Ch2

C++中的输入与输出：cin>>（存放输入值），cout<<(屏幕输出）的使用，#include <iostrem>与using namespace std，的作用是建立一些互相分隔的作用域，把一些全局实体分隔开来。

Ch3

内联函数：在编译时将没一个函数体嵌入在每一个调用处

函数的重载：两个以上的函数，具有相同的的函数名，但是形参的个数或者类型不同，编译器根据实参和形参的类型及个数的最佳匹配，自动确定调用哪一个函数

带有默认值的函数：函数在定义时可以预先声明默认的形参值，在相同作用域内，不允许在同一个函数的多个声明中对同一个参数的默认值重复定义，即使前后定义值相同也不行

Ch4

类与对象的概念：类：对逻辑上相关的函数与数据的封装，是对问题的抽象描述，描述了一类事物的共同属性和行为；对象：类的某一特定实体

面向对象程序设计的四个基本特性：抽象、封装、继承、多态

类成员的访问控制：公有类型定义外部接口、私有类型只能被本类成员函数保护、保护类型公有和保护都以保护成员出现在派生类中，而基类私有成员不能直接访问

三种访问控制的属性：公有，私有，保护

构造函数：概念与作用，执行的顺序，在类中，函数名和类名相同的函数称为构造函数。它的作用是在建立一个对象时，作某些初始化的工作，顺序:基类构造函数按继承时声明的顺序，对派生类新增的成员对象初始化调用顺序按它们在类中声明的顺序，执行派生类的构造函数体中的内容

析构函数：概念与作用，执行的顺序，对象所在的函数已调用完毕时，系统自动执行析构函数；析构函数被调用；顺序与其建立时的顺序相反，即后构造的对象先析构。

构造函数对数据成员初始化的方法：无参数()，有参数(…)，初始化列表{}

复制构造函数:1.当用类的一个对象去初始化该类的另一个对象时

2.如果函数的形参是类的对象，调用函数时，进行形参和实参结合时

3.如果函数的返回值是类的对象，函数执行完成返回调用者时

Ch5

静态数据成员：P154 5.3.1，静态数据成员是类的一部分，不是某个类对象一部分的变量；初始化： “类名：：成员名”，

静态数据成员的使用例5-4：具有静态数据成员的Point类

#include <iostream>

using namespace std;

class Point

{

public:

    Point(int x = 0, int y = 0) : x(x), y(y)

    {

        count++;

    }

    Point(Point &p)

    {

        x = p.x;

        y = p.y;

        count++;

    }

    ~Point() { count--; }

    int getX() { return x; }

    int getY() { return y; }

    void showCount()

    {

        cout << " Object count =" << count << endl;

    }

private:

    int x, y;

    static int count;

};

int Point::count = 0;

int main()

{

    Point a(4, 5);

    cout << "Point A: " << a.getX() << "," << a.getY();

    a.showCount();

    Point b(a);

    cout << "Point B: " << b.getX() << "," << b.getY();

    b.showCount();

    return 0;

}

友元函数的概念：友元函数是在类中用关键词Friend修饰的非成员函数

友元函数的定义与使用：

1）必须在类的说明中说明友元函数，说明时以关键字friend开头，后跟友元函数的函数原型，友元函数的说明可以出现在类的任何地方，包括在private和public部分；

2）注意友元函数不是[类的成员函数](https://baike.baidu.com/item/%E7%B1%BB%E7%9A%84%E6%88%90%E5%91%98%E5%87%BD%E6%95%B0" \t "_blank)，所以友元函数的实现和普通函数一样，在实现时不用"::"指示属于哪个类，只有成员函数才使用"::"作用域符号；

3）友元函数不能直接访问类的成员，只能访问对象成员，

4）友元函数可以访问对象的私有成员，但普通函数不行；

5）调用友元函数时，在实际参数中需要指出要访问的对象，

6）类与类之间的友元关系不能继承。

7）一个类的成员函数也可以作为另一个类的友元，但必须先定义这个类。

Ch6

动态内存分配：new 与 delete

new 后面跟一个数据类型，如果要求的元素多于一个，需要加上 []

大部分情况下，动态分配的内存只在程序运行的具体的阶段内才有用，一旦它不再被需要，就要被释放掉，以便后面的内存分配能够使用。

Ch7

继承中的访问控制：P256 7.2 公有继承，私有继承，保护继承

派生类与基类的类型兼容规则：任何需要使用基类的地方, 都可以使用派生类作为代替;

代替基类的派生类, 只能使用从基类继承来的成员;

（1）派生类对象可以隐含转换为基类对象，即用派生类对象中从基类继承来的成员，逐个赋值给基类对象的成员:

(2)派生类的对象也可以初始化基类对象的引用:

(3)派生类对象的地址也可以隐含转换为指向基类的指针P262 7.3

派生类构造函数的执行顺序：7.4.1小节，P266 例7-4

Ch8

运算符的重载，哪些运算符可以进行重载：除了类属关系运算符”.”，成员指针运算符“.\*”，作用域分辨符“::”，三目运算符“?:”以外的都可以

虚函数的概念与作用：虚函数是一个可以被覆盖的函数，派生类中的虚函数会覆盖基类中同名的函数, 并隐藏基类中所有同名的其它重载函数;

哪些函数可以定义为虚函数：析构函数 普通的成员函数

Ch9

模板的概念：模板是C++支持参数化程序设计的工具，通过它可以实现参数化多态性。所谓参数化多态性，就是将程序所处理的对象的类型参数化.使得一段程序可以用于处理多种不同类型的对象

模板的两个种类：模板分为类模板和函数模板

编程

函数：递归函数

P75 例3-9，

//用递归法计算从n个人中选择k个人组成一个委员会的不同组合数

#include <iostream>

using namespace std;

int comm(int n, int k)

{

    if (k > n)

        return 0;

    else if (n == k || k == 0)

        return 1;

    else

        return comm(n - 1, k) + comm(n - 1, k - 1);

}

int main()

{

    int n, k;

    cout << "Please enter two integers n and k:";

    cin >> n >> k;

    cout << "C(n,k)=" << comm(n, k) << endl;

    return 0;

}

P97 习题3-13

//用递归求Fibonacci级数

#include <iostream>

using namespace std;

int fib(int n);

int main ()

{

    int n, answer;

    cout << "Enter number:";

    cin >> n;

    cout << "\n\n";

    answer = fib(n);

    cout << answer << "is the" << n << "the Fib number\n";

    return 0;

}

int fib(int n)

{

    cout << "Processing fib(" << n << ")...";

    if (n<3){

        cout << "Return 1!\n";

        return (1);

    }

    else{

        cout << "Call fib(" << n - 2 << ") and fib (" << n - 1 << ").\n";

        return (fib(n - 2) + fib(n - 1));

    }

}

类与对象

P144 习题4-11

//定义并实现一个矩形类，有长宽两个属性，有成员函数计算面积

#include <iostream>

using namespace std;

class Rectangle

{

public:

    Rectangle(float l, float w)

    {

        length = l;

        width = w;

    }

    ~Rectangle()

    {

    }

    float getArea()

    {

        return length \* width;

    }

    float getlength()

    {

        return length;

    }

    float getwidth()

    {

        return width;

    }

private:

    float length;

    float width;

};

int main()

{

    float length, width;

    cout << "put in the length:";

    cin >> length;

    cout << "put in the width";

    cin >> width;

    Rectangle r(length, width);

    cout << "the long is " << length << ", the wide is " << width << " and the square is:" << r.getArea() << endl;

    return 0;

}

4-13

//定义一个circle类用成员函数计算圆的面积

#include <iostream>

using namespace std;

class Circle

{

public:

    Circle(float radius)

    {

        this->radius = radius;

    }

    ~Circle()

    {

    }

    float getArea()

    {

        return 3.14 \* radius \* radius;

    }

private:

    float radius;

};

int main()

{

    float radius;

    cout << "please put in :";

    cin >> radius;

    Circle p(radius);

    cout << "radius is " << radius << "'s circle area is " << p.getArea() << endl;

    return 0;

}

多态性：运算符重载的两种方式

P309 例8-1，

//复数类加减法运算重载为成员函数形式

#include <iostream>

using namespace std;

class Complex

{

public:

    Complex(double r=0.0,double i = 0.0):real(r),imag(i){}

    Complex operator+(const Complex &c2)const;

    Complex operator-(const Complex &c2)const;

    void display() const;

private:

    double real;

    double imag;

};

Complex Complex::operator+(const Complex &c2)const{

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

}

Complex Complex::operator-(const Complex &c2)const{

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

}

void Complex::display()const{

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

}

int main(){

    Complex c1(5, 4), c2(2, 10), c3;

    cout << "c1=";

    c1.display();

    cout << "c2=";

    c2.display();

    c3 = c1 - c2;

    cout << "c3=c1-c2=";

    c3.display();

    c3 = c1 + c2;

    cout << "c3=c1+c2=";

    c3.display();

    return 0;

}

/\*输出结果

c1=(5,4)

c2=(2,10)

c3=c1-c2=(3,-6)

c3=c1+c2=(7,14)\*/

P313 例8-3

//以非成员函数形式重载Complex的加减法运算和<<运算符

#include <iostream>

using namespace std;

class Complex

{

public:

    Complex(double r=0.0,double i = 0.0):real(r),imag(i){}

    friend Complex operator+(const Complex &c1, const Complex &c2);

    friend Complex operator-(const Complex &c1, const Complex &c2);

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

private:

    double real;

    double imag;

};

Complex operator+(const Complex &c1,const Complex &c2){

    return Complex(c1.real + c2.real, c1.imag + c2.imag);

}

Complex operator-(const Complex &c1,const Complex &c2){

    return Complex(c1.real - c2.real, c1.imag - c2.imag);

}

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

    cout << "(" << c.real << "," << c.imag << ")";

    return out;

}

int main()

{

    Complex c1(5, 4),c2(2, 10), c3;

    cout << "c1=" << c1 << endl;

    cout << "c2=" << c2 << endl;

    c3 = c1 - c2;

    cout << "c3=c1-c2=" << c3 << endl;

    c3 = c1 + c2;

    cout << "c3=c1+c2=" << c3 << endl;

    return 0;

}