**实验一：**

**1**、

#include"MyClass.h"

int main()

{

MyClass obj1(1,3), obj2(5, 8);

obj1.Print();

obj2.Print();

return 0;

}

#include"MyClass.h"

using namespace std;

MyClass::MyClass(int a, int b)

{

x = a;

y = b;

}

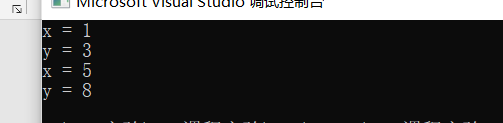
void MyClass::Print()

{

cout << "x = " << x << endl;

cout << "y = " << y << endl;

}



**2、**

#include<iostream>

#include<iomanip>

using namespace std;

class Rectangle

{

private:

double weight, length;

public:

double Circumference();

double Area();

Rectangle(double a = 0, double b = 0);

~Rectangle() {};

};

Rectangle::Rectangle(double a, double b)

{

length = a;

weight = b;

}

double Rectangle::Area()

{

return weight \* length;

}

double Rectangle::Circumference()

{

return (weight + length) \* 2;

}

int main()

{

Rectangle rect1(20, 50), rect2(3.6, 4.5); cout << fixed << setprecision(2);

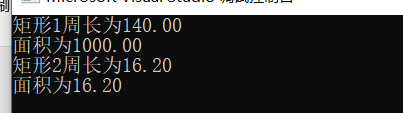
cout << "矩形1周长为" << rect1.Circumference() << endl;

cout << "面积为" << rect1.Area() << endl;

cout << "矩形2周长为" << rect2.Circumference() << endl;

cout << "面积为" << rect2.Area() << endl;

}



**3、**

#include<iostream>

using namespace std;

class IntArray {

public:

IntArray(int sz); // 数组初始化为sz个元素，初值全为0

IntArray(int \*array, int sz); // 用静态数组array的前sz个元素初始化新数组

IntArray(const IntArray &rhs); // 拷贝构造函数

void copy(int \*array, int sz);

void printAll();

~IntArray() { delete[]ia; }

private:

int \*ia;

int size;

};

IntArray::IntArray(int sz)

{

copy(NULL, sz);

}

IntArray::IntArray(int \*array, int sz)

{

copy(array, sz);

}

IntArray::IntArray(const IntArray &rhs)

{

copy(rhs.ia, rhs.size);

}

void IntArray::copy(int \*array, int sz)

{

size = sz; // 设置数据成员

ia = new int[size];

if (array != NULL)

{

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

ia[ix] = array[ix]; // 拷贝数据成员

}

else

{

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

ia[ix] = 0; // 初始化数组元素

}

}

void IntArray::printAll()

{

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

cout << ia[ix] << " ";

cout << endl;

}

int main()

{

int a[10] = { 1,2,3,4,5,6,7,8,9,10 };

IntArray arr1(10), arr2(a, 5), arr3(arr2);

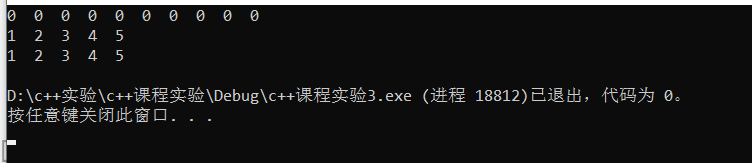
arr1.printAll();

arr2.printAll();

arr3.printAll();

return 0;

}



**4、**

#include<iostream>

#include <iomanip>

using namespace std;

class myPoint {

public:

myPoint(double x0 = 0.0, double y0 = 0.0) :x(x0), y(y0) {}

myPoint(myPoint& np) :x(np.x), y(np.y) {}

double GetX() { return x; }

double GetY() { return y; }

void SetX(double x0) { x = x0; }

void SetY(double y0) { y = y0; }

void SetPoint(double x0, double y0) { x = x0; y = y0; }

void SetPoint(myPoint& np) { x = np.x; y = np.y; }

double GetLength(myPoint p) {

return sqrt((x - p.x) \* (x - p.x) + (y - p.y) \* (y - p.y));

}

void Printit() { cout << " (" << x << "," << y << ") "; }

private:

double x, y;

};

class Triangle

{

myPoint p1, p2, p3;

public:

Triangle(){} //默认构造函数

Triangle(myPoint point1, myPoint point2, myPoint point3):p1(point1),p2(point2),p3(point3){}

double Perimeter();

double Area(double s);

};

double Triangle::Perimeter()

{

return p1.GetLength(p2) + p2.GetLength(p3) + p3.GetLength(p1);

}

double Triangle::Area(double s)

{

double a = p1.GetLength(p2);

double b = p2.GetLength(p3);

double c = p3.GetLength(p1);

return sqrt(s \* (s - a) \* (s - b) \* (s - c));

}

int main()

{

double x, y;

cout << "输入第一个点的坐标:";

cin >> x >> y;

myPoint m1(x, y);

cout << "输入第二个点的坐标:";

cin >> x >> y;

myPoint m2(x, y);

cout << "输入第二个点的坐标:";

cin >> x >> y;

myPoint m3(x, y);

Triangle t(m1, m2, m3);

x = t.Perimeter();

y = t.Area(x/2);

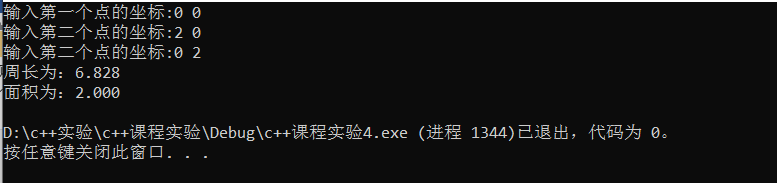
cout << fixed;

cout << setprecision(3)<< "周长为：" << x << endl;

cout << setprecision(3) << "面积为：" << y << endl;

return 0;

}



**实验二：**

**1、**

#include <iostream>

using namespace std;

class Base

{

public:

Base(int p1, int p2) { data1 = p1; data2 = p2; }

int Inc1() { return ++data1; }

int Inc2() { return ++data2; }

void Display()

{

cout << "data1 = " << data1 << " data2 = " << data2 << endl;

}

protected:

int data1, data2;

};

class D1 : virtual public Base {

public:

D1(int p1, int p2, int p3) : Base(p1, p2) { data3 = p3; }

int Inc1() { return Base::Inc1(); }

int Inc3() { return ++data3; }

void Display()

{

cout << "data1 = " << data1 << " data2 = " << data2 << " data3 = " << data3 << endl;

cout << "Base::Display() ----";

Base::Display();

}

protected:

int data3;

};

class D2 :virtual public Base {

public:

D2(int p1, int p2, int p4) : Base(p1, p2) { data4 = p4; }

int Inc1()

{

Base::Inc1(); Base::Inc2();

return Base::Inc1();

}

int Inc4() { return ++data4; }

void Display()

{

cout << "data1 = " << data1 << " data2 = " << data2 << " data4 = " << data4 << endl;

cout << "Base::Display() ----";

Base::Display();

}

protected:

int data4;

};

class D12 : public D1, public D2 {

public:

D12(int p11, int p12, int p13, int p21, int p22, int p23, int p)

: D1(p11, p12, p13), D2(p21, p22, p23),Base(p11,p12) {

data5 = p;

}

int Inc1()

{

D1::Inc1(); D2::Inc1();

return D1::Inc1();

}

int Inc5() { return ++data5; }

void Display()

{

cout << "data1 = " << data1 << " data2 = " << data2 << endl; // ①

cout << "data3 = " << data3 << " data4 = " << data4 << " data5 = " << data5 << endl;

cout << "D1::Display( )----";

D1::Display();

cout << "D2::Display( )----";

D2::Display();

}

private:

int data5;

};

int main()

{

D12 d(1, 2, 3, 4, 5, 6, 7);

d.Display();

cout << endl;

d.Inc1();

d.Inc2(); // ②

d.Inc3();

d.Inc4();

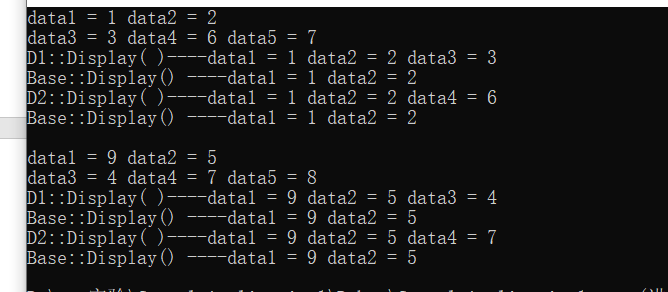
d.Inc5();

d.D12::Inc1();

d.Display();

return 0;

}



**2、**

#include<iostream>

#include <cmath>

#include<iomanip>

#define PI 3.14159

using namespace std;

class Point{

public:

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

Point(double xv,double yv) { x = xv; y = yv; }

Point(Point& pt) { x = pt.x; y = pt.y; }

double getx() { return x; }

double gety() { return y; }

double Area() { return 0; }

void Show() { cout << "x=" << x << ' ' << "y=" << y << endl; }

private:

double x,y;

};

class Rectangle : public Point {

public:

Rectangle() :Point(0, 0) { length = 0; width = 0; }

Rectangle(Point &p1, double a, double b) : Point(p1), length(a), width(b) {}

void position(Point& pt);

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

double Perimeter() { return 2 \* (length+width); }

private:

double length, width;

};

void Rectangle::position(Point& pt)

{

if (pt.getx() > getx() && pt.getx() < getx() + length && pt.gety() > gety() && pt.gety() < gety() + width)

cout << "点在矩形内" << endl;

else if((pt.getx() == getx()|| pt.getx() == getx() + length) && pt.gety() >= gety() && pt.gety() <= gety() + width)

cout << "点在矩形上" << endl;

else if ((pt.gety() == gety()|| pt.gety() == gety() + width) && pt.getx() >= getx() && pt.getx() <= getx() + length)

cout << "点在矩形上" << endl;

else

cout << "点在矩形外" << endl;

}

class Circle :public Point {

public:

Circle(Point &p1, double ioradius) :Point(p1), radius(ioradius) {}

void position(Point& pt);

double Area() { return PI\* radius\*2; }

double Perimeter() { return PI\* radius\*radius; }

private:

double radius;

};

void Circle::position(Point& pt)

{

double dis = sqrt((pt.getx() - getx())\*(pt.getx() - getx()) + (pt.gety() - gety())\*(pt.gety() - gety()));

if (dis < radius)

cout << "点在圆内" << endl;

else if(dis== radius)

cout << "点在圆上" << endl;

else

cout << "点在圆外" << endl;

}

int main()

{

double x, y;

cout << fixed << setprecision(2);

cout << "输入点的坐标:";

cin >> x >> y;

Point p1(x, y);

cout << "输入矩形左上角坐标:";

cin >> x >> y;

Point p2(x, y);

cout << "输入矩形的长和宽:";

cin >> x >> y;

Rectangle r1(p2, x, y);

r1.position(p1);

cout << "面积为:" << r1.Area() << endl;

cout << "周长为:" << r1.Perimeter() << endl<<endl;

cout << "输入圆的圆心坐标:";

cin >> x >> y;

Point p3(x, y);

cout << "输入圆的半径:";

cin >> x ;

Circle c1(p3, x);

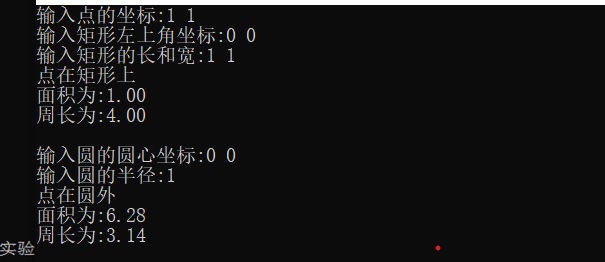
c1.position(p1);

cout << "面积为:" << c1.Area() << endl;

cout << "周长为:" << c1.Perimeter() << endl << endl;

return 0;

}



**实验三：**

**1**、

#include<iostream>

#include<cmath>

#include<iomanip>

#define PI 3.14159

using namespace std;

class CClosedFigure //图形类

{

public:

virtual void show() = 0; //图形画法

virtual double perimeter() = 0; //图形周长

virtual double area() = 0; //图形面积

virtual double volume() = 0; //图形体积

};

class Rectangle :public CClosedFigure //矩形类

{

protected:

double width, length;

public:

Rectangle(double w, double l) :width(w), length(l) {}

void show()

{

cout << "这是一个矩形： " << endl;

}

double perimeter()

{

double p;

p = 2 \* (width + length);

return p;

}

double area()

{

double a;

a = width \* length;

return a;

}

double volume()

{

return 0;

}

};

class Circle :public CClosedFigure //圆类

{

protected:

double radius;

public:

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

void show()

{

cout << "这是一个圆形： " << endl;

}

double perimeter()

{

double p;

p = 2 \* PI\*radius;

return p;

}

double area()

{

double a;

a = PI \*radius\*radius;

return a;

}

double volume()

{

return 0;

}

};

class Triangle :public CClosedFigure //三角形类

{

protected:

double side1, side2, side3;

public:

Triangle(double s1, double s2, double s3) :side1(s1), side2(s2), side3(s3) {}

void show()

{

cout << "这是一个三角形： " << endl;

}

double perimeter()

{

double p;

p = side1 + side2 + side3;

return p;

}

double area()

{

double a, p, pi;

p = perimeter() / 2;

pi = p \* (p - side1)\*(p - side2)\*(p - side3);

a = sqrt(pi);

return a;

}

double volume()

{

return 0;

}

};

class Box :public Rectangle //长方体类

{

protected:

double height;

public:

Box(double w, double l, double h) :Rectangle(w, l), height(h) {

}

void show()

{

cout << "这是一个长方形： " << endl;

}

double volume()

{

double v;

v = area()\*height;

return v;

}

};

class Cylinder :public Circle //圆柱类

{

protected:

double height;

public:

Cylinder(double r, double h) :Circle(r), height(h) {

}

void show()

{

cout << "这是一个圆柱： " << endl;

}

double volume()

{

double v;

v = area()\*height;

return v;

}

};

class Cone :public Circle //圆锥类

{

protected:

double height;

public:

Cone(double r, double h) :Circle(r), height(h) {

}

void show()

{

cout << "这是一个圆锥： " << endl;

}

double volume()

{

double v;

v = height \* area()/3;

return v;

}

};

class T\_pyramid :public Triangle //三棱锥类

{

protected:

double height;

public:

T\_pyramid(double s1, double s2, double s3, double h) :Triangle(s1, s2, s3), height(h) {

}

void show()

{

cout << "这是一个三棱锥： " << endl;

}

double volume()

{

double v;

v = area()\*height/3;

return v;

}

};

class T\_prism :public Triangle //三棱柱类

{

protected:

double height;

public:

T\_prism(double s1, double s2, double s3, double h) :Triangle(s1, s2, s3), height(h) {

}

void show()

{

cout << "这是一个三棱柱： " << endl;

}

double volume()

{

double v;

v = area()\*height/3;

return v;

}

};

int main()

{

CClosedFigure \*gs[] = { new Circle(10),new Rectangle(6,8),new Triangle(3,4,5),new Box(6,8,3),new Cylinder(10,3),

new Cone(10,3),new T\_pyramid(3,4,5,3),new T\_prism(3,4,5,3) };

cout << fixed << setprecision(2);

cout << "平面图形：" << endl;

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

{

gs[i]->show();

cout << "图形周长：" << gs[i]->perimeter() << '\t';

cout << "图形面积：" << gs[i]->area() << '\t';

cout << "图形体积：" << gs[i]->volume() << endl;

}

cout << "立体图形：" << endl;

for (int i = 3; i < 8; i++)

{

gs[i]->show();

cout << "图形底周长：" << gs[i]->perimeter() << '\t';

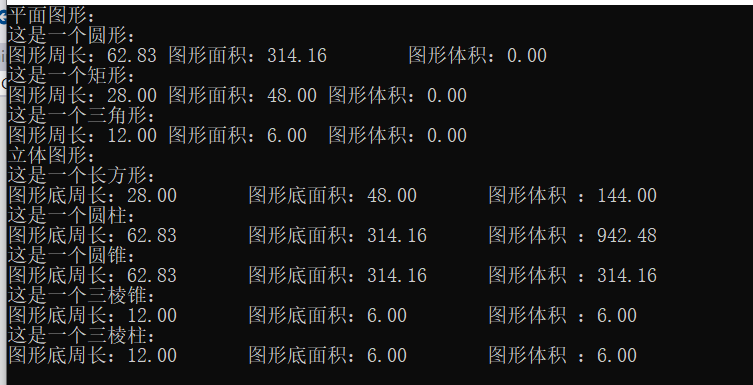
cout << "图形底面积：" << gs[i]->area() << '\t';

cout << "图形体积 ：" << gs[i]->volume() << endl;

}

return 0;

}



**2、**

#include<iostream>

using namespace std;

class Complex

{

double real;

double imag;

public:

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

//Complex operator + (Complex a1);

//Complex operator - (Complex a2);

friend Complex operator + (Complex a1, Complex a2);

friend Complex operator - (Complex a1,Complex a2);

void display();

};

/\*

Complex Complex::operator + (Complex a1)

{

return Complex(this->real + a1.real, this->imag + a1.imag);

}

Complex Complex::operator - (Complex a2)

{

return Complex(this->real - a2.real, this->imag - a2.imag);

}

\*/

Complex operator + (Complex a1, Complex a2)

{

return Complex(a1.real + a2.real, a1.imag + a2.imag);

}

Complex operator - (Complex a1, Complex a2)

{

return Complex(a1.real - a2.real, a1.imag - a2.imag);

}

void Complex::display()

{

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

}

int main()

{

Complex c1(6, 7), c2(7, 8);

cout << "c1=";

c1.display();

cout << "c2=";

c2.display();

cout << "c1+c2=";

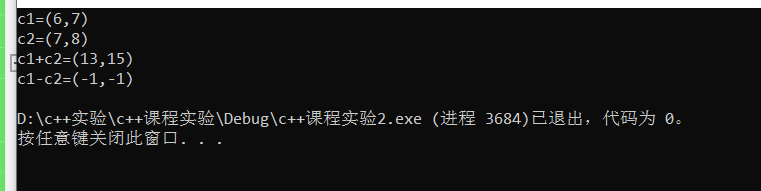
(c1 + c2).display();

cout << "c1-c2=";

(c1 - c2).display();

return 0;

}



**实验四：**

**1、**

#include<iostream>

using namespace std;

const int MaxSize = 100; // 栈中能保存的最多元素个数

template <class T>

class IStack

{

public:

IStack() :elem{ 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 elem[MaxSize]; // 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 < MaxSize)

elem[Top++] = n;

else

{

cout << "错误" << endl;

return ;

}

}

template <class T>

T IStack<T>::Pop()

{

if (Top >= 0)

{

Top -= 1;

elem[Top]=0;

return elem[Top - 1];

}

else

{

cout << "错误" << endl;

return elem[0];

}

}

template <class T>

T IStack<T>::GetTop()

{

if (Top >= 0)

return elem[Top-1];

else

{

cout << "错误" << endl;

return elem[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();

if (s.Empty())

cout << "栈为空" << endl;

else

cout << "栈不为空" << endl;

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

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

s.ClearStack();

IStack<char> t;

char ch;

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

{

ch = i+'a';

t.Push(ch);

}

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();

if (t.Empty())

cout << "栈为空" << endl;

else

cout << "栈不为空" << endl;

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

cout << "栈长度为" << t.Size() << endl;

t.ClearStack();

return 0;

}



**2、**

#include<iostream>

#include<string.h>

using namespace std;

class Phone

{

char number[20];

public:

Phone() { strcpy(number, ""); }

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

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

};

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

{

out << "输出电话号码" << endl;

out << c.number<<endl;

return out;

}

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

{

cout << "输入电话号码:"<<endl;

in >> c.number;

return in;

}

int main()

{

Phone c1, c2;

cin >> c1;

cout << c1;

cout << endl;

cin >> c2;

cout << c2;

return 0;

}

