1. 选择题

1．下面对于类的描述，是正确的是（ ）

A．类是抽象数据类型的实现

B．类是具有共同行为和属性的若干对象的统一描述体

C．所有的类都能创建对象

D．类就是C语言中的结构类型

2．下列关于C++函数的说明中，正确的是 ( )

A．内联函数就是定义在另一个函数体内部的函数

B．函数体的最后一条语句必须是return语句

C．标准C++要求在调用一个函数之前，必须先声明其原型

D．编译器会根据函数的返回值类型和参数表来区分函数的不同重载形式

3．下列不是描述类的成员函数的是( )

A．构造函数  B．析构函数

C．友元函数  D．拷贝构造函数

4．构造函数不具备的特征的是( )

A．构造函数的函数名与类名相同 B．构造函数可以重载

C．构造函数可以设置默认参数  D．构造函数必须指定类型说明

5．下面有关重载函数的说法中正确的是（ ）

A．重载函数必须具有不同的返回值类型

B．重载函数形参个数必须不同

C．重载函数必须有不同的形参列表

D．重载函数名可以不同

6．下面关于C++中类的继承与派生的说法错误的是（ ）

A．基类的protected成员在公有派生类的成员函数中可以直接使用

B．基类的protected成员在私有派生类的成员函数中可以直接使用

C．有派生时，基类的所有成员访问权限在派生类中保持不变

D．继承可以分为单一继承与多重继承

7．下面关于运算符重载的描述错误的是（ ）

A．运算符重载不能改变操作数的个数、运算符的优先级、运算符的结合性和运算符的语法结构

B．不是所有的运算符都可以进行重载

C．运算符函数的调用必须使用关键字operator

D．在C++语言中不可通过运算符重载创造出新的运算符

8．关于虚函数的描述中，（ ）是正确的。

A．虚函数是一个static类型的成员函数

B．虚函数是一个非成员函数

C．基类中说明了虚函数后，派生类中将其对应的函数可不必说明为虚函数

D．派生类的虚函数与基类的虚函数具有不同的参数个数和类型

9．假定AB为一个类，则执行AB x；语句时将自动调用该类的( )

A．有参构造函数 B．无参构造函数

C．拷贝构造函数 D．赋值构造函数

10．下面关于编写异常处理代码规则中不正确的是( )

A. 可以有数量不限的catch处理程序出现在try块之后，在try 块出现之前不能出现catch块。

B. try块中必须包含throw语句。

C. 在关键字catch之后的圆括号内的数据声明必须包括其类型声明。

D．如果catch中处理程序执行完毕，而无返回或终止指令将跳过后面的catch块继续执行

11．关于类和对象，下列说法不正确的是( )

A.对象是类的一个实例

B.任何一个对象必定属于一个特定的类

C.一个类只能有一个对象

D.类与对象的关系类似于数据类型与变量的关系

12. 下列叙述中，不正确的是( )

A.类的构造函数可以重载 B.类的析构函数可以重载

C.一个类可以不定义构造函数 D.一个类可以不定义析构函数

13.关于静态数据成员，下列描述正确的是( )

1. 在对象创建之前就存在了静态数据成员
2. 静态数据成员只能被静态成员函数访问
3. 不能再构造函数中对静态数据成员赋值
4. 对静态数据成员的访问不受访问权限的限制

14.下列对派生类的描述中，( )是错误的。

1. 一个派生类可以做另一个派生类的基类
2. 派生类至少有一个基类
3. 派生类的成员除了它自己的成员外，还包括了它的基类的成员
4. 派生类中继承的基类成员的访问权限到派生类保持不变

15．关于虚函数的描述中，（ ）是正确的。

A．.基类中说明了虚函数后，派生类中将其对应的函数可不必说明为虚函数

B．虚函数是一个static类型的成员函数

C．虚函数是一个非成员函数

D．派生类的虚函数与基类的虚函数具有不同的参数个数和类型

16.下列叙述，不正确的是( )

1. 纯虚函数是一个特殊的虚函数，它没有具体的实现。
2. 一个基类中说明有纯虚函数，该基类的派生类一定不再是抽象类
3. 抽象类是指具有纯虚函数的类
4. 抽象类只能作为基类来使用，其纯虚函数的实现由派生类给出

17.关于函数模板，描述错误的是 ( )

A．函数模板必须由程序员实例化为可执行的函数模板

B．函数模板的实例化由编译器实现。

C．一个类定义中，只要有一个函数模板，则这个类既可以是类模板，也可以是普通类

1. 类模板的成员函数都是函数模板，类模板实例化后，成员函数也随之实例化

18．在C++中，数据封装要解决的问题是 ( )

A．数据的规范化

B．便于数据转换

C．避免数据丢失

D．防止不同模块之间数据的非法访问

20．假定MyClass为一个类，则执行MyClass a[3],\*p[2];语句时，自动调用该类构造函数 ( )次。

A．2 B．3 C．4 D．5

21．对于下面的类MyClass，在函数f()中将MyClass对象的数据成员n的值修改为50的语句应该是( )

class MyClass

{

public:

MyClass(int x) {n = x;}

void SetNum(int n) {this->n = n;}

private:

int n;

}

int f( )

{

MyClass \*ptr = new MyClass(45);

}

A．MyClass(50); B．SetNum(50);

C．ptr->SetNum(50) ; D．ptr->n = 50;

22．友员运算符\_left>\_right被C++编译器解释为(  )

A．operator >(\_left, \_right)  B． >(\_left, \_right)

C．\_right.operator >(\_left)  D．\_left.operator >( \_right)

23．派生类的构造函数的成员初始化列表中，不能包含( )

A．基类的构造函数

B．派生类中子对象的初始化

C．基类中子对象的初始化

D．派生类中一般数据成员的初始化

24．下面关于C++中类的继承与派生的说法错误的是( )

A．基类的protected成员在公有派生类的成员函数中可以直接使用

B．基类的protected成员在私有派生类的成员函数中可以直接使用

C．私有派生时，基类的所有成员访问权限在派生类中保持不变

D．继承可以分为单一继承与多重继承

25．下列关于虚基类的描述中，错误的是( )

A．虚基类子对象的初始化由最终的派生类完成

B．虚基类子对象的初始化次数与虚基类下面的派生类个数有关

C．设置虚基类的目的是消除二义性

D．带有虚基类的多层派生类构造函数的成员初始化列表中都要列出对虚基类构造函数的调用

26．关于虚函数的描述中正确的是( )

A．虚函数是一个static类型的成员函数

B．虚函数是一个非成员函数

C．虚函数既可以在函数说明时定义，也可以在函数实现时定义

D．派生类的虚函数与基类的虚函数具有相同的参数个数和类型

27．关于纯虚函数和抽象类的描述中错误的是( )

A．纯虚函数是一种特殊的虚函数，它没有具体的实现

B．抽象类是指具有纯虚函数的类

C．一个基类中说明有纯虚函数，该基类的派生类一定不再是抽象类

D．抽象类只能作为基类来使用，其纯虚函数的实现由派生类给出

28．下列对模板的声明，正确的是( )

A. template<T>

B. template<typename T1,T2>

C. template<typename T1,class T2>

D．template<typename T1;typename T2>

29. 已定义了一个类A并有语句A\* pa=new A(5); 那么以下说法正确的是：( )

A. 该语句会创建A类的一个对象,并将此对象的地址赋给指针pa;

B. 该语句会创建A类的5个对象,且它们的地址是连续的;

C. 当指针变量pa超出其作用域时，pa和为对象分配的空间都将被自动释放；

D. 类A中一定有一个显式定义的构造函数和一个显式定义的虚析构函数；

30. 定义类A的非静态成员函数A& f(A& one)时，需有语句return exp;则exp不能是：( )

A.类A中类型为A的静态数据成员

B. f中用语句A a = one; 定义的量a

C. one

D. \*this

31. 在一个类中，下列哪些成员可以是多个？ ( )

A. 无参构造函数

B. 析构函数

C. 原型相同的非静态成员函数

D. 静态成员函数

32. 对类T中的下列成员，若不考虑代码优化，一般来说它们中的哪个执行起来最慢？( )

A. 缺省的无参构造函数

B. 缺省的拷贝构造函数

C. T f( ) { T t; return t; }

D. 不能判定A、B、C哪个最慢

33. 对函数原型为int& min(int& a, int& b);的全局函数，下列语句块哪个是错误的：( )

A. int x=10,y=5; min(x,y)=100;

B. int x=10,y=5, z; z=min(x,y+=x);

C. int x=10,y=5; min(x,10)=y;

D. int x=10,y=5; x=min(x,y);

34. 关于拷贝构造函数错误的说法有：( )

A. 拷贝构造函数是一种特殊的构造函数，且不能在类中进行函数重载。

B. 若自定义派生类的拷贝构造函数，那么也必须自定义基类的拷贝构造函数。

C. 通常拷贝构造函数只能有一个参数，且为常量引用或非常量引用。

D. 拷贝构造函数不允许在函数体中使用return语句。

35. 下列对虚基类声明正确的是 ( )

A. class virtual B: public A

B. class B: virtual public A

C. class B: public Avirtual

D. virtual class B: public A

36. 若有语句A\*const p=new B;，其中类B是从类A直接派生得到的，那么下列说法错误的是( )

A. 若要执行语句p->f( );,那么类A中一定有一个与f( )函数匹配的函数声明。

B. 类B一定是通过public继承方式从类A派生的。

C. 类B的构造函数中，至少有一个无参的或提供全部缺省参数的构造函数。

D. 通过指针p，只能访问类中的常量成员函数或静态成员函数。

37. 下面说明的哪个数据不能作为类T的数据成员定义( )

A. T\* pT

B. class Q{ } q1,q2;

C. static T t;

D. const T t;

38. 下列哪种运算符可以被重载：( )

A. sizeof B. ：： C. ? : D. [ ]

1. 程序改错题，指出程序中的错误语句并分析错误原因

1. 指出下面程序段中的错误，并说明出错原因

class X{

private:

int a=0; //A行

int &b; //B行

const int c; //C行

void setA(int i){a=i;} //D行

X(int i){a=i;} //E行

public:

int X():b(a),c(a){a =0;} //F行

X(int i,int j,int k):b(j),c(k){a=i; } //G行

static void setB(int k){b=k;} //H行

setC(int k)const {c=c+k;} //I行

};

void main()

{

X x1; //J行

X x2(3); //K行

X x3(1,2,3); //L行

x1.setA(3); //M行

}

2.指出下面程序段中的错误，并说明出错原因

#include<iostream>

using namespace std;

class Base1 {

int b1; //A行

public:

Base1(int b1=0) {this->b1=b1;} //B行

void f(){ cout<<"From Base1"<<endl;} //C行

};

class Base2 {

int b2; //D行

public:

Base2(int b2){this->b2=b2;} //E行

void f() { cout<<"From Base2"<<endl;} //F行

};

class Derived: public Base1, public Base2 { //G行

int d;

public:

Derived(int d){this->d=d;} //H行

void g(){ cout<<"From Derived"<<b1<<b2<<endl; } //I行

};

void main(){

Derived dObj(10); //J行

dObj.f(); //K行

dObj.Base1::f(); //L行

}

3.指出下面程序段中的错误，并说明出错原因（6分）

#include<iostream.h>

class A

{ int a;

public:

A(int x){ a=x; } //A行

};

class B:public A //B行

{ int a;

public:

B(int x){ a=x; } //C行

void show(){cout<<a<<'\t'<<A::a<<endl; } //D行

};

void main()

{ B b1; //E行

b1.show(); //F行

}

1. 阅读程序，写出程序的运行结果

1.

#include<iostream>

using namespace std;

class Implementation{

public:

Implementation(int y){value=y;}

void setValue(int v){value=v;}

int getValue() const {return value;}

private:

int value;

};

class Interface{

public:

Interface(int);

void setValue(int);

int getValue() const;

private:

Implementation \*ptr;

};

Interface::Interface(int v):ptr(new Implementation(v)){}

void Interface::setValue(int v){ptr->setValue(v);}

int Interface::getValue() const {return ptr->getValue();}

void main()

{

Interface i(5);

cout<<i.getValue()<<endl;

i.setValue(10);

cout<<i.getValue()<<endl;

}

2.

#include<iostream>

using namespace std;

class B1

{

public:

B1(int a){cout<<"constructing B1 "<<a<<endl;}

};

class B2:public B1{

public:

B2(int b,int a):B1(a){cout<<"constructing B2 "<<b<<endl;}

};

class B3:public B2

{

public:

B3(int a,int b,int c,int d,int e):B2(a,b),memberB2(c,d),memberB1(e)

{cout<<"constructing B3"<<endl;}

private:

B1 memberB1;

B2 memberB2;

};

void main()

{

B3 b3(1,2,3,4,5);

}

3.

#include<iostream.h>

class Character

{

char i;

public:

Character (char a=0){i =a; }

Character operator ++();

Character operator ++(int);

void print(){cout<<i<<endl;}

};

Character Character::operator ++()

{

i++;

return\*this;

}

Character Character::operator ++(int)

{

Character j;

j.i=i++;

return j;

}

void main()

{

Character x(65), y(98), z;

z = ++x;

x.print();

z.print();

z = y++;

y.print();

z.print();

}

4．

#include <iostream>

using namespace std;

enum errs{error0,error1};

double Divide(double test1, double test2)

{

try{

if(test2==0) throw error0;

if(test2>=1000) throw error1;

}

catch(errs er){

switch(er)

{

case error0:

cout<<"除数不能为0!"<<endl;

break;

case error1:

cout<<"除数太大!"<<endl;

break;

}

}

return test1/test2;

}

void main()

{

cout<<Divide(2,0)<<endl;

cout<<Divide(1,1000)<<endl;

}

5.

#include<iostream.h>

class Sample

{

int n;

static int sum;

public:

Sample(int x){n=x; }

void add(){sum+=n;}

void disp()

{

cout << “n=” << n << “,sum=” << sum << endl;

}

};

int Sample::sum=0;//静态数据成员赋初值

void main()

{

Sample a(2),b(3),c(5);

a.add();

a.disp();

b.add();

b.disp();

c.add();

c.disp();

}

6.

#include <iostream>

using namespace std;

class B {

public:

B(){ cout<<"Constructing B"<<endl;}

};

class B1:virtual public B {

public:

B1(int i){ cout<<"Constructing B1"<<endl; }

};

class B2:virtual public B {

public:

B2(int j){ cout<<"Constructing B2"<<endl; }

};

class D: public B1, public B2 {

public:

D(int m,int n): B1(m),B2(n){ cout<<"Constructing D"<<endl; }

};

void main(){

D d(1,2);

}

7.

#include <iostream.h>

class MEMBER {

public:

virtual void answer()

{

cout << "I am a member.\n";

}

};

class TEACHER: public MEMBER {

public:

void answer()

{

cout << "I am a teacher.\n";

}

};

class STUDENT: public MEMBER {

public:

void answer()

{

cout << "I am a student.\n";

}

};

void main()

{

MEMBER member;

TEACHER teacher;

STUDENT student;

MEMBER\* who;

who = &member;

who->answer();

who = &teacher;

who->answer();

who = &student;

who->answer();

}

8.

#include<iostream>

using namespace std;

class A{

protected:

int a;

public:

A(int x){a=x;}

virtual void show(){ cout<<a++<<"\n"; }

void print() { cout<<++a<<"\n"; }

};

class B:public A{

public:

B(int x):A(x){}

void show(){ cout<<a--<<"\n"; }

void print() { cout<<--a<<"\n"; }

};

void main()

{ A a(5),\*p;

B b(10);

p=&a; p->show(); p->print();

p=&b; p->show(); p->print();

}

9.

# include <iostream>

using namespace std;

class MyClass{

public:

MyClass();

void print();

~MyClass();

private:

int i;

static int j;

};

int MyClass::j = 0;

MyClass::MyClass()

{

cout<<"This is a constructor"<<endl;

j += 10;

}

void MyClass::print(){

cout<<"The value of j is"<<j<<endl;

}

MyClass::~MyClass(){

cout<<"This is a destructor!"<<endl;

}

void main(){

MyClass first,second;

first.print();

second.print();

}

10.

#include<string>

#include<iostream>

using namespace std;

class Dog{

string name;

int age;

public:

Dog(string name,int age):name(name),age(age){

cout<<"invoking Dog constructor"<<endl;

}

Dog(const Dog& dog):name(dog.name),age(dog.age){

cout<<"invoking Dog copy constructor"<<endl;

}

};

class Person{

string name;

Dog dog;

public:

Person(string name,Dog dog):name(name),dog(dog){

cout<<"invoking Person constructor"<<endl;

}

};

int main()

{

Dog dog("Fido",4);

Person p1("zaphod",dog);

Person p2 = p1;

return 0;

}

11.

#include <iostream>

using namespace std;

class A {

int a;

public:

A(int i){ cout<<"Constructing A "<<i<<endl; }

};

class B {

public:

B(){ cout<<"Constructing B "<<endl;}

};

class B1: public B ,virtual public A{

public:

B1(int i):A(i){ cout<<"Constructing B1 "<<i<<endl; }

};

class B2:virtual public A,public B {

public:

B2(int j):A(j){ cout<<"Constructing B2 "<<j<<endl; }

};

class D: public B1, public B2 {

public:

D(int m,int n): B1(m),B2(n),a(3),A(4){ cout<<"Constructing D"<<endl; }

A a;

};

void main(){

D d(1,2);

}

12.

# include <iostream>

using namespace std;

class Base{

protected:

int n;

public:

Base(int m){n=m++;}

virtual void g1(){cout<<"Base::g1()..."<<n<<endl;g4();}

virtual void g2(){cout<<"Base::g2()..."<<++n<<endl;g3();}

virtual void g3(){cout<<"Base::g3()..."<<++n<<endl;g4();}

virtual void g4(){cout<<"Base::g4()..."<<++n<<endl;}

};

class Derive: public Base{

int j;

public:

Derive (int n1,int n2):Base(n1){j=n2;}

void g1(){cout<<"Deri::g1()..."<<++n<<endl;g2();}

void g3(){cout<<"Deri::g2()..."<<++n<<endl;g4();}

};

void main(){

Derive Dobj(1,0);

Base Bobj=Dobj;

Bobj.g1();

Base &bobj2=Dobj;

bobj2.g1();

}

13.

#include <iostream.h>

class A {

public:

A(int anInt = 0 ):i(anInt)

{

cout << "A::A( )" << endl;

}

A(const A& anA)

{

cout << “A::A(const A&)” << endl;

i = anA.i;

}

int getI( ) const

{

return i;

}

~A( )

{

cout << "A::~A( )" << endl;

}

private:

int i;

};

class B

{

public:

B( )

{

cout << "B::B( )" << endl;

}

B(const A& anA): a(anA)

{

cout << "B::B(constA&) " << endl;

}

virtual void f( )

{

cout << "B::f( )" << endl; cout << a.getI( ) << endl;

}

virtual ~B( )

{

cout <<"B::~B( )"<<endl;

}

private:

A a;

};

class D : public B

{

public: D( )

{

cout << "D::D( )" << endl;

}

D(const A& anA): a(anA)

{

cout << "D::D(constA&)" << endl;

}

void f( )

{

B::f( ); cout << "D::f( )" << endl; cout << a.getI( ) << endl;

}

~D( )

{

cout << "D::~D( )" << endl;

}

private: A a;

};

void main( )

{

A a(10);

B\* pB = new D(a);

pB->f( );

delete pB;

}

2. 程序填空题

1．在横线处填上适当的字句，完成下面复数类的定义（4分）。

#include<iostream.h>

class Complex

{

private:

double real,image;

public:

Complex(double r=0,double i=0) {real=r;image=i;}

friend Complex operator+(Complex &a,const Complex &b); //复数加法运算符

Complex operator=(Complex c); //复数赋值运算符

};

Complex operator+(Complex &a,const Complex &b)

{

Complex\*t=new Complex(\_\_ \_\_\_);

return \*t;

}

Complex Complex::operator=(Complex c)

{

real=c.real;

image=c.image;

return\_\_\_\_\_;

}

2．下列程序用虚函数print和运行的多态性，把从键盘输入的一个int型数值n，按八进制和十六进制输出，完善程序（8分）。

#include<iostream.h>

class OCT{

protected:

int n;

public:

OCT(int x){ n=x; }

\_\_\_\_\_\_\_\_{ cout<<n<<"的八进制为："<<oct<<n<<endl; }

};

class HEX:public OCT{

public:

HEX(int x):\_\_\_\_\_\_\_\_

void print(){ cout<<n<<"的十六进制为："<<hex<<n<<endl; }

};

void main()

{ int n;

cout<<"请输入一个十进制：";

cin>>n;

OCT oc(n);

HEX he(n);

\_\_\_\_\_\_\_\_;

p=&oc;

p->print();

\_\_\_\_\_\_\_\_;

p->print();

}

3．在横线处填上适当的语句，完成下列程序

#include<iostream>

#include<cmath>

using namespace std;

class Point{

public:

Point(double a,double b,double c){

}

double getX(){return x;}

double getY(){return y;}

double getZ(){return z;}

private:

protected:

};

class Line:Point{

public:

Line(double a,double b,double c,double d) ) {

}

void show(){

cout<<getX()<<endl;

cout<<getY()<<” ”<<y<<endl;

cout<<getZ()<<” ”<<z<<endl;

cout<<k<<endl;

}

private:

double k;

};

int main()

{

Line obj(1.2,3.4,5.6,7.8);

obj.show();

return 0;

}

4．本程序自定义了一个复数类Complex类和一个求最小值的函数模板min，在横线处填上适当的语句，使得该程序的运行结果为:

hello

1+2i

#include<iostream>

#include<string>

#include<complex>

using namespace std;

class Complex{

private:

int real;

int image;

public:

Complex(int,int);

bool operator<(Complex c);

;

};

Complex::Complex(int real = 0,int image = 0){

this->real = real;

this->image = image;

}

bool Complex::operator <(Complex c){

if( )

return true;

else return false;

}

(3) operator<<( ){

;

return os;

}

T \_min(T \_left, T \_right)

{

return \_left < \_right ? \_left : \_right;

}

int main()

{

string str1 = "hello";

string str2 = "hollow";

cout<<\_min(str1,str2)<<endl;

Complex c1(1,2),c2(3,4);

cout<<\_min(c1,c2);

return 0;

}

5．在横线处填上适当的语句，利用异常处理机制合理得处理由主函数的两条调用语句导致的异常，使得：

当调用语句为cout<<Divide(2,0)<<endl;时，输出结果为：除数不能为0!

当调用语句为cout<<Divide(1,1000)<<endl;时，输出结果为：除数太大!

当调用语句为cout<<Divide(1,2)<<endl;时，输出结果为：0.5

#include <iostream>

using namespace std;

enum errs{error0,error1};

double Divide(double dividend, double divided)

{

if(divided == 0) ;

else if(divided >= 1000) ;

else ;

}

void main()

{

try{

cout<<Divide(2,0)<<endl;

cout<<Divide(1,1000)<<endl;

cout<<Divide(1,2)<<endl;

}catch( ){

switch(er)

{

case error0:

cout<<"除数不能为0!"<<endl;

break;

case error1:

cout<<"除数太大!"<<endl;

break;

}

}

}

1. 程序设计题

1.（10分）定义一个大学生类student，函数私有数据成员：姓名、学号、校名，并为它定义带参数的构造函数、参数带缺省值的构造函数和输出数据成员值的print()公有成员函数，另定义研究生类，它以公有继承方式派生于类student，新增加“研究方向、导师名”两个私有数据成员，并定义带参数的构造函数和输出研究生数据的print()公有成员函数。在main()函数中定义基类和派生类对象，对类进行测试。

主函数的测试程序如下：

void main()

{

Student stu1("Li","1600141","XingJiang University");

stu1.print();

GraStudent gstu("Wang","1600240","XJUniversity","Computer","Zhang");

gstu.print();

}

程序运行输出结果如下：

name=Li

StuNum=1600141

universty\_name=XJU

name=Wang

StuNum=1600240

universty\_name=XJU

special is Compute

director is Zhang

2. 定义一个圆类，数据成员有颜色、圆心坐标、半径；成员函数有构造函数（有四个参数），设置圆的各种参数，显示圆的各种参数，计算圆的面积和周长，并编写一个主函数对所定义的圆类进行测试。（12分）

3. 定义一个点类（Point）、矩形类（Rectangle）和立方体类（Cube）的层次结构。Point为基类，成员有点的x坐标和y坐标；Rectangle包括长度和宽度两个新数据成员，Rectangle的位置从Point类继承。Cube类的数据成员包括长度、宽度和高度，长度和宽度从Rectangle类继承。要求各类提供支持初始化的构造函数和显示自己成员的成员函数。编写主函数，测试这个层次结构，输出立方体类的相关信息。（12分）

4.(共20分，每问题10分) 某程序员为了灵活地对各种的给定的曲线函数f(x)画出其曲线图形，设计并部分实现了一个曲线类curve，该类的成员数据中，count代表坐标点的个数，pxs代表的数组存放这些坐标点的横坐标，pys代表的数组存放利用f(x)计算得到的这些坐标点的纵坐标。由于不同曲线的计算公式f(x)是不同的，该程序员希望曲线函数的种类可以通过继承curve类的方式任意增加，增加一个新的f(x)时不改变curve类中的内容，也不改变利用curve类进行图形绘制的算法。已部分完成的curve类定义和实现如下：

class curve {

public:

void setPxs( ) {/\*把获取的横坐标数据存放在pxs代表的数组中，并为count置值\*/ }

double\* getPxs( ) const { return pxs;}

int getCount( ) const { return count;}

double\* getPys( ) const ;

private:

double\* pxs;

double\* pys;

int count

};

请按照该程序员的设计意图给出成员函数getPys的完整实现。实现过程中，可以为curve类增加其它成员。可以假设setPxs函数已经完整实现，不需要考虑曲线的绘制和显示。

5. (共10分）定义一个抽象类Stereo，包含纯虚函数Volume（用来计算体积）。然后派生出立方体Cube类、圆球体Sphere类、圆柱体Cylinder类，分别求其体积。各立体形状的相关数据通过各类的构造函数设置。total( )则是一个通用的用以求不同立体形状的体积总和的全局函数。(圆球体的体积公式为 V=4/3\*PI\*R\*R\*R)。

1）请根据下面主函数代码，给出各个类的定义和实现。

2）请根据下面主函数代码，给出total( )函数的定义和实现。

void main( ) {

Stereo\* s[3];

s[0]=new Cube(5.0,5.0,8.0); //实参为立方体的长、宽、高

s[1]=new Sphere(6.0); //实参为圆球体的半径

s[2]=new Cylinder(5.0,8.0); //实参为圆柱体的底面半径、高

float sum=total(s,3);

cout<<sum<<endl;

for(int i=0;i<3;i++) delete s[i];

}