提示：练习随课程进度，每章讲完后一周内完成。第十三章没有讲到的习题可以不做。作业提交到学校网络教学平台。按章随教学进度，每章讲完开放提交，一周后关闭本章作业提交。

第八章

1．下面是一个类的测试程序，试设计出能使用如下测试程序的类

int main()

{

Test x;

x.initx(30,20);

x.printx();

return 0;

}

输出结果：30\*20=600

8.1、

class Test

{

int a;

int b;

public:

void initx(int x, int y) {

a = x;

b = y;

}

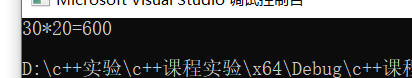
void printx()

{

cout << a << "\*" << b << "=" << a \* b << endl;

}

};



4．定义并实现一个矩形类Crectangle。该类包括以下成员函数：

Crectangle（）： 类的构造函数，根据需要可以定义多个构造函数

SetTop（）、SetLeft（）： 设置矩形的左上角坐标

SetLength（）、SetWidth（）：设置矩形的长和宽

Perimeter（）： 求矩形的周长

Area（）： 求矩形的面积

GetWidth（）： 返回矩形的宽度

GetLength（）： 返回矩形的长度

IsSquare（）： 判断矩形是否为一个正方形

Move（）： 将矩形从一个位置移到另一个位置

Size（）： 改变矩形的大小

Where（）： 返回矩形的左上角坐标

PrintRectangle（）： 输出矩形的四个顶点的坐标

（有关函数的参数及其返回类型，根据需要自行定义）

该类还包括以下数据成员：

int top，left； // 分别保存矩形的左上角顶点坐标

int length，width； // 分别保存矩形的长和宽

8.4：

#include<iostream>

using namespace std;

class Crectangle

{

public: Crectangle(int le = 0, int w = 0, int t = 0, int l = 0) :top(t), left(l),length(t),width(l) {}

void settop(int t) { top = t; }

void setleft(int l) { left = l; }

void setLength(int l) { length = l; }

void setwidth(int w) { width = w; }

int perimeter() { return 2 \* (length + width); }

int Area() { return length \* width; }

int Getwidth() { return width; }

int Getlength() { return length; }

bool IsSquare() { return length == width ? true : false; }

void move(int t, int l) { top += t, left += l; }

void Size(int l, int w) { Crectangle(l, w); }

void Where(int& x, int& y);

void PrinteRectangle();

private: int top, left, length, width;

};

void Crectangle::Where(int& x, int& y)

{

x = top;

y = left;

}

void Crectangle::PrinteRectangle()

{

cout << "四个位置点\n(" << top << "," << left << ")" << endl;

cout << "(" << top + length << "," << left << ")" << endl;

cout << "(" << top << "," << left - width << ")" << endl;

cout << "(" << top + length << "," << left - width << ")" << endl;

}

int main()

{

Crectangle a(0, 0, 10, 10);

cout << "长度为" << a.Getlength() << endl;

cout << "宽为" << a.Getwidth() << endl;

cout << "面积为" << a.Area() << endl;

if (a.IsSquare())cout << "是正方形";

else cout << "不是正方形";

a.PrinteRectangle();

cout << "移动，输入移动单位" << endl;

cout << "向右移动：" << endl;

int x, w;

cin >> x;

cout << "向上移动：" << endl;

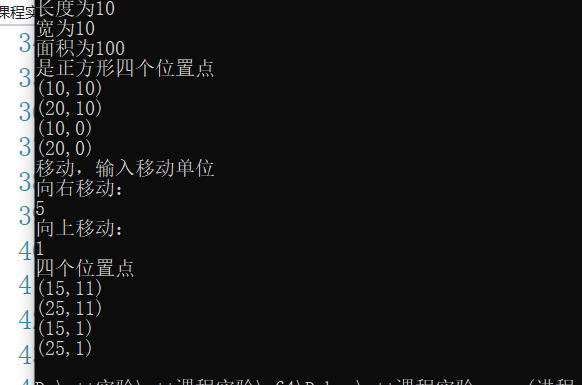
cin >> w;

a.move(x, w);

a.PrinteRectangle();

return 0;

}



6．在某次歌手大赛中，有JudgeNum个评委给选手打分，参加比赛的选手有PlayerNum名，现为比赛记分编写一个CompetitionResult类，类的定义如下：

class CompetitionResult {

short num; // 选手号码

char name[10]; // 选手姓名

float score[JudgeNum]; // 记录各评委给选手的打分

float average; // 选手最后得分，去掉一个最高分和一个最低分后的平均分

public:

CompetitionResult( ); // 无参数构造函数

CompetitionResult(short n, char \* ps); // 一般构造函数

float MaxScore( ); // 求评委打的最高分

float MinScore( ); // 求评委打的最低分

void SetAvg(); // 求选手的最后得分

float GetAvg() // 读选手的最后得分

short GetNo( ) // 读选手的编号

void setNo (int i) // 设置选手的编号

char \* GetName( ) // 读选手的姓名

float GetScore(int j) // 读第j个评委的打分

void SetScore(int k, float av) // 记录第j个评委的打分

friend void Sort(CompetitionResult \* pr, int n); // 按最后得分由高到低排序

};

（1）写出所有成员函数的实现代码；

（2）编写main( )函数对该类进行测试。在函数体中，定义一个CompetitionResult类的对象数组r[PlayerNum]，它的每个元素记录着每个运动员的所有信息，各评委的打分通过键盘输入，在屏幕上应有提示信息进行交互式操作，比赛结果按选手得分从高到低排序输出。

8.6：

#include<iostream>

#include<string>

#include<iomanip>

using namespace std;

const int JudgeNum = 4;

class CompetitionResult

{

short num; //选手号码

char name[10]; //选手姓名

float score[JudgeNum]; //记录各评委给选手的打分

float average; //选手最后得分，去掉一个最高分和最低分后的平均分

public:

CompetitionResult();

CompetitionResult(short n, char \*ps);

float MaxScore(); //求评委打的最高分

float MinScore(); //求评委打的最低分

void SetAvg(); //求选手的最后得分

float GetAvg(); //读选手的最后得分

short GetNo(); //求选手的编号

void setNo(int j); //设置选手的编号

char \* GetName(); //读选手的姓名

float GetScore(int j); //读第j个评委的打分

void SetScore(int j, float av); //记录第j个评委的打分

friend void Sort(CompetitionResult \*pr, int n); //按最后得分从高到低排序

};

CompetitionResult::CompetitionResult()

:num(0)

{

strcpy(name, "");

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

{

score[i] = 0.0;

}

average = 0.0;

}

CompetitionResult::CompetitionResult(short n, char \*ps)

:num(n)

{

strcpy(name, ps);

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

{

score[i] = 0.0;

}

average = 0.0;

}

float CompetitionResult::MaxScore()

{

float Max = score[0];

for (int i = 1; i < JudgeNum; ++i)

{

if (score[i] > Max)

{

Max = score[i];

}

}

return Max;

}

float CompetitionResult::MinScore()

{

float Min = score[0];

for (int i = 1; i < JudgeNum; ++i)

{

if (score[i] < Min)

{

Min = score[i];

}

}

return Min;

}

void CompetitionResult::SetAvg()

{

float sum = 0.0;

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

{

sum += score[i];

}

average = (sum - MaxScore() - MinScore()) / (JudgeNum - 2);

}

float CompetitionResult::GetAvg()

{

return average;

}

short CompetitionResult::GetNo()

{

return num;

}

void CompetitionResult::setNo(int j)

{

num = j;

}

char\*CompetitionResult::GetName()

{

return name;

}

float CompetitionResult::GetScore(int j)

{

return score[j];

}

void CompetitionResult::SetScore(int j, float av)

{

score[j] = av;

}

void Sort(CompetitionResult \*pr, int n)

{

CompetitionResult t;

for (int i = 0; i < n - 1; i++)

{

for (int j = 0; j < n - 1 - i; ++j)

{

if ((pr[j].average) < (pr[j + 1].average))

{

t = pr[j];

pr[j] = pr[j + 1];

pr[j + 1] = t;

}

}

}

}

int main()

{

const int num = 3;

CompetitionResult player[num] =

{

CompetitionResult(1,(char\*)"a"),

CompetitionResult(2,(char\*)"b"),

CompetitionResult(3,(char\*)"c"),

};

cout << fixed << setprecision(2);

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

{

cout << "请给第" << i + 1 << "个选手打分：" << endl;

float a;

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

{

cin >> a;

player[i].SetScore(j, a);

}

player[i].SetAvg();

}

Sort(player, num);

cout << "最终得分为：" << endl;

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

{

cout << i + 1<<" ";

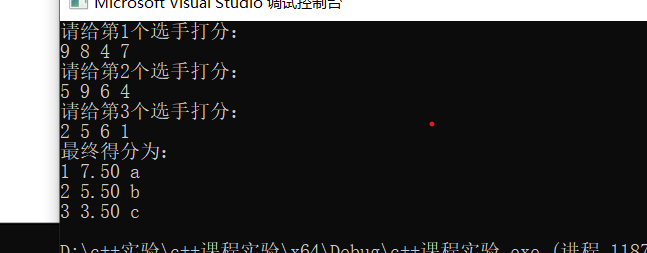
cout << player[i].GetAvg();

cout << " " << player[i].GetName() << endl;

}

return 0;

}



第九章

1．在下面的程序中，派生类Derived的成员函数sets()能否访问基类Base中的变量a和b？为什么？如果在基类Base中将数据成员a和b定义为私有成员，下面的程序能否通过编译？为什么？

#include <iostream>

using namespace std;

class Base {

public:

void set(int i,int j) { a=i; b=j; }

void show() { cout<<"a="<<a<<" b="<<b<<endl;}

protected:

int a,b;

};

class Derived:public Base {

public:

void sets() { s=a\*b;}

void shows() { cout<<"s="<<s<<endl; }

private:

int s;

};

int main() {

Derived obj;

obj.set(2,3);

obj.show();

obj.sets();

obj.shows();

return 0;

}

9.1：

（1）能，a，b类型为protected，派生类中可以通过派生类对象访问基类的protected成员。

（2）不可以，私有成员只能由定义类型或者任何嵌套类型中的方法使用。

2．声明一个图形基类Shape，在它的基础上派生出矩形类Rectangle和圆形类Circle，它们都有计算面积和周长、输出图形信息等成员函数，再在Rectangle类的基础上派生出正方形类Square。编写程序完成各类的定义和实现，以及类的使用。

9.2：

#include<iostream>

#include<iomanip>

using namespace std;

#define PI 3.1415926535

class Shape

{

public:

protected:

double mianji;

private:

};

class Circle :public Shape

{

public:

Circle();

void CalculateArea();

void display();

protected:

double R;

private:

};

Circle::Circle()

{

cout << "请输入圆的半径：" << endl;

cin >> R;

}

void Circle::CalculateArea()

{

mianji = PI \* R \* R;

}

void Circle::display()

{

cout << "这个圆的面积为：" << endl;

cout << mianji << endl;

}

class Rectangle :public Shape

{

public:

Rectangle();

void CalculateArea();

void display();

protected:

double x;

double y;

private:

};

Rectangle::Rectangle()

{

cout << "请输入矩形的长和宽：" << endl;

cin >> x >> y;

}

void Rectangle::CalculateArea()

{

mianji = x \* y;

}

void Rectangle::display()

{

cout << "这个矩形的面积为：" << mianji << endl;

}

void main()

{

Circle x;

cout << fixed << setprecision(2);

x.CalculateArea();

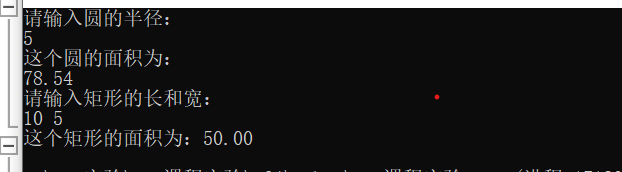
x.display();

Rectangle y;

y.CalculateArea();

y.display();

}



5．某销售公司有销售经理和销售员工，月工资的计算办法是：销售经理的底薪为4000元，并将销售额的2/1000作为提成，销售员工无底薪，只提取销售额的5/1000作为工资。编写程序：

（1）定义—个基类Employee，它包含3个数据成员number（职员编号）、name（姓名）和salary（工资），以及用于输入编号和姓名的构造函数。

（2）由Employee类派生Salesman类。Salesman类包含两个新数据成员commrate（提成比例）和sales（销售额），还包含用于输入销售额并计算销售员工工资的成员函数pay()和用于输出的成员函数print()。

（3）由Salesman类派生Salesmanager类。Salesmanager类包含新数据成员monthlypay（底薪），以及用于输入销售额并计算销售经理工资的成员函数pay()，用于输出的成员函数print（）。

（4）编写main函数中，测试所设计的类结构，并求若干个不同员工的工资。

9.5

#include<iostream>

using namespace std;

class Employee

{

char name[30];

int number;

double salary;

public:

Employee(int num, char \*p):number(num) {strcpy(name,p);}

void Setsalary(double sala) { salary = sala; }

double Getsalary() { return salary; }

char\*Getname() { return name; }

int GetNum() { return number; }

};

class Saleman :public Employee

{

double commrate, sales;

public:

Saleman(int num, char \*p, double commra) :Employee(num, p), commrate(commra) {}

void pay(double sa);

void print() { cout <<GetNum()<< "号销售员工" << Getname() << "工资为：" << Getsalary() << endl; }

};

void Saleman::pay(double sa)

{

sales = sa;

Setsalary(commrate\*sales);

}

class Salesmanager :public Saleman

{

double monthlypay;

public:

Salesmanager(int num, char \*p, double commrate, double mon) : Saleman(num, p, commrate), monthlypay(mon) {}

void pay(double sa) ;

void print() { cout << GetNum() << "销售经理" << Getname() << "工资为：" << Getsalary() << endl; }

};

void Salesmanager::pay(double sa)

{

Saleman::pay(sa);

double a = Getsalary();

Setsalary(a + monthlypay);

}

int main()

{

int num;

char name[20];

double sales;

cout << "输入销售员工的编号:";

cin >> num;

cout << "输入销售员工的姓名:";

cin >> name;

cout << "输入销售员工的销售额:";

cin >> sales;

Saleman s1(num, name, 0.005);

s1.pay(sales);

s1.print();

cout << endl;

cout << "输入销售经理的编号:";

cin >> num;

cout << "输入销售经理的姓名:";

cin >> name;

cout << "输入销售经理的销售额:";

cin >> sales;

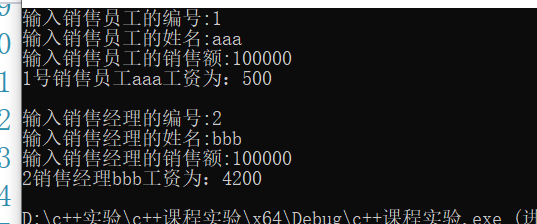
Salesmanager s2(num, name, 0.002,4000);

s2.pay(sales);

s2.print();

return 0;

}



第十章

定义一个复数类Complex，重载运算符+、-、\*、/，使之能用于复数的加、减、乘、除运算。运算符重载函数作为Complex类的成员函数。编写程序，分别求两个复数之和、差、积和商。

#include <iostream>

#include <iomanip>

using namespace std;

class Complex

{

public:

Complex() :real(0), imag(0) {};

Complex(double r, double i) :real(r), imag(i) {};

Complex operator+(Complex &c2);

Complex operator-(Complex &c2);

Complex operator\*(Complex &c2);

Complex operator/(Complex &c2);

void display();

private:

double real;

double imag;

};

Complex Complex::operator+(Complex &c2)

{

return Complex(real + c2.real, imag + c2.imag);

}

Complex Complex::operator-(Complex &c2)

{

return Complex(real - c2.real, imag - c2.imag);

}

Complex Complex::operator\*(Complex &c2)

{

return Complex(real \* c2.real - imag \* c2.imag, imag \* c2.real + real \* c2.imag);

}

Complex Complex::operator/(Complex &c2)

{

return Complex((real \* c2.real + imag \* c2.imag) / (c2.real\*c2.real + c2.imag\*c2.imag),

(imag \*c2.real - real \* c2.imag) / (c2.real\*c2.real + c2.imag\*c2.imag));

}

void Complex::display()

{

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

}

int main()

{

Complex c1(1, 3), c2(0.5, 2);

cout << fixed << setprecision(2);

cout << "c1:";

c1.display();

cout << "c2:";

c2.display();

cout << "c1+c2=" ;

(c1 + c2).display();

cout << "c1-c2=" ;

(c1 - c2).display();

cout << "c1\*c2=" ;

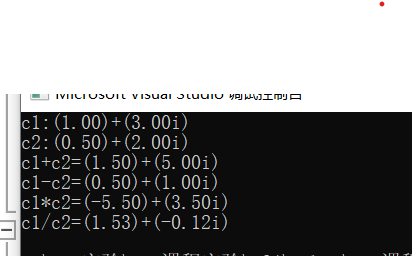
(c1 \* c2).display();

cout << "c1/c2=" ;

(c1 / c2).display();

return 0;

}



2．定义一个日期类Date，包括年、月、日等私有数据成员。要求为所定义的Date类中设计如下重载运算符函数：

Date operator+(int days); 返回一日期加天数days后得到的日期

Date operator-(int days); 返回一日期减去天数days后得到的日期

int operator-(Date &b); 返回两日期相差的天数

10.2、

#include<iostream>

using namespace std;

class Date

{

private:

int year;

int month;

int day;

public:

Date(int, int, int);

Date operator+(int days);

Date operator-(int days);

int operator-(Date& b);//运算符重载

int getYear();

int getMonth();

int getDay();//访问器

};

Date::Date(int y = 0, int m = 0, int d = 0) :year(y), month(m), day(d) {

}//初始化列表

Date Date::operator+(int days)

{

if (day + days > 30)

{

month++;

if (month > 12)

{

month = 1;

year++;

}

day = day + days - 30;

}

else

{

day += days;

}

return \*this;

}//运算符重载“+”，几天后的天数

Date Date::operator-(int days)

{

if (day - days < 0)

{

month--;

if (month == 0)

{

month = 12;

year--;

}

day = day - days + 30;

}

else

{

day -= days;

}

return \*this;

}//运算符重载“-”，几天前的天数

int Date::operator-(Date& b)

{

int days1 = year \* 360 + month \* 30 + day;

int days2 = b.year \* 360 + b.month \* 30 + b.day;

if (days1 > days2)

return days1 - days2;

else

return days2 - days1;

}//运算符重载“-”，两天差距天数

int Date::getYear()

{

return year;

}

int Date::getMonth()

{

return month;

}

int Date::getDay()

{

return day;

}

int main()

{

Date date1(2021, 6, 1);

cout << "Data1是: " << "(" << date1.getYear() << "," << date1.getMonth() << "," << date1.getDay() << ")" << endl;

Date date2(2021, 6, 27);

cout << "Data2是: " << "(" << date2.getYear() << "," << date2.getMonth() << "," << date2.getDay() << ")" << endl;

int tmpdays3;

tmpdays3 = date1 - date2;

cout << "Data1和Data2相隔: " << tmpdays3 << " 天" << endl;

Date date3;

date3 = date1 + 2;

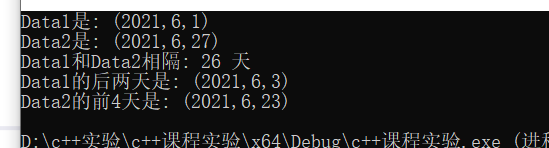
cout << "Data1的后两天是: " << "(" << date3.getYear() << "," << date3.getMonth() << "," << date3.getDay() << ")" << endl;

date3 = date2 - 4;

cout << "Data2的前4天是: " << "(" << date3.getYear() << "," << date3.getMonth() << "," << date3.getDay() << ")" << endl;

return 0;

}



3．编程计算正方体、圆柱体和球的表面积和体积。要求抽象出一个公共的基类Body，把它作为抽象类，在该类中定义求表面积和体积的纯虚函数。抽象类中定义一个数据成员data，它可以作为球的半径、正方体的边长或圆柱体低面圆的半径。由这个抽象类派生出描述球、正方体和圆柱的3个具体类，在这3个类中都有计算表面积和体积的函数的具体实现。

10.3、

#include<iostream>

#include<iomanip>

#include<cmath>

#define PI 3.14159

using namespace std;

class Body

{

protected:

double data;

public:

Body(double b) :data(b) {}

virtual double S() = 0;

virtual double V() = 0;

};

class Cube:public Body

{

public:

Cube(double r) :Body(r) {}

double S() { return data \* data \* 6; }

double V() { return data \* data \* data; }

void print();

};

void Cube::print()

{

cout << "正方体表面积为:" << S() << endl;

cout << "正方体体积为:" << V() << endl;

}

class Circle :public Body

{

public:

Circle(double r) :Body(r) {}

double S() { return data \* data \* 4\*PI; }

double V() { return data \* data\*data\*PI\*4/3; }

void print();

};

void Circle::print()

{

cout << "球体表面积为:" << S() << endl;

cout << "球体体积为:" << V() << endl;

}

class Cylinder :public Body

{

double height;

public:

Cylinder(double r, double h) :Body(r), height(h) {}

double S() { return data \* data \* PI\*2 + PI \* data \* 2 \* height; }

double V() { return data \* data\* PI\* height ; }

void print();

};

void Cylinder::print()

{

cout << "球体表面积为:" << S() << endl;

cout << "球体体积为:" << V() << endl;

}

int main()

{

double r,h;

cout << fixed << setprecision(2);

cout << "输入正方体的边长:";

cin >> r;

Cube c1(r);

c1.print();

cout << endl;

cout << "输入圆的半径:";

cin >> r;

Circle c2(r);

c2.print();

cout << endl;

cout << "输入圆柱体的半径:";

cin >> r;

cout << "输入圆柱体的高:";

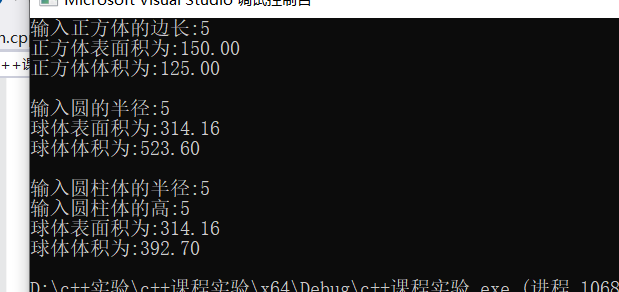
cin >> h;

Cylinder c3(r,h);

c3.print();

return 0;

}



第十一章

1．已知

void Sort(int a[],int size);

void Sort(double a[],int size);

是一个函数模板的两个实例，其功能是将数组a中的前size个元素按从小到大顺序排列。试设计这个函数模板。

11.1

#include<iostream>

using namespace std;

template <class T1,class T2>

void Sort(T1 &a, T2 size)

{

int temp;

for (int i = 0; i < size - 1; i++)

{

for (int j = 0; j < size - 1 - i; j++)

{

if (a[j] > a[j + 1])

{

temp = a[j];

a[j] = a[j + 1];

a[j + 1] = temp;

}

}

}

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

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

}

int main()

{

int a[5] = { 5,6,3,8,4 };

double b[5] = { 1.2,3,5.6,7,9 };

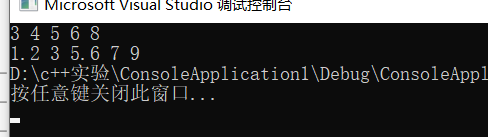
Sort(a, 5);

cout << endl;

Sort(b, 5);

return 0;

}



2．设有如下的类声明：

class Test

{

public:

void SetData1(int val) { data1=val; }

void SetData2(double val) {data2=val: }

int GetData1() { return data1; }

double GetData2() { return data2; }

private:

int data1;

double data2;

}

试将此类声明改为类模板声明，使得数据成员data1和data2可以是任何类型。

11.2

#include<iostream>

using namespace std;

template<class T1,class T2>

class Test

{

public:

void SetData1(T1 val) { data1=val; }

void SetData2(T2 val) {data2=val;}

T1 GetData1() { return data1; }

T2 GetData2() { return data2; }

private:

T1 data1;

T2 data2;

};

int main()

{

Test<int,double>t1;

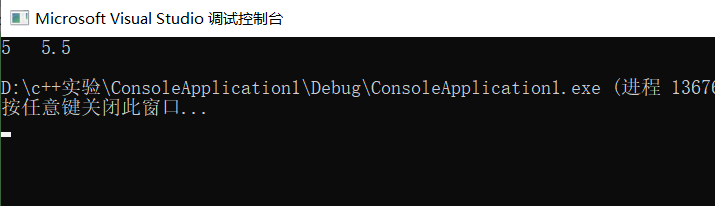
t1.SetData1(5);

t1.SetData2(5.5);

cout<<t1.GetData1()<<" "<<t1.GetData2()<<endl;

return 0;

}



3．栈是一种重要的数据结构，它是一种只允许在表的一端进行插入或删除操作的线性表。表中允许进行插入、删除操作的一端称为栈顶。表的另一端称为栈底。栈顶的当前位置是动态的，对栈顶当前位置的标记称为栈顶指针。当栈中没有数据元素时，称之为空栈。栈的插入操作通常称为进栈或入栈，栈的删除操作通常称为退栈或出栈。下面是一个整型栈类的定义：

const int SIZE= 100； // 栈中能保存的最多元素个数

class IStack

{

public:

Istack(); // 栈的构造函数

void Push(int n); // 往栈顶增加元素

int Pop(); // 从非空栈的栈顶删除一个元素

int GetTop(); // 返回非空栈的栈顶元素

bool Empty(); // 判断栈是否为空

int Size(); // 返回栈中元素的个数

void ClearStack(); // 将栈清空

~Istack(); // 栈的析构函数

private:

int stack[SIZE]； // 保存栈中各元素的数组

int top; // 保存栈顶的当前位置

}

试编写一个栈的类模板（包括其成员函数的实现），以便为任何类型的对象提供栈结构的数据操作。

11.3

#include<iostream>

using namespace std;

const int SIZE = 100; // 栈中能保存的最多元素个数

template <class T>

class IStack

{

public:

IStack() :stack{ 0 }, top(0){} // 栈的构造函数

void Push(T n); // 往栈顶增加元素

T Pop(); // 从非空栈的栈顶删除一个元素

T GetTop(); // 返回非空栈的栈顶元素

bool Empty(); // 判断栈是否为空

int Size(); // 返回栈中元素的个数

void ClearStack(); // 将栈清空

~IStack() {}; // 栈的析构函数

private:

T stack[SIZE]; // 保存栈中各元素的数组

int top; // 保存栈顶的当前位置

};

template <class T>

void IStack<T>::Push(T n)

{

if (top < SIZE)

stack[top++] = n;

else

{

cout << "错误" << endl;

return ;

}

}

template <class T>

T IStack<T>::Pop()

{

if (top >= 0)

{

top -= 1;

stack[top]=0;

}

else

{

cout << "错误" << endl;

return 0;

}

}

template <class T>

T IStack<T>::GetTop()

{

if (top >= 0)

return stack[top-1];

else

{

cout << "错误" << endl;

return 0;

}

}

template <class T>

int IStack<T>::Size()

{

return top;

}

template <class T>

bool IStack<T>::Empty()

{

if (top == 0)

return true;

else

return false;

}

template <class T>

void IStack<T>::ClearStack()

{

while (top > 0)

Pop();

}

int main()

{

IStack<int> s;

for (int i=0; i < 5; i++)

{

s.Push(i);

}

cout << "栈顶元素为" << s.GetTop() << endl;

cout << "栈长度为" << s.Size() << endl;

cout << "栈元素减一" << endl;

s.Pop();

cout << "栈顶元素为" << s.GetTop() << endl;

cout << "栈长度为" << s.Size() << endl<<endl;

s.ClearStack();

IStack<char> t;

for (int i=0; i < 5; i++)

{

t.Push('a'+i);

}

cout << "栈顶元素为" << (char)t.GetTop() << endl;

cout << "栈长度为" << t.Size() << endl;

cout << "栈元素减一" << endl;

t.Pop();

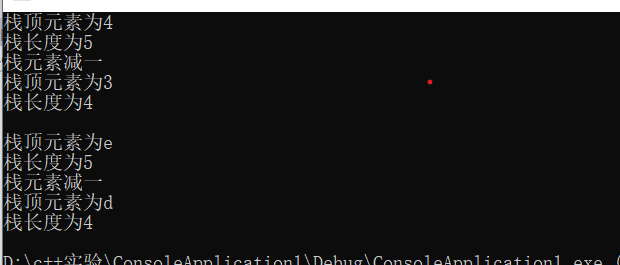
cout << "栈顶元素为" <<(char)t.GetTop() << endl;

cout << "栈长度为" << t.Size() << endl;

t.ClearStack();

return 0;

}



第十三章

1．编写程序，重载运算符“<<”和“>>”，使用户能直接输入和输出复数。

13.1

#include<iostream>

using namespace std;

class Complex

{

double real;

double imag;

public:

Complex() { real = 0; imag = 0; }

friend ostream &operator<<(ostream &out, Complex &c);

friend istream &operator>>(istream &in, Complex &c);

};

ostream &operator<<(ostream &out, Complex &c)

{

out << "输出复数" << endl;

out << c.real << "+" << c.imag << "i" << endl;

return out;

}

istream &operator>>(istream &in, Complex &c)

{

cout << "输入复数的实部和虚部:";

in >> c.real;

in >> c.imag;

return in;

}

int main()

{

Complex c1, c2;

cin >> c1;

cout << c1;

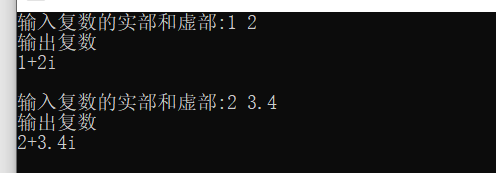
cout << endl;

cin >> c2;

cout << c2;

return 0;

}



后面两题涉及文件部分，属于未学内容。