C++程序填空复习

Test1：

#include<iostream>

#include<cmath>

#include<cstdlib>

using namespace std;

int main(){

double a,b,c;

double delta,x1,x2;

const double zero=1e-9;//定义一个很小的常数

int sign;

cout<<“输入三个系数a(a!=0),b,c:”<<endl;

cin>>a>>b>>c;//输入a、b、c的值

if（fabs(a)<zero）{ //绝对值很小的数即被认为是0

cout<<“二次项系数为0，方程的根是-c/b”；

exit(0);

}

cout<<“a=”<<a<<’\t’<<“b=”<<b<<’\t’<<”c=”<<c<<endl;

delta=b\*b-4\*a\*c;

if(fabs(delta)<zero){

cout<<“方程有两个相等的实根：”；

cout<<“x1=x2=”<<-b/(2\*a)<<endl;

}else{

if(delta>0) sign=1;

else sign=0;

delta=sqrt(fabs(delta));

x1=-b/(2\*a);

x2=delta/(2\*a);

if(sign!=0){

cout<<“方程有两个不同的实根：”；

cout<<”x1=”<<x1+x2<<’\t’<<”x2=”<<x1-x2<<endl;

}else {// sign==0

cout<<“方程有两个不同的复数根：”；

cout<<“x1=”<<x1<<“+i”<<x2<<’\t’<<“x2=”<<x1<<”-I”<<x2<<endl;

}

}return 0;

}

Test2:设计程序将输入夫人百分制成绩转化为五分制输出，90分以上为5分，80~89分为四分，70~79分为3分，60~69分为2分，60分以下为1分.（提示：百分制成绩的分档用整除10的方法实现）

#include <iostream>// [1]

using namespace std;

int main() {

int mark, result; //mark是百分制成绩，result是5分制

cout<<"请输入百分制成绩："<<endl;

cin>>mark;

if(mark<0) {

cout<<"缺考！"<<endl;

return 0;

}

switch(mark/10) { //对百分制成绩进行分档 [2](mark]

case 10:

case 9 ://[3]case 9:

result=5;

cout<<"输出五分制成绩："<<result<<endl;

break;

case 8:

result=4;

cout<<"输出五分制成绩："<<result<<endl;

break; //[4]break;

case 7:

result=3;

cout<<"输出五分制成绩："<<result<<endl;

break;

case 6:

result=2;

cout<<"输出五分制成绩："<<result<<endl;

break;

case 5: case 4: case 3: case 2: case 1: case 0:

result=1;

cout<<"输出五分制成绩："<<result<<endl;

break;

default:

cout<<"输入错误！"<<endl;

}

}

Test3:输入n，求1！+2！+3！+…+n!。（提示：通常求和作为外循环，阶乘作为内循环）

using namespace std;

【1】int main(){

int n, i, jch; //jch是阶乘

double result=0; //result是结果

【2】 cin>>n; //输入n的值

if(n<1) {

cout<<"输入错误！"<<endl;

return 0;

}

result=1;

for(i=2,jch=1; i<=n; 【3】 i++) {

【4】jch=jch\*i; //计算阶乘

result+=jch;

}

cout<<result<<endl;

return 0;

}

Test4：编写一个函数，将一个数插入到已是升序的数组中，且插入后该数组仍是升序数组。已是升序数组的内容由主函数给出，待插入的数在主函数中输入。

#include <iostream>

using namespace std;

int main() {

int i, j, number, a[11]= {1,2,4,6,8,9,12,15,20,30};

cout<<"请输入一个数:";

cin>>number; //输入number的值

cout<<"开始的数组:";

for(i=0; i<10; i++)

cout<<a[i]<<" ";

cout<<endl;

i=0;

while (i<10&&number>=a[i]) i++;

for(j=10; j>=i; j--)

a[j]=a[j-1];

a[i]=number;

cout<<"插入后的数组:";

for(i=0; 【4】i<11; i++)

cout<<a[i]<<" ";

return 0;

}

Test5：编程找出1~500之中满足除以3余2，除以5余3，除以7余2的整数。

【1】#include <iostream>

using namespace std;

int main() {

int i;

for(i=1; 【2】i<=500; i++)

if((i%3==2)&&(i%5==3)&&【3】i%7==2 )

【4】cout<<i<<endl;

return 0;

}

Test6：编程求1000之内的所有完全数。所谓完全数是指一个。例如6=1+2+3，就是一个完全数。（提示：采用穷举法。两两循环，外层是1到999依次处理，内层是求该数的所有因子之和）

#include <iostream>

【1】using namespace std;

int main() {

int i, a, sum; //sum是a的因子和

for(【2】a=1; a<1000; a++) {

sum=0;

for(i=1; i<a; i++)

if(a%i==0 ) 【3】sum+=i; //如果i是a的一个因子，则求因子和

if(【4】sum==a) //如果这个数等于它的所有因子和

cout<<a<<endl;

}

return 0;

}

Test7：设计函数factors(num,k),返回值整数num包含因子k的个数，如果没有该因子，则返回0。

#include <iostream>

using namespace std;

int factors(int num, int k) {

int count=0; //count用来统计个数

while (num%k==0) {

【1】count++;

num/=k;

}

【2】return count;

}

【3】int main(){ //

【4】cout<<"factors(64,3)="<<factors(64,3)<<endl;

return 0;

}

Test8:函数myStrCat(char\*dst,char\*src)将字符串src连接到字符串dst的后面，填写适当的代码，是的mySum（）完成正确的功能。

#include <iostream>

#include <cstring>

using namespace std;

char myStrCat(char \*dst, char \*src) ; //myStrCat函数的声明

int main() {

void dst[100]="Hello, ";

char src[100]="Good luck!";

myStrCat(dst, src);

cout << dst << endl;

return 0;

}

void myStrCat(char \*dst, char \*src) {//【3】\_\_\_\_\_\_

int i=0, len;

len = strlen(dst);

while(src[i]) {

dst[len] = src[i];

len++;

i++;

}

dst[len] = '\0';

}

Test9：定义max函数实现求三个实数中的最大值。

#include<iostream>

using namespace std;

double max(double a, double b, double c) {//【1】\_\_\_\_\_\_

if(a>b&&a>c) return a;

if(b>a&&b>c) return b;//[2]

return c;//[3]

}

int main() {

cout<<max(3.2,7.4,4.5)<<endl;//[4]

return 0;

}

Test10:编写函数getSum()，求整数m的各位数字之和并返回该值。例如m=25时，各位数字之和为9。

#include<iostream>//【1】\_\_\_\_\_\_

using namespace std;

int getSum(int m);

int main() {

int m,s=0;

m=252;

s=getSum(m);//【2】\_\_\_\_\_\_; 求m的各位数字之和并赋值给s

cout<<s;

return 0;

}

int getSum(int m) { //求m的各位数字之和

int s=0,n; //n记录各个位上的数字

while(m!=0) {

n=m%10;//【3】\_\_\_\_\_\_; 求m各个位上的数字

s+=n;

m=m/10;

}

return s;//[4]

}

Test11:编写函数reverse，对给定的10个数置逆序排列。

#include<iostream>

using namespace std;

#define N 10 //[1]符号常量的定义

void reverse(int a[], int len) { //a是待操作的数组，len是数组中元素的数目

int i, t;

for(i=0;i<N/2; i++) {//[2]

t = a[i];

a[i]=a[N-1-i];//[3]

a[N-1-i]=t;

}

}

int main() {

int a[N], k;

for(k=0; k<N; k++)

cin>>a[k];//[4]

reverse(a,N);

for(k=0; k<N; k++)

cout<<a[k]<<endl;

return 0;

}

Test12：函数compare比较两个长度为N的数组是否相等（即两个数组中下标相同的数组元素均相等）。请完成该函数。

#include <iostream>

using namespace std;

void compare(int a[], int b[], int n) {

bool equal = true;//[1]

for(int index=0; index<n; index++) {

if(a[index] != b[index]) {

equal = false;//[2]

//[3]

}

}

if(equal==true)

cout << "Equal!" << endl;

else

cout << "Not equal!" << endl;

}

void input(int a[], int b[], int n) {

int index;

for(index=0; index<n; index++) {//[4]

cin >> a[index];

}

for(index=0; index<n; index++) {

cin >> b[index];

}

}

int main() {

int a[5], b[5];

input(a, b, 5);

compare(a, b, 5);

return 0;

}

Test13：编写一个函数void changeString(char str[]);功能是把其中的大写字母变成小写字母，小写字母变成大写字母，非字母的字符不作变换。变换结果依旧保存在str数组内。大写字母的ASCII码值比对应的小写字母 的ASCII码值小32，如A的ASCII码值比a的ASCII码值小32.

#include <iostream>

using namespace std;

void changeString(char str[]) {

for(int i=0; str[i]!='\0'; i++) {//[1]

if(str[i]>='a'&&str[i]<='z')

str[i]-= 32;//【2】\_\_\_\_\_\_

else if(str[i]>='A'&&str[i]<='Z')

str[i] += 32;

}

}

int main() {

char str[50];//[3]

cout<<"输入要变换的字符串：";

cin>>str;

changeString(str);//[4]

cout<<"变换结果是："<<str<<endl;

return 0;

}

Test14：

#include <iostream>  
#include <cstring>  
using namespace std;  
void conv(char s[]);//conv函数的声明  
int main(){  
char s[10];  
int i;  
cin>>s;  
conv(s);  
cout<<s<<endl;  
return 0;  
}  
void conv(char \*s) {  
int len = strlen(s), j,c;  
for (j=0;j<len/2; j++) {  
char c = s[j];  
s[j]= s[len-1-j];  
s[len-1-j]=c;  
}  
}

Test15：利用冒泡法将10个数按降序排列。

#include<iostream>

using namespace std;

int main() {

int arr[10],i,j,k;

for(i=0;i<10;i++)//遍历数组元素进行赋值[1]

cin>>arr[i];

for(i=0; i<10; i++) {

for(j=0; j<10; j++) {//[2]

if(arr[j]<arr[j+1]) {//[3]

k=arr[j+1];

arr[j+1]=arr[j];

arr[j]=k;

}

}

}

for(i=0; i<10; i++)

cout<<arr[i] <<" ";//[4]

return 0;

}

Test16：声明人民币类IntRMB,私有数据成员包括：元（IYuan）、角（Jiao）、分（Fen），均为整形。类型转换函数将人民币类强制转化为浮点数，以“元”为单位。例如：“10元25角3分”转换成“12.53元”。（参考：类类型可以通过类型转换函数实现类型强制转换，它只能是类的成员函数，不能是友元函数。格式为：operator转换后的数据类型（）{…}使用时的格式为：转换后的数据类型（对象名）；或（转换后的数据类型）对象名；）

#include <iostream>

using namespace std;

class IntRMB { //人民币类

private: //访问属性

int IYuan;

int Jiao;

int Fen;

public:

IntRMB(int y=0,int j=0,int f=0); //构造函数的声明

void print();

operator float() { //float类型转换函数的定义

float temp;

temp=float(IYuan+(Jiao/10.0)+(Fen/100.0));

return temp;

}

};

IntRMB::IntRMB(int y,int j,int f) { //构造函数的实现

IYuan=y;

Jiao=j;

Fen=f;

}

void IntRMB::print() {

cout <<IYuan << "元" <<Jiao << "角" <<Fen <<"分" <<endl;

}

int main() {

float a;

IntRMB m(10,25,3); //定义人民币对象

cout << "\*\*\*转换前\*\*\*" <<endl;

m.print();

a=(float)m

..; //使用重载类型float进行强制类型转换

cout << "\*\*\*转换后\*\*\*" <<endl;

cout<<a<<"圆"<<endl;

return 0;

}

Test17：生命complex类表示负数类型，私有数据成员表示其实部，img表示其虚部。在类中以成员函数的形式定义运算符+的重载函数实现两个复数的加法，并返回结果。

#include <iostream>

using namespace std;

class Complex {

private:

float real;

float img;

public:

Complex(float r=0, float i=0):real(r),img(i) {}

//定义运算符+的成员重载函数，形参是Complex对象的常引用two

Complex operator+(Complex&two){//[1]

return Complex(real+two.real,img+two.img );//[2]

}

void display(){

cout<<"加法的结果为："<<real;//[3]

if(img>=0) cout << "+"; //如果虚部为负数，负数本身有一个"-"号

cout <<img<<"i"<<endl;

}

};

int main() {

Complex one(1.3,-5.2),two(3.7,2.1),result;

result=one+two; //计算复数one和two的加法[4]

result.display();

return 0;

}

Test18：有一个Time类，数据成员有时，分，秒。要求模拟秒表，每次走一秒，满60秒进位，秒又从零开始计数；满60分进位，分又从零开始计数；时使用24小时制，并以“时：分：秒”的格式输出时间。

#include <iostream>

using namespace std;

class Time {

public:

//在Time类的构造函数中使用成员初始化列表初始化数据成员

Time(int h=0, int m=0, int s=0): hour(h),minute(m),sec(s) {}//【1】

Time operator++() { //前置++运算符重载函数

sec++;

if(sec>=60) {//【2】

sec=sec-60;

minute++;

if(minute>=60) {

minute=minute-60;

hour++;

if(hour>24) hour=hour%24;//【3】

}

}

return \*this;

}

Time operator++(int) {//【4】 //后置++运算符重载函数

Time temp(\*this); //保存修改前的对象做返回值

++(\*this);

return temp;

}

void Time::display() {

cout<<hour<<":"<<minute<<":"<<sec<<endl;

}

private:

int hour ;

int minute;

int sec;

};

int main()

{

Time time1(21,34,59), time2;

cout<<" time1 : ";

time1.display();

++time1;

cout<<"++time1: ";

time1.display();

time2 = time1++;

cout<<"time1++: ";

time1.display();

cout<<" time2 : ";

time2.display();

return 0;

}

Test19：利用函数模板设计一个求所有数组元素的和的函数。

#include <iostream>  
using namespace std;  
template<typename T>  
T sum(T a[],int n){//求数组a所有元素的和  
int i;  
T s=0;  
for(i=0;i<n;i++)  
s = s + a[i];  
return s;  
}  
int main() {  
int a[5]={1,2,3,4,5}, s;  
s=sum(a,5); //求数组a所有元素的和  
cout<<s<<endl;  
return 0;  
}

Test20：用函数模板实现3个数值的升序排列。

#include <iostream>

using namespace std;

template <typename T>//[1]

void sort(T a, T b, T c) {

int array[3], temp;//[2]

array[0] = a;

array[1] = b;

array[2] = c;

int i,j;

for(i=0;i<3;i++) {

for(j=0;j<2;j++)

if(array[j]>array[j+1]) { //[3]如果条件为真，则交换两个数的位置

temp = array[j];

array[j] = array[j+1];

array[j+1]=temp;//【4】\_\_\_\_\_\_

}

}

cout<< array[0]<< ' ' << array[1]<< ' ' << array[2]<< endl;

}

int main() {

sort(15,10,9);

return 0;

}

Test21：编写一个程序，该程序使用new运算符建立一个动态的double数组，然后为数组元素赋值，并显示元素值，最后使用delete运算符释放这个动态数组。

#include<iostream>//【1】\_\_\_\_\_\_

using namespace std;

int main() {

int i, n, temp=0;

cout<<"输入数组大小:";

cin>>n; //输入需要的数组大小

double \*array = new double[n];//【2】\_\_\_\_\_\_; //动态申请数组

cout<<"给每个数组元素赋值："<< endl;

for(i=0; i<n; i++) {

cout<<"array["<< i <<"] = ";

cin>>array[i];//【3】\_\_\_\_\_\_; //输入各个数组元素的值

}

cout<<"动态数组个元素的值如下："<< endl;

for(i=0; i<n; i++) {

cout<<"array["<< i <<"] = "<< array[i] << endl; //打印数组元素

}

delete[] array; //[4]释放动态数组

return 0;

}

Test22：设计一个圆类，从圆类派生圆柱，用成员函数输出它们的面积和体积。

#include <iostream>

using namespace std;

class Circular { //圆类

public:

Circular(double a) { //[1]圆类的构造函数

r = a;

area = 3.1415926 \* r \* r;

}

double getArea() { //计算圆面积

return area;

}

private: //[2]数据成员的访问属性

double r;

double area;

};

class Column: public Circular { //从圆类派生圆柱类

protected:

double h;

double cubage;

public:

Column(double a, double b) :Circular(a) { //[3]成员初始化

h = b; //派生类数据成员的赋值

cubage = getArea() \* h;

}

double getCubage() { //计算圆柱体积

return cubage;

}

};

int main() {

Circular circular(45);

Column column(12, 10);

cout<<"圆的面积："<< circular.getArea() << endl; //[4]输出圆的面积

cout<<"圆柱的体积："<< column.getCubage() << endl;

return 0;

}

Test23:

#include<iostream>

using namespace std;

class Box{//类的声明

public:

int wight;

int length;

int high;

void box\_shape(int w,int l,int h);

int box\_volume(int w,int l,int h);

int box\_area(int w,int l,int h);

};

void Box::box\_shape(int w,int l,int h){//判断盒子的形状

if((w==l)&&(l==h))

cout<<“这是一个正方体！”<<endl;

else

cout<<“这是一个长方体！”<<endl;

}

int main(){

Box mybox;//对象定义

cout<<“请输入盒子的长、宽、高：”；

cin>>mybox.width>>mybox.length>>mybox.hight;

//调用成员函数box\_shape判断形状

mybox.box\_shape(mybox.width,mybox.length,mybox.hight;

return 0;

}

Test24设计一个圆类，从圆类派生圆柱，用成员函数输出它们的面积和体积

#include <iostream>

using namespace std;

class Box {

private:

int length;

int width;

int hight;

public: //访问属性 1】\_\_\_\_\_\_ //访问属性

Box(int l,int w,int h) { //构造函数2】\_\_\_\_\_\_

length = l;

width = w;

hight = h;

}

int volume() {

return hight \* width \* length;

}

};

int main() {

Box box1(30, 20, 10); //定义对象box1 【3】\_\_\_\_\_\_ box1(30, 20, 10);

cout << "Box1's volume = " << box1.volume() << endl; //输出box1的体积cout << "Box1's volume = " << 【4】\_\_\_\_\_\_ << endl;

return 0;

}

Test25：声明一个模板，利用他们分别实现两个整数和浮点数的比较，输出最大数。

#include <iostream>

using namespace std;

template <class numtype> //[1]类模板声明

class Compare {

public:

Compare(numtype a,numtype b) { //[2]构造函数定义

x=a;

y=b;

}

numtype max() {//[3]

return (x>y)?x:y;}

numtype min(){

return (x<y)?x:y;}

private:

numtype x,y;

};

int main() {

Compare<int> cmp1(3,4); //定义对象cmp1实现两个整数的比较

cout<<cmp1.max()<<" is the Maximum of two inteder numbers."<<endl;

cout<<cmp1.min()<<" is the Minimum of two inteder numbers."<<endl<<endl;

Compare<float> cmp2(45.78,93.6); //[4]定义对象cmp2实现两个单精度浮点数的比较

cout<<cmp2.max()<<" is the Maximum of two float numbers."<<endl;

cout<<cmp2.min()<<" is the Minimum of two float numbers."<<endl<<endl;

return 0;

}

Test26：建立一个对象数组，内放5个学生的数据（学号，成绩），用指针指向数组的首地址，使用该指针输出第1，3，5个学生的数据。其中，对象数组名必须是stud。

#include<iostream>

using namespace std;

class Student {

public:

Student(int n,int s):num(n) ,score(s){ } //[1]构造函数使用成员初始化列表进行初始化

void display() {

cout<<num<<" "<<score<<endl;

}

private:

int num;

int score;

};

int main() {

Student stud[5] = {Student(1,70) ,Student(2,71), Student(3,72),//[2]

Student(4,73), Student(5,74)};

Student \*p=stud;

for(int i=0; i<=2; i++) { //输出第1，3，5个学生的数据

p->display(); //[3]调用成员函数display

p=p+2; //[4]更新指针变量p

}

return 0;

}

Test27：建立一个对象数组，内放5个学生的数据（学号，成绩），设立一个函数max，用指向对象的指针作函数参数，在max函数中找出5个学生中成绩最高者，并输出其学号。

#include<iostream>

using namespace std;

class Student {

public:

Student(int n,int s):num(n),score(s) {}

int num;

int score;

};

int max(Student\*arr) { //【1】max函数的参数是一个Student类的指针变量arr

float max\_score=arr[0].score; //用max\_ score记录最高的成绩

int i, max\_num=0; //用max\_num记录成绩最高的学生的学号

for(i=0; i<3; i++)

{ if(arr[i].score>max\_score) { //[2]比较大小，并把大值保存在变量max\_score中

max\_score=arr[i].score;

max\_num=i+1; } //[3]更新max\_num

}

cout<<max\_score<<" "<<max\_num<<endl;

return 0;

}

int main() {

Student stud[3] = {Student(1,70),Student(2,71),Student(3,72)};

Student \*p =&stud[0]; //[4]初始化指向对象数组首地址的指针变量

max(p);

return 0;

}

Test28：定义一个复数类Complex，重载运算符“+”，使之能用于复数的加法运算。参加运算的两个运算量可以都是类对象，也可以其中有一个是整数，顺序任意。例如：c1+c2,i+c1,c1+i均合法（设i为整数，c1、c2为复数）。编程序，分别求两个复数之和、整数和复数之和。

#include <iostream>

using namespace std;

class Complex {

private:

double real;

double imag;

public:

Complex() {

real=0;

imag=0;

}

Complex(double r,double i) {

real=r;

imag=i;

}

Complex operator+(Complex &c) { //[1]两个复数的加法，参数使用引用形式

return Complex(real+c.real, imag+c.imag );

}

Complex operator+(int &i) { //复数+整数，参数使用引用

return Complex(i+real,imag);//[2]

}

friend Complex operator+(int &i,Complex &c); //[3]友元函数声明

void display() {

cout<<"("<<real<<","<<imag<<"i)"<<endl;

}

};

Complex operator+(int &i,Complex &c) { //整数+复数

return Complex(i+c.real,c.imag);

}

int main() {

Complex c1(3,4),c2(5,-10),c3; //[4]对象的定义

int i=5;

c3=c1+c2;

cout<<"c1+c2=";

c3.display();

c3=i+c1;

cout<<"i+c1=";

c3.display();

c3=c1+i;

cout<<"c1+i=";

c3.display();

return 0;

}

Test29：有两个矩阵a和b，均为2行3列。求两个矩阵之和。重载运算符“+”，使之能用于矩阵相加。如a+b。

#include <iostream>

using namespace std;

class Matrix { //定义Matrix类

public:

Matrix(); //默认构造函数

friend Matrix operator+(Matrix &,Matrix &); //【1】运算符"+"的友元重载函数

void input(); //输入函数

void display(); //输出函数

private:

int mat[2][3];

};

Matrix::Matrix() { //定义构造函数

for(int i=0; i<2; i++)

for(int j=0; j<3; j++)

mat[i][j]=0;

}

Matrix operator+(Matrix &a,Matrix &b) { //定义重载运算符"+"函数

Matrix c;

for(int i=0; i<2; i++)

for(int j=0; j<3; j++) {

c.mat[i][j]=a.mat[i][j]+b.mat[i][j];

}

return c;//【2】\_\_\_\_\_\_;

}

void Matrix::input() { //[3]定义输入函数

cout<<"input value of matrix:"<<endl;

for(int i=0; i<2; i++)

for(int j=0; j<3; j++)

cin>>mat[i][j];

}

void Matrix::display() { //定义输出函数

for (int i=0; i<2; i++) {

for(int j=0; j<3; j++) {

cout<<mat[i][j]<<" ";

}

cout<<endl;

}

}

int main() {

Matrix a,b,c;

a.input();

b.input();

cout<<endl<<"Matrix a:"<<endl;

a.display();

cout<<endl<<"Matrix b:"<<endl;

b.display();

c=a+b;//【4】\_\_\_\_\_\_; //用重载运算符"+"实现两个矩阵相加

cout<<endl<<"Matrix c = Matrix a + Matrix b :"<<endl;

c.display();

return 0;

}

Test30：声明一个字符串类，用来存放不定长的字符串。重载运算符“==”，用于两个字符串的等于关系的比较运算。

#include <iostream>

#include <cstring>

using namespace std;

class MyString {

public:

MyString() {p=NULL;}

MyString(char \*str) {p=str;} //数据成员赋值

friend bool operator==(MyString &string1,MyString &string2); //友元函数声明

void display() {cout<<p;} //输出P所指向的字符串

private:

char \*p;

};

bool operator==(MyString &string1,MyString &string2) {

if(strcmp(string1.p,string2.p)!=0) return true; //比较两个字符串

else return false;

}

void compare(MyString &string1,MyString &string2) {

if(operator==(string1,string2)==0) { //调用运算符重载函数比较两个字符串

string1.display();

cout<<"=";

string2.display();

} else {

string1.display();

cout<<"!=";

string2.display();

}

cout<<endl;

}

int main() {

MyString string1("Hello"),string2("Book"),string3("Hello");

compare(string1,string2);

compare(string1,string3);

return 0;

}

Test31：先建立一个Point（点）类，包含数据成员下，有（坐标点）。以它为基类，派生出一个Circle（圆类，增加数据成员r（半径），再以Circle类为直接基类，派生出一个Cylinder（圆柱体）类，在增加数据成员h（高）。编写程序，重载运算符“<<”和“>>”,使之能够用于输出以上类的对象。

#include <iostream>

using namespace std;

class Point {

protected:

int x,y;

public:

Point() {x=0; y=0;}

Point(int a, int b) {x=a; y=b;}

void setX(int a) {x=a;}

void setY(int b) {y=b;}

int getX() {return x;}

int getY() {return y;}

};

class Circle: public Point {

protected:

int r;

public:

Circle(int x,int y,int r):Point(x,y) {this->r=r;}

void setR(int a) {r=a;}

int getR() {return r;}

};

class Cylinder: public Circle {

protected:

int h;

public:

Cylinder():Circle(0,0,0),h(0) {}

Cylinder(int x,int y,int r,int h):Circle(x,y,r) {this->h = h;}

void setH(int a) {h=a;}

int getH() {return h;}

friend istream & operator>>(istream &, Cylinder &);

friend ostream & operator<<(ostream &, Cylinder &);

};

istream & operator>>(istream &input, Cylinder &cc) {

int \_x, \_y, \_r, \_h;

cout<<"Enter the Cylinder: "<<endl;

cin >>\_x >>\_y >>\_r >>\_h;

cc.setX(\_x);

cc.setY(\_y);

cc.setR(\_r);

cc.setH(\_h);

return input;

}

ostream & operator<<(ostream & output, Cylinder & cc) {

output<<cc.getX()<<' '<<cc.getY()<<' '<<cc.getR()<<' '<<cc.getH()<<endl;

return output;

}

int main() {

Cylinder cc;

cin>>cc;

cout<<cc;

return 0;

}

Test32：写一个程序，定义抽象类Shape，由他派生的三个类：Circle（圆形），Rectangle（矩形），Trapezoid（梯形），用一个函数print Area分别输出三者的面积，3个图形的数据在定义对象是给定的。

// P188

#include <iostream>

using namespace std;

class Shape { //形状类

public:

virtual double area() const=0; //纯虚函数area()的声明

};

class Circle: public Shape { //圆形类

private:

double r;

public:

Circle(double a):r(a) {} //构造函数的定义

virtual double area() const {

return 3.14 \* r \* r;

}

};

class Rectangle: public Shape {

private:

double h, w;

public:

Rectangle(double a, double b):h(a),w(b) {} //构造函数的成员初始化列表

virtual double area() const {

return h \* w;

}

};

class Trapezoid: public Shape { //梯形类

private:

double h, w;

public:

Trapezoid(double a, double b):h(a),w(b) {}

virtual double area() const {

return 0.5 \* h \* w;

}

};

void printArea(const Shape &c) {

cout <<c.area()<< endl; //输出对象c的面积

}

int main() {

Circle c(2);

printArea(c);

Rectangle r(2,4);

printArea(r);

Trapezoid t(3,5);

printArea(t);

return 0;

}

Test33：定义一个学生类Student做基类，再派生一个Graduate类，学生类有学号，姓名，和分数，研究生增加工资，他们有同名的函数display（），利用虚函数，变成分别输出学生和研究生的数据，具体数据自拟。

#include <iostream>

#include <string>

using namespace std;

class Student {

protected://【1】\_\_\_\_\_\_ //声明数据成员的访问属性

int id;

string name;

int score;

public:

Student(int,string,int);

virtual void display();

};

Student::Student(int i, string n, int cr) {

id = i;

name = n;

score = cr;

}

void Student::display() {

cout << id << ": " << name << endl;

cout << score << endl;

}

class Graduate:public Student { //[2]Graduate类公有继承Student类

protected:

int salary;

public:

Graduate(int i,string n,int cr,int sa)//【3】\_\_\_\_\_\_ //构造函数的定义

:Student(i, n, cr),salary(sa) {}

void display();

};

void Graduate::display() { //[4]Graduate类成员函数display()函数的定义

cout << id << ": " << name << endl;

cout << score << endl;

cout << salary << endl;

}

int main() {

Student stu(1, "John", 99);

stu.display();

Graduate gra(2, "JOHNLIU", 100, 5000);

gra.display();

return 0;

}

Test34商店销售某一商品，商店每天公布统一的折扣（discount）。同时允许销售人员在销售时活掌握售价(price)，在此基础上，对一次购买10件以上者，还可以享受9.8折扣优惠。现在已知当天3名销售员的销售情况为：

售货员工号（num） 售出商品数目（quantity） 商品单价（price）

101 5 23.5

102 12 24.56

103 100 21.5

请编写程序，计算当日此商品的总销售额sum，以及每件商品的平均售价。要求用静态数据成员和静态成员函数。（提示：将折扣discount、总销售额sum和商品总销售数目n声明为静态数据成员，再定义静态成员函数average（求平均售价）和display（输出结果）。

//P145

#include <iostream>

using namespace std;

class Product {

public:

Product(int n,int q,float p):num(n),quantity(q),price(p) {};

void total(); //计算总销售额和总销售数目

static float average(); //静态average()函数的声明，求平均售价

static void display(); //输出总销售额和平均售价

private:

int num; //售货员工号

int quantity; //售出商品数目

float price; //商品单价

static float discount; //统一的折扣

static float sum; //总销售额

static int n; //总销售数目

};

void Product::total() {

float rate=1.0;

if(quantity>10) rate=0.98\*rate;

sum=sum+quantity\*price\*rate\*(1-discount);

n=n+quantity;

}

void Product::display(){ //静态display()函数的定义

cout<<sum<<endl;

cout<<average()<<endl;

}

float Product::average() {

return(sum/n);

}

float Product::discount =0.05; //静态成员数据discount的初始化

float Product::sum=0;

int Product::n=0;

int main() {

Product Prod[3]=

{Product(101,5,23.5),Product(102,12,24.56),Product(103,100,21.5)};

for(int i=0; i<3; i++)

Prod[i].total();

Product::display(); //调用静态成员函数display()输出总销售额和平均售价

return 0;

}

Test35 请编写程序，处理一个复数与一个double数相加的运算，结果存放在一个double型的变量d1中，输出d1的值，再以复数的形式输出此值。定义Complex（复数）类，在成员函数中包含重载类型转换运算符：operator double（）{return real；}。

#include <iostream>

using namespace std;

class Complex {

public:

Complex() {

real=0;

imag=0;

}

Complex(double r){ //类型转换构造函数，将double类型转换为Complex类型

real=r;

imag=0;

}

Complex(double r,double i) {

real=r;

imag=i;

}

operator double(){ //类型转换成员函数，将Complex类型转换为double类型

return real;

}

void display();

private:

double real;

double imag;

};

void Complex::display() {

cout<<"("<<real<<", "<<imag<<")"<<endl;

}

int main() {

Complex c1(3,4),c2;

double d1;

d1=c1+2.5 ;// double型常量2.5与复数c1相加，运算结果存放在d1中

cout<<"d1="<<d1<<endl;

c2=d1; //将double型变量d1转换为复数类型

cout<<"c2=";

c2.display();

return 0;

}

Test36 有一个Time类，包含数据成员mintue(分)和sec（秒），模拟秒表，每次走疫苗，满60秒进一分钟，此时秒又从0开始算。要求输出分和秒的值，且对后置自增运算符的重载。

#include <iostream>

using namespace std;

class Time {

public:

Time() {

minute=0;

sec=0;

}

Time(int m,int s):minute(m),sec(s) {}

Time operator++() { //前置++的重载成员函数

if(++sec>=60) {

sec-=60;

++minute;

}

return \*this;//return 【1】\_\_\_\_\_\_;

}

Time operator++(int) { //后置++的重载成员函数Time【2】\_\_\_\_\_\_

Time temp(\*this);

sec++;

if(sec++>=60 ) {// if(【3】\_\_\_\_\_\_ )

sec-=60;

++minute;

}

return temp;//return 【4】\_\_\_\_\_\_;

}

void display() {

cout<<minute<<":"<<sec<<endl;

}

private:

int minute;

int sec;

};

int main() {

Time time1(34,0), time2(35,0);

for (int i=0; i<61; i++) {

++time1;

time1.display();

}

for (int j=0; j<61; j++) {

time2++;

time2.display();

}

return 0;

}

Test37定义一个国家基类Country，包含国名、首都、人口等属性，派生出省类Province，增加省会城市、人口数量属性。

#include<iostream>

#include<string>

using namespace std;

/\*【1】\*/class Country {

public:

/\*【2】\*/Country(string nam,string c,int cp=0) { //初始化构造函数的定义

name=nam;

capital=c;

country\_population=cp;

}

protected:

string name;

string capital;

long int country\_population;

};

class Province:public Country {

public:

//派生类构造函数的定义

Province(string nam,string c,long int cp,string pc,long int pp):/\*【3】\*/Country(nam,c,cp){

Province\_capital=pc;

Province\_population=pp;

};

void show() {

cout<<"name:"<<name<<endl;

cout<<"capital:"<<capital<<endl;

cout<<"country\_population:"<<country\_population<<endl;

cout<<"Province\_capital:"<<Province\_capital<<endl;

cout<<"Province\_population:"<<Province\_population<<endl;

}

private:

string Province\_capital;

long int Province\_population;

};

int main() {

//对象定义

/\*【4】\*Province prov1("China","Beijing",1300000000,"Nanchang",45000000);

prov1.show();

return 0;

}

Test38定义一个车基类Vehicle，含私有成员speed，weight。派生出自行车类Bicycle，增加high成员；汽车类Car，增加seatnum（座位数）成员。从bicycle和car中派生出摩托车类Motocycle。

#include<iostream>

using namespace std;

class Vehicle {

public:

Vehicle(float sp,float w) { //构造函数的定义

speed=sp;

weight=w;

}

void display() {

cout<<"speed:"<<speed<<" weight"<<weight<<endl;

}

private:

float speed;

float weight;

};

class Bicycle: virtual public Vehicle { //声明Vehicle为虚基类

public:

Bicycle(float sp,float w,float h):Vehicle(sp,w) { //构造函数的定义

high = h;

}

protected:

float high;

};

class Car:virtual public Vehicle {

public:

Car(float sp,float w,int num):Vehicle(sp,w) { //构造函数的定义

seatnum = num;

}

protected:

int seatnum;

};

class Motorcycle: public Bicycle, public Car {

public:

Motorcycle(float sp,float w,float h,int num):

Vehicle(sp,w),Bicycle(sp,w,h),Car(sp,w,num) {} //成员初始化

void display() {

Vehicle::display();

cout<<" high:"<<high<<" seatnum:"<<seatnum<<endl;

}

};

int main() {

Motorcycle m(120,120,120,1); //对象定义

m.display();

return 0;

}

Test39声明一个哺乳动物Mammal类，再由此派生出狗Dog类，二者都定义Speak（）成员函数，基类中定义为虚函数。声明一个Dog类的对象，调用Speak（）函数，观察运行结果

#include <iostream>

class Mammal {

public:

Mammal():itsAge(1) {

cout << "Mammal constructor"<<endl;

}

~Mammal() { //析构函数的定义

cout << "Mammal destructor"<<endl;

}

virtual void Speak() const { //虚函数Speak()的定义

cout << "Mammal speak!"<<endl;

}

private:

int itsAge;

};

class Dog : public Mammal {

public:

Dog() {

cout << "Dog Constructor" <<endl;

}

~Dog() {

cout << "Dog destructor" <<endl;

}

void Speak() const {

cout << "Woof!"<<endl;

}

};

int main() {

Mammal \*pDog = new Dog;

pDog->Speak(); //使用pDog调用Speak()函数

delete pDog; //释放对象

return 0;

}

Test40定义一个抽象类Cshape，包含纯虚数Area（用来计算面积）。然后派生出三角形Ctriangle类、矩形Crect类、圆Ccircle类，分别求其面积。最后定义一个CArea类，计算着几个形状的面积之和，各形状的数据通过CArea类构造函数或成员函数来设置。

#include<iostream>

using namespace std;

class Cshape {

public:

virtual float Area() const=0; //纯虚函数Area()的声明 【1】\_\_\_\_\_\_ float Area()【2】\_\_\_\_\_\_;

};

class CTriangle: public Cshape {

int vect;

public:

CTriangle(int v):vect(v) {}

float Area() {

return vect\*1.732/4;

}

};

class CRect: public Cshape {

int length,height;

public:

CRect(int l,int h):length(l),height(h) {}

float Area() {

return length\*height;

}

};

class CCircle: public Cshape {

int radius;

public:

CCircle(int r):radius(r) {}

float Area() {

return 3.14\*radius\*radius;

}

};

class Area {

CTriangle t;

CRect r;

CCircle c;

public:

Area(int v,int l,int h,int r):t(v),r(l,h),c(r) {} //子对象初始化列表 不会

float sum() {

return t.Area()+r.Area()+c.Area();

}

};

int main() {

Area a(10,20,30,5); //对象定义 4】\_\_\_\_\_\_ a(10,20,30,5);

cout<<a.sum()<<endl;

return 0;

}