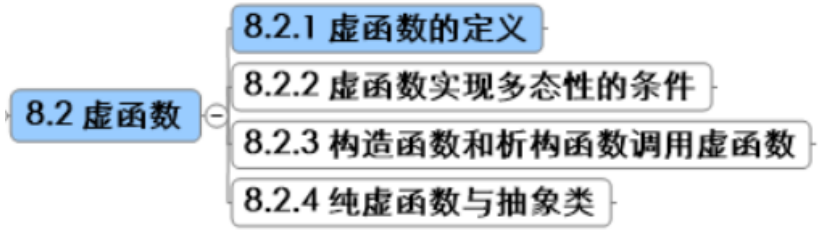
如果派生类Circle没有重写这个area虚函数，则派生类的虚函数表里的元素所指向的地址就是基类Point的虚函数area的地址，即派生类仅继承基类的虚函数，它调用的也是基类的area函数。如果派生类Circle改写了这个area虚函数，如语句“virtual double area(){return PI*radius*radius;}”，则此时编译器也将派生类虚函数表里的元素指向派生类的area函数的地址，当程序运行到语句“p->area();”时，就自动调用派生类的area函数，这就实现了让类Point的指针指向派生类函数area的地址。

由此可见，虚函数的地址翻译取决于对象的内存地址，虚函数的调用规则是：**根据当前对象，优先调用对象本身的虚成员函数。**

小结

小结：**派生类能继承基类的虚函数表，而且只要是和基类同名（参数相同）的成员函数，无论是否使用virtual声明，它们都自动成为虚函数。**如果派生类没有改写继承基类的虚函数，则函数指针调用基类的虚函数；如果派生类改写了基类的虚函数，编译器将重新为派生类的虚函数建立地址，函数指针会调用改写过的虚函数。

§8.2 虚函数

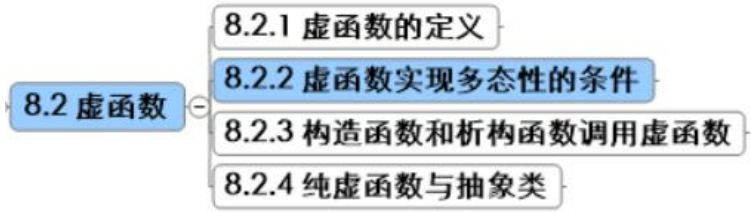


一旦基类定义了虚函数，该基类的派生类中的同名函数也自动成为虚函数。

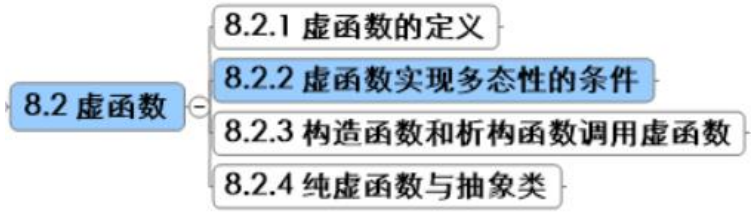
一、虚函数的定义

虚函数只能是类中的一个成员函数，但不能是静态成员，关键字**virtual**用于类中该函数的声明中。当在派生类中定义了一个同名的成员函数时，只要该成员函数的参数个数和相应类型及其返回类型与基类中同名的虚函数完全一样，则无论是否为该成员函数使用virtual，它都将成为一个虚函数。

二、虚函数实现多态性的条件



关键字**virtual**指示C++编译器对调用虚函数进行**动态联编**，这种**多态性是程序运行到需要的语句处才动态确定的，所以称为运行时的多态性**。不过，使用虚函数并不一定产生多态性，也不一定使用动态联编，例如在调用中对虚函数使用成员名限定，可以强制C ++对该函数的调用使用静态联编。



产生运行时的多态性有三个前提

第一，类之间的继承关系满足赋值兼容性规则；

第二，改写了同名虚函数；

第三，根据赋值兼容性规则使用指针或引用。

满足前两条并不一定产生动态联编，必须同时满足第三条才能保证实现动态联编。第三条又分两种情况：第一种是已经演示过的按赋值兼容性规则使用基类指针或引用访问虚函数；第二种是把指针或引用作为函数参数，即这个函数不一定是类的成员函数，可以是普通函数，而且可以重载。

例2、分别使用指针和引用的display函数

```
#include <iostream>
using namespace std;
const double PI=3.14159;
class Point {
private:
        double x,y;
public:
        Point(double i,double j) {x=i;y=j;}
        virtual double area(){return 0;}
};
class Circle:public Point {
private:
        double radius;
public:
        Circle(double a,double b,double r):Point(a,b){radius=r;}
        double area() {return PI*radius*radius;}
};
void display(Point *p){cout<<p->area()<<endl;}
void display(Point&a){cout<<a.area()<<endl;}
```

```
void main(){
    Point a(1.5,6.7);
    Circle c(1.5,6.7,2.5);
    Point *p=&c;  //派生类对象的地址赋给基类指针
    Point &rc=c;  //派生类对象初始化基类引用
    display(a);  //基类对象调用基类虚函数area，输出0
    display(p);  //指针调用派生类虚函数area，输出19.6349
    display(rc); //指针调用派生类虚函数area，输出19.6349
}
```

```
0
19.6349
19.6349
```

三、构造函数和析构函数调用虚函数

1、构造函数和析构函数中调用虚函数

在构造函数和析构函数中调用虚函数采用静态联编，即它们调用的虚函数是自己的类或基类中定义的函数，而不是任何在派生类中重新定义的虚函数。

```
#include <iostream>
using namespace std;
class A {
public:
        A(){ }
        virtual void func() {cout<<"构造A"<<endl;}
        ~A(){ }
        virtual void fund() {cout<<"清除A"<<endl;}
};
class B:public A {
public:
        B() {func();}
        void fun() {cout<<"开始…";func();}
        ~B() {fund();}
};
class C:public B {
public:
        C() { }
        void func() {cout<<"类C"<<endl;}
        ~C() {fund();}
        void fund() {cout<<"清除C"<<endl;}
};
```

```
void main() {
        C c;
        c.fun();
}
```

```
构造A
开始…类C
清除C
清除A
```

```
#include <iostream>
using namespace std;
class A {
public:
        A(){ cout<<"构造函数A"<<endl;}
        virtual void func() {cout<<"构造A"<<endl;}
        ~A(){ cout<<"析构函数A"<<endl; }
        virtual void fund() {cout<<"清除A"<<endl;}
};
class B:public A {
public:
        B() {func();}
        void fun() {cout<<"开始…";func();}
        ~B() {fund();}
};
class C:public B {
public:
        C() { cout<<"构造函数C"<<endl; }
        void func() {cout<<"类C"<<endl;}
        ~C() {fund();}
        void fund() {cout<<"清除C"<<endl;}
};
```

```
void main() {
        C c;
        c.fun();
}
```

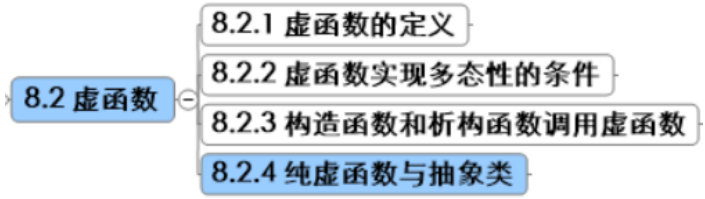
```
构造函数A
构造A
构造函数C
开始…类C
清除C
清除A
析构函数A
```

2、虚析构函数

目前的C + +标准不支持虚构造函数，由于析构函数不允许有参数，所以一个类只能有一个虚析构函数。

虚析构函数使用virtual说明，且只要基类的析构函数被说明为虚函数，则派生类的析构函数，无论是否使用virtual说明，都自动成为虚函数。

运算符new和构造函数一起工作，运算符delete和析构函数一起工作，当使用delete删除一个对象时，delete隐含着对析构函数的一次调用，如果析构函数为虚函数，则这个调用采用动态联编。



四、纯虚函数和抽象类

1、纯虚函数的说明形式

许多情况下，不能在基类中为虚函数给出一个有意义的定义，此时可将它说明为纯虚函数，将其定义留给派生类去做，说明纯虚函数的一般形式为：

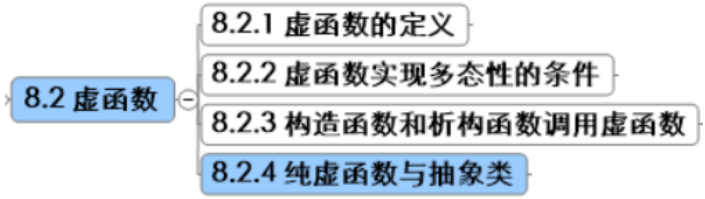
virtual 函数类型 函数名（参数列表）=0;

2、抽象类

一个类可说明多个纯虚函数，包含有纯虚函数的类称为抽象类。一个抽象类只能作为基类来派生新类，不能说明抽象类的对象，但可以说明指向抽象类的指针或引用。抽象类至少含有一个虚函数，而且至少有一个虚函数是纯虚函数，以便将它与空的虚函数区分开来，如下：

```
virtual void area( )=0;    //纯虚函数
virtual void area( ) { };  //空的虚函数
```

从一个抽象类派生的类必须提供纯虚函数的实现代码，或在该派生类中仍将它说明为纯虚函数，否则将出错，此时说明了纯虚函数的派生类仍是抽象类。如果派生类中给出了基类所有纯虚函数的实现，则该派生类将不再是抽象类。



3、类族

3、类族

如果通过同一个基类派生一系列的类，则将这些类总称为类族。抽象类的这一特点保证了进入类族的每个类都具有提供纯虚函数所要求的行为，进而保证了围绕这个类族所建立起来的软件能正常运行。

4、纯虚函数的调用

在成员函数内可以调用纯虚函数，因为没有为纯虚函数定义代码，所以在构造函数或析构函数内调用一个纯虚函数将导致程序运行错误。

例4：编写一个计算正方形、矩形、直角三角形和圆的总面积的程序

此例中shape类的虚函数area仅起到为派生类提供一个一致的接口的作用，派生类中重定义的area用于决定以什么样的方式计算面积。由于在shape类中不能决定以什么样的方式计算面积，所以其中的area函数被说明为纯虚函数。

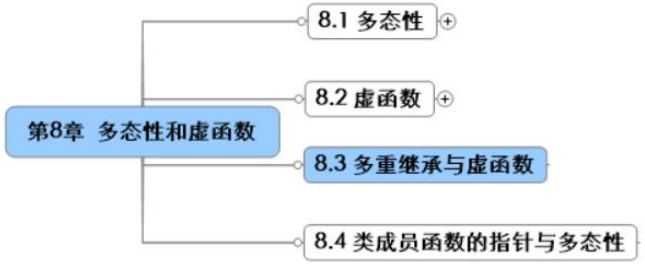
```
#include <iostream>
using namespace std;

class shape {  //抽象类
public:
        virtual double area()=0;  //纯虚函数
};
class square:public shape{  //抽象类派生正方形类
protected:
        double H;
public:
        square(double i) {H=i;}
        double area() {return H * H;}   //重定义纯虚函数
};
class circle:public square{  //正方形类派生圆类
public:
        circle(double r):square(r){ }
        double area() {return H * H * 3.14159;}
};
```

```
class triangle:public square{  //正方形类派生直角三角形类
protected:
        double W;
public:
        triangle(double h,double w):square(h) {W=w;}
        double area() {return H * W *0.5;}
};
class rectangle:public triangle{  //直角三角形类派生矩形类
public:
        rectangle(double h,double w):triangle(h,w) { }
        double area() {return H * W;}
};
double total(shape *s[ ],int n)  //计算总面积函数
{
        double sum=0.0;
        for(int i=0;i<n;i++)
                sum +=s[i]->area();  //复合赋值符，等价于sum=sum+s[i]->area( );
        return sum;
}
```

```
void main(){
        shape *s[5];  //定义shape类对象数组s，其内含有五个元素
        s[0]=new square(4);  //定义正方形类对象
        s[1]=new triangle(3,6);  //定义直角三角形类对象
        s[2]=new rectangle(3,6);  //定义矩形类对象
        s[3]=new square(6);  //定义正方形类对象
        s[4]=new circle(10);  //定义圆形类对象
        for (int i=0;i<5;i++)
                cout<<"s["<<i<<"]="<<s[i]->area()<<"  ";
        double sum=total(s,5);
        cout<<"总面积为:"<<sum<<endl;
}
```

s[0]=16 s[1]=9 s[2]=18 s[3]=36 s[4]=314.159 总面积为:393.159

§8.3 多重继承与虚函数



多重继承可被视为多个单一继承的组合，因此，分析多重继承情况下的虚函数调用与分析单一继承有相似之处。

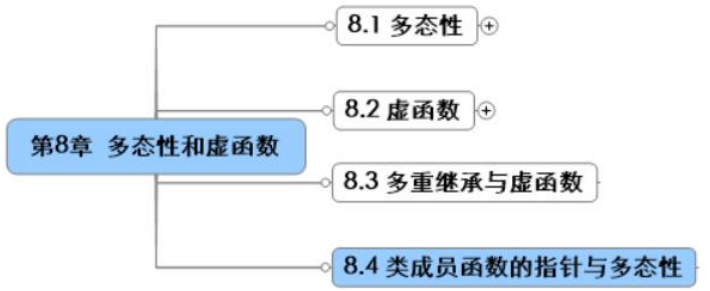
例5、多重继承使用虚函数

```
#include <iostream>
using namespace std;
class A {
public:
        virtual void f() {cout<<"Call A"<<endl;}  //必须使用virtual声明
};
class B {
public:
        virtual void f() {cout<<"Call B"<<endl;}  //必须使用virtual声明
};
class C:public A,public B {
public:
        void f() {cout<<"Call C"<<endl;}  //可省略关键字virtual
};
```

```
Call C
Call C
Call C
```

```
void main() {
        A *pa;
        B *pb;
        C *pc,c;
        pa=&c; pb=&c; pc=&c;
        pa->f();  //输出Call c
        pb->f();  //输出Call c
        pc->f();  //输出Call c
}
```

§8.4 类成员函数的指针与多态性

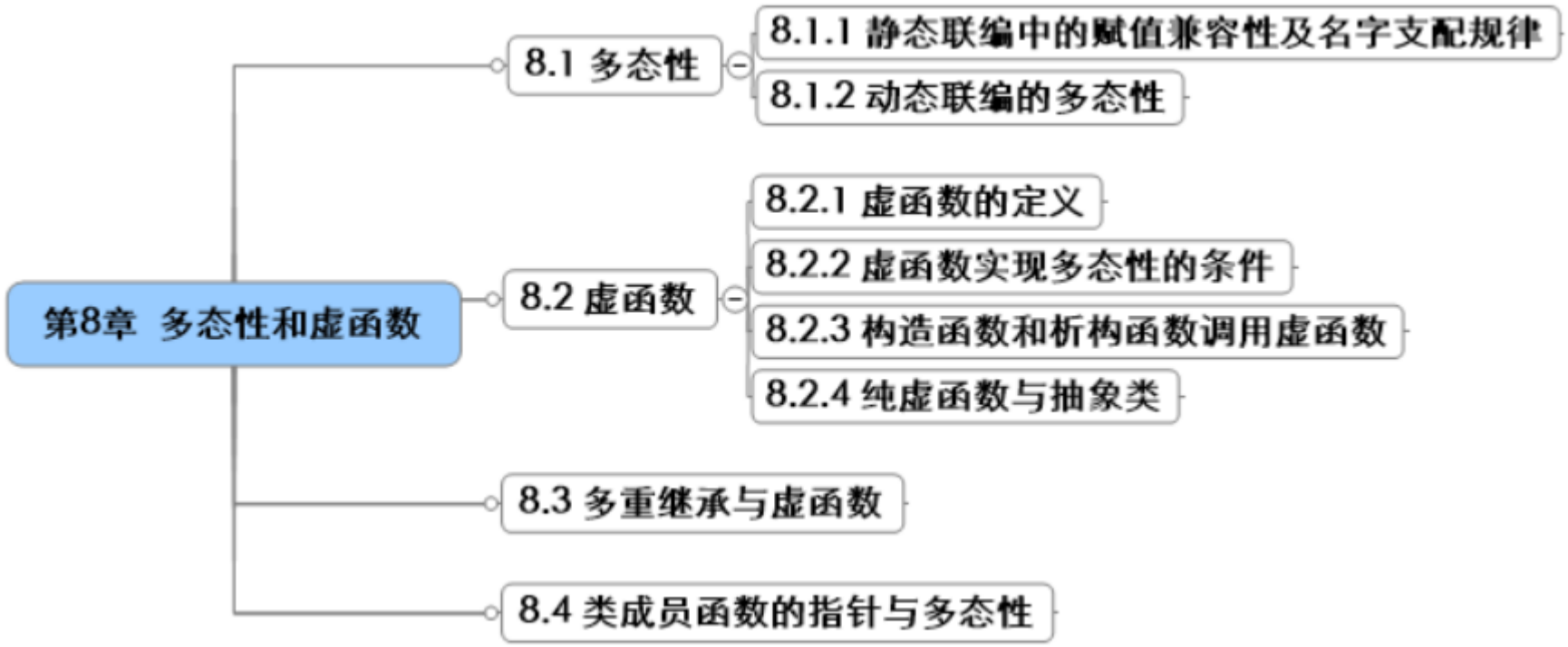


在派生类中，当一个指向基类成员函数的指针指向一个虚函数，并且通过指向对象的基类指针访问这个虚函数时，仍发生多态性。

例6、使用基类成员函数的指针产生多态性

例中display函数有两个参数，第一个参数是基类指针，第二个参数是指向类成员函数的指针（见第5.6节），函数定义是使用基类指针调用指向成员函数的指针所指向的成员函数，至于是调用基类还是派生类的成员函数，取决于基类指针所指向的对象，基类指针指向的对象是基类对象，则调用基类的成员函数，基类指针指向的对象是派生类对象，则调用派生类的成员函数。也可直接将对象的地址作为参数，而不是基类指针。

本章总结


第8章 多态性和虚函数
8.1 多态性
8.1.1 静态联编中的赋值兼容性及名字支配规律
8.1.2 动态联编的多态性
8.2 虚函数
8.2.1 虚函数的定义
8.2.2 虚函数实现多态性的条件
8.2.3 构造函数和析构函数调用虚函数
8.2.4 纯虚函数与抽象类
8.3 多重继承与虚函数
8.4 类成员函数的指针与多态性

```
#include <iostream>
using namespace std;
class base {
public:
        virtual void print() {cout<<"In base"<<endl;}  //虚函数
};
class derived:public base {
public:
        void print() {cout<<"In Derived"<<endl;}  //虚函数
};
void display(base *pb,void(base::*pf) ( ) ) { (pb->*pf) ( );}
 //定义函数display，使用base类的指针pb与指向base类成员函数的指针pf为参数，函数体内执行的语句意思为指针pb调用指针pf指向的成员函数
void main() {
        base b;
derived d;
        base *pb=&d;    //定义基类指针pb，指针指向派生类对象d
        void (base::*pf)();  //声明指向类base的成员函数的指针pf
        pf=base::print;  //指针pf指向base类的具体成员函数print
        display(pb,pf);  //输出“In Derived”，因pb指向派生类对象，优先调用本类的成员函数
   display(&b,base::print);  //输出“In Base”
        display(&d,base::print);  //输出“In Derived”}
```

```
In Derived
In base
In Derived
```

24. 一个抽象类的派生类可以实例化的必要条件是实现了所有的 _____ 。

30. 如果要把 A 类成员函数 f( )且返回值为 void 声明为类 B 的友元函数,则应在类 B 的定义中加入语句 _____ 。

31. C++ 语言的_____提供了与要操作的元素类型无关的算法。

33. 在保护派生中，基类权限为 public 的成员在派生类中为_____。

34. 在函数前面用_____保留字修饰时，则表示该函数为内联函数。

35. 面向对象的四个基本特性是多态性、继承性、封装性、_____ 。

36. 派生类的主要用途是可以定义其基类中_____。

37. 若 int a = 8; int b = (++ a) ++ ; 则 b = _____。

38. 在 C++ 中，利用向量类模板定义一个具有 10 个 int 的向量 A，其元素均被置为 1，实现此操作的语句是 _____ 。

```
#include <iostream.h>
#include <string.h>
class Base
{public:
Base(char *s="\0"){strcpy(name,s);}
void show();
protected:
char name[20];
};
Base b;
void show()
{cout<<"name:"<<b.name<<endl;}
void main()
{Base d2("hello");
show();
}
```

```
class line;
class box
{ private:
int color;
int upx, upy;
int lowx, lowy;
public:
_____
void set_color (int c) {color = c;}
```

```
void define_box (int x1, int y1, int x2, int y2)
{upx = x1; upy = y1; lowx = x2; lowy = y2;}
};
class line
{ private:
int color;
int startx, starty;
int endx, endy;
public:
friend int same_color(line l,box b);
void set_color (int c) {color = c;}
void define_line (_____)
{startx = x1; starty = y1; endx = x2; endy = y2;}
};
int same_color(line l, box b)
{if (l. color == b. color) return 1;
return 0;
}
```

48. 在下面程序的横线处填上适当的字句，使该程序执行结果为40。

```
#include <iostream.h>
class Test
{ public:
_____;
Test (int i=0)
{x=i+x;}
int Getnum()
{return Test::x+10;}
};
_____;
void main()
{Test test;
cout<<test.Getnum()<<endl;
}
```

50. 下面是一个输入半径,输出其面积和周长的 C ++ 程序,在横线处填上正确的语句。

(圆周率为 3.14159)

```
#include" stdafx. h"
#include  < iostream >
#include" math. h"
using namespace std;
_____  //宏定义
void main( )
{double rad;
cout << " rad = " ;
cin >> rad;
double l = 2. 0 * pi * rad;
_____;
cout << " \n The long is:" << l << endl;
cout << " The area is:" << s << endl;}
```

51. #include <iostream.h>

```
class example
{int a;
public:
example(int b=5){a=b++;}
void print(){a=a+1;cout <<a<<" ";}
void print()const
{cout<<a<<endl;}
};
void main()
{example x;
const example y(2);
x.print();
y.print();
}
```

52. #include <iostream.h>

```
class Based
{public:
Based()
{cout << "Based 构造函数\n";fun();}
virtual void fun()
{cout << "Based::fun() 函数\n";}
};
class Derived:public Based
{public:
Derived()
{cout << "Derived 构造函数\n";fun();}
void fun() {cout << "Derived::fun() 函数\n";}
};
void main()
{Derived d;
}
```

19. 下列关于虚函数的描述中，正确的是

A. 使用虚函数就一定产生多态性

B. 虚函数只能是类中的一个成员函数，但不能是静态成员

C. 一个类中仅可以声明一个纯虚函数

D. 在构造函数和析构函数中调用虚函数采用动态联编

41. 以下是对类 Sample 的定义

```
#include < iostream >
using namespace std;
class Sample
{
public:
    Sample( int val) ;
     ~ Sample( ) ;
private:
    float a = 2. 5;
    Sample( ) ;
} ;
```

42.
```
#include <iostream>
using namespace std;
void swap(int &,int &);
void main()
{int a=5,b=10;
swap(a,b);
cout<<"a="<<a<<",b="<<b<<endl;
}
void swap(int x,int y)
{int temp;
temp=x;
x=y;
y=temp;
}
```

43. 此程序改正后的运行结果为 1 2 5 11 21

```
#include < iostream >
using namespace std;
int f( int) ;
int main( )
{
    int i;
    for( i =0;i <5;i ++ )
        cout <<f( i) <<" ";
return 0;
}
int f( int i)
{
    int k =1;
    for( ;i >0;i - - )
        k + =i;
    return k;
}
```

44.
```
#include < iostream >
using namespace std;
int main( )
{ int x = 15;
while( 10 < x < 50)
{ x ++ ;
if( x/3 ) { x ++ ; break; }
}
cout << x << endl;
return 0;
}
```

45. 此程序调用 findmax( ) 函数,返回数组中的最大值

```
#include <iostream>
using namespace std;
int findmax(int *a,int n)
{
    int *p,*s;
    for(p=a,s=a;p-a<n;p++)
        if(*p<*s)*s=*p;
    return (*s);
}
void main()
{
    int x[5]={12,2,8,47};
    cout<<findmax(x,5)<<endl;
}
```

46. 程序在主函数中创建派生类 Derived 的对象 obj，调用 f( ) 函数后输出 DBC

```
#include <iostream>
using namespace std;
class Base
{public:
      virtual void f( ) {cout << "B" ;}
};
```

```
{public:
      Derived( ) {cout << "D" ;}
      virtual void f( ) {Base::f( ) ;cout << "C" ;}
};
```

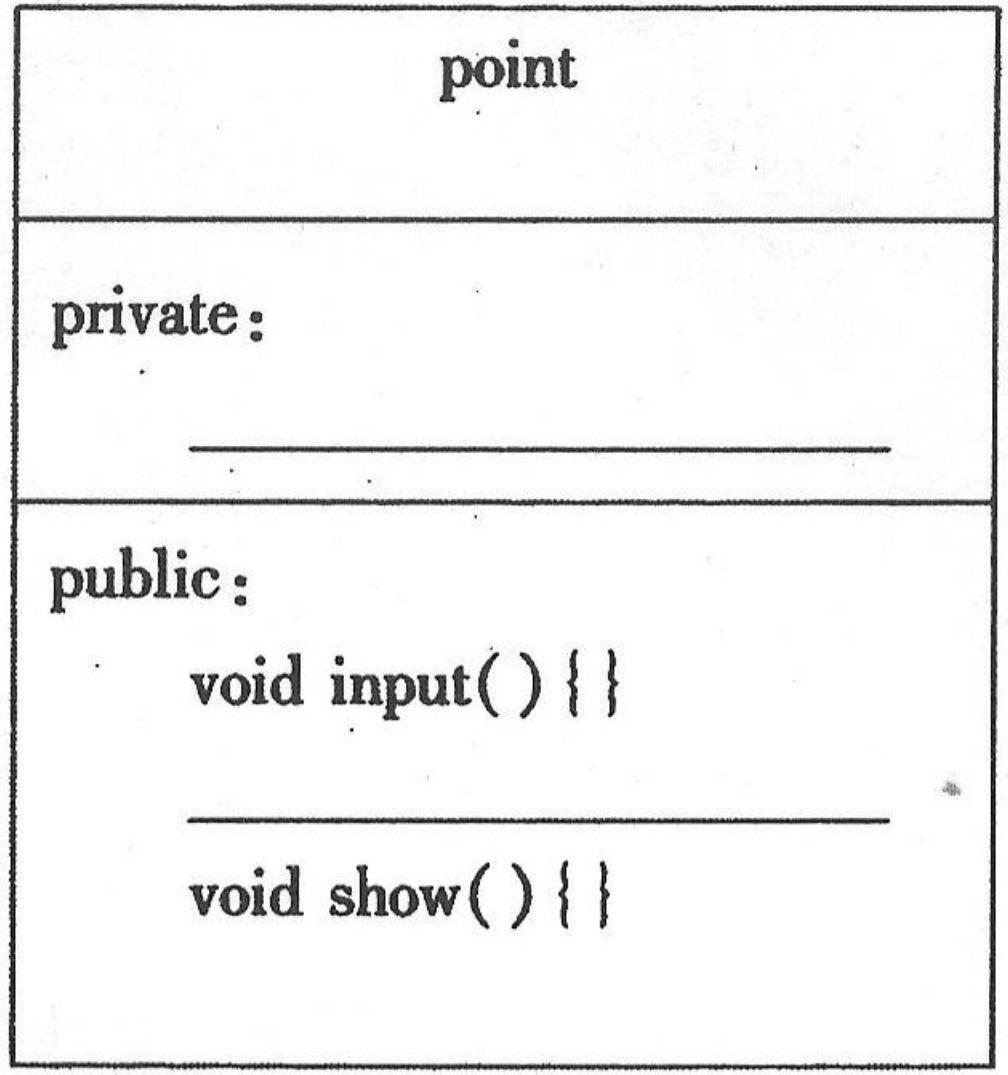
```
int main( )
{Base  * ptr;
Derived obj;
ptr = &obj;
```

```
return 0;
}
```

47. 程序的运行结果为:12

```
#include <iostream>
using namespace std;
class Base
{public:
    int a;
    Base(int i){a=i;}
};
class Derived:public Base
{
    int a;
public:
    Derived(int x):Base(x){}
    void show()
    {
    _____;//输出基类数据成员 a 的值
    }
};
void main()
{
    _____
    d.show();
}
```

定义平面上一个点的类 point，有双精度型私有数据成员横坐标 x 和纵坐标 y，公有成员函数有写入函数 input( )、求原点距离函数 distance( ) 和显示函数 show( )，请完成 point( ) 的类图。



51.
```
#include < iostream >
using namespace std;
void sort( int L[ ] , int n)
{
    int j , k , flag , temp;
    flag = n - 1;
    while( flag > 0)
    { k = flag - 1 ; flag = 0 ;
    for( j = 0 ; j < = k ; j ++ )
        { if( L[ j ] > L[ j + 1 ] )
            { temp = L[ j ] ; L[ j ] = L[ j + 1 ] ;
            L[ j + 1 ] = temp ; flag = j ; }
            }
    }
}
void main( )
{ int array[ 4 ] = { 7 , 2 , 3 , 4 } ;
sort( array , 4 ) ;
cout << "The sorted numbers : " ;
for( int i = 0 ; i < 4 ; i ++ )
    cout << array[ i ] ;
}
```

祝大家顺利通过考试！