第四章 函 数

本笔记记录课本第三章自己不是很会的东西，知识点不会很全

参数的传递机制

值的传递（值调用）

包括*变量值调用*和*地址值调用*

1.传值调用

（比较简单）
特点：形参的变化不会影响实参

2.传地址调用

特点：被调用函数的形参是**指针**，调用函数的实参给出的是**地址值**，**能通过改变形参所指向变量的值影响实参**

传地址的示例

```
void swap(int *px,int *py)      //函数原型
void main()
{
        int a(15),b(18);
        cout<<"主函数第一次输出a="<<a<<" "<<"b="<<b<<endl;
        swap(&a,&b);        //实参是变量的地址
        cout<<"主函数第二次输出a="<<a<<" "<<"b="<<b<<endl;
}
void swap(int *px,int *py)
{
        int temp;
        temp=*px;*px=*py;*py=temp;
        cout<<"swap函数输出:*px="<<*px<<" "<<"*py="<<*py<<endl;
}
```

数组作参数

1.形参和实参都是数组

特点：被调用函数的形参**写数组的定义**，调用函数的实参直接**写数组名**，此时数组名被自动转换成指向实参数组的指针，即**形参和实参处理的是同一个数组**

2.函数原型与调用

```
void swap(int a[10]);          //函数原型
//————————分割线————————
void main()
{
        int s[10]={2,3,4,5,6,7,8,9,1,1};
        sort(s);                    //函数调用
}
```

引用传递（引用调用）

引用做形参时，可以被认为是**实参的一个别名，传递后形参和实参对应同一个量，形参的任何操作都直接作用于实参**

函数原型（以swap函数为例）

```
void swap(ing &x,int &y);
```

有个地方注意一下：*一个函数只能返回一个值，如果需要从函数返回多个值时，可以用引用实现*(课本P85)，考试时灵活一点

函数参数的缺省

（就是参数的初始值，又称为默认）

注意：

- 缺省值可以都有，也可以部分有
- **当函数部分有缺省值的时候，有缺省值得参数必须列在形参表右边**
- 若函数既有原型声明又有定义时，缺省值只需在原型声明中给出，函数定义时的参数表中不能再给出

函数与指针

函数返回指针和返回引用

指针函数

1.概念：当一个函数被调用时，若返回值是一个指针（地址），则称该函数为指针函数。指针函数主要用在函数结束时需要大量的数据从被调用函数返回调用函数的情况（有多个顺序存

放的值，例如数组、字符串需要返回时，可以用指针函数）
2.定义形式：

```
类型 *函数名(参数)
```

示例

```
类型 *函数名(参数)
4.15  输入0~6六个数字,输出对应的英文星期表示。
#include <iostream.h>
char *fun( int n)                                    //定义指针函数
{   static char *pstr[ ] = {"Sunday","Monday","Tuesday","Wednesday","Thursday","Fri-
    day","Saturday"};
```

```
    return pstr[n];
}
int main()
{   int n;
    cout << "输入0-6:"; cin >> n;
    if(n<0||n>6) cout << "输入错!\n";
    else cout << fun(n) << endl;
    return 0;
}
```

执行结果示例1：

输入0-6:3

Wednesday

执行结果示例2：

输入0-6:8

输入错!

3.注意事项：**不能吧局部作用域内数据的地址作为返回值**（所以在指针函数内定义字符型指针数组pstr时声明为static）

返回引用

返回引用与返回值的区别：

- 处理函数返回值的时候，系统生成一个返回值的副本（临时变量），由这个临时变量给到调用函数的接受变量。
- 处理函数返回引用的时候，不生成副本，直接将引用的值给到调用函数（说人话就是能改变参数的值）

定义类型：

```
类型 &函数名(参数)
```

函数指针

1.定义：数组名是表示数组首地址的地址常量，相似的，在C++中函数名是代表函数的入口地址的地址常量。因此可以声明一个指向函数的指针，让这个指针指向函数，通过这个指针调用函数，这个指针就成为函数指针。
2.声明形式：

```
类型 (*指针)(参数表)
```

3.主要作用：在函数间传递函数。

函数重载

定义

函数名相同，**参数类型或参数个数不同**的两个函数叫函数重载

判断标准

- 1.函数的参数个数、参数类型、参数顺序三者至少一个不同
- 2.如果只有函数返回值不同，不是函数重载；返回值不同，且参数不同，可以作为函数重载

运算符重载

#重载

1.概念：运算符的重载是一种特殊的函数重载，必须定义一个函数，并告诉C++编译器，当遇到该运算符时就调用此函数来行使运算符功能。这个函数叫做运算符重载函数（常为类的成员函数）

2.基本格式：

```
返回值类型 类名::operator重载的运算符(参数表)
{
......
}       //operator是关键字，它与重载的运算符一起构成函数名
```

3.运算符重载的两种方法：

- 类的成员函数
- 友元函数

二元运算符重载

1.类内重载

注意，一个一元运算符重载为类的成员函数时，参数个数为0，一个二元运算符重载为类的成员时，参数个数必为1

```
#include <iostream>
using namespace std;

class Point{
public:
    Point(){};
    Point (int x, int y): x(x),y(y) {};
    Point operator+(const Point &b){ //类内重载，运算符重载函数作为类的成员函数
        Point ret;
        ret.x = this->x + b.x;
        ret.y = this->y + b.y;
        return ret;
    }
    int x,y;
};

int main() {
    Point a(2,4),b(5,3);
    Point c = a + b;        //这里c++编译器会，自动去找 + 运算符的重载函数
        cout<< "x :" << c.x << endl;
    cout<<"y :" << c.y << endl;
```

```
}
```

运行结果：x:7 y:7

重点：**运算符重载是类内重载时，运算符重载函数作为类的成员函数，以上述代码为例 a + b 相当于 a 对象调用+方法并且传入参数时 b 对象。**

2.类外重载（用友元函数的方法实现）友元函数

（由于友元函数不是类的成员函数，不能使用this指针指向隐含参数，因此友元函数必须有两个参数）

（如果不使用友元函数，则定义形式为 返回类型 类名::operator 运算符（形参表））

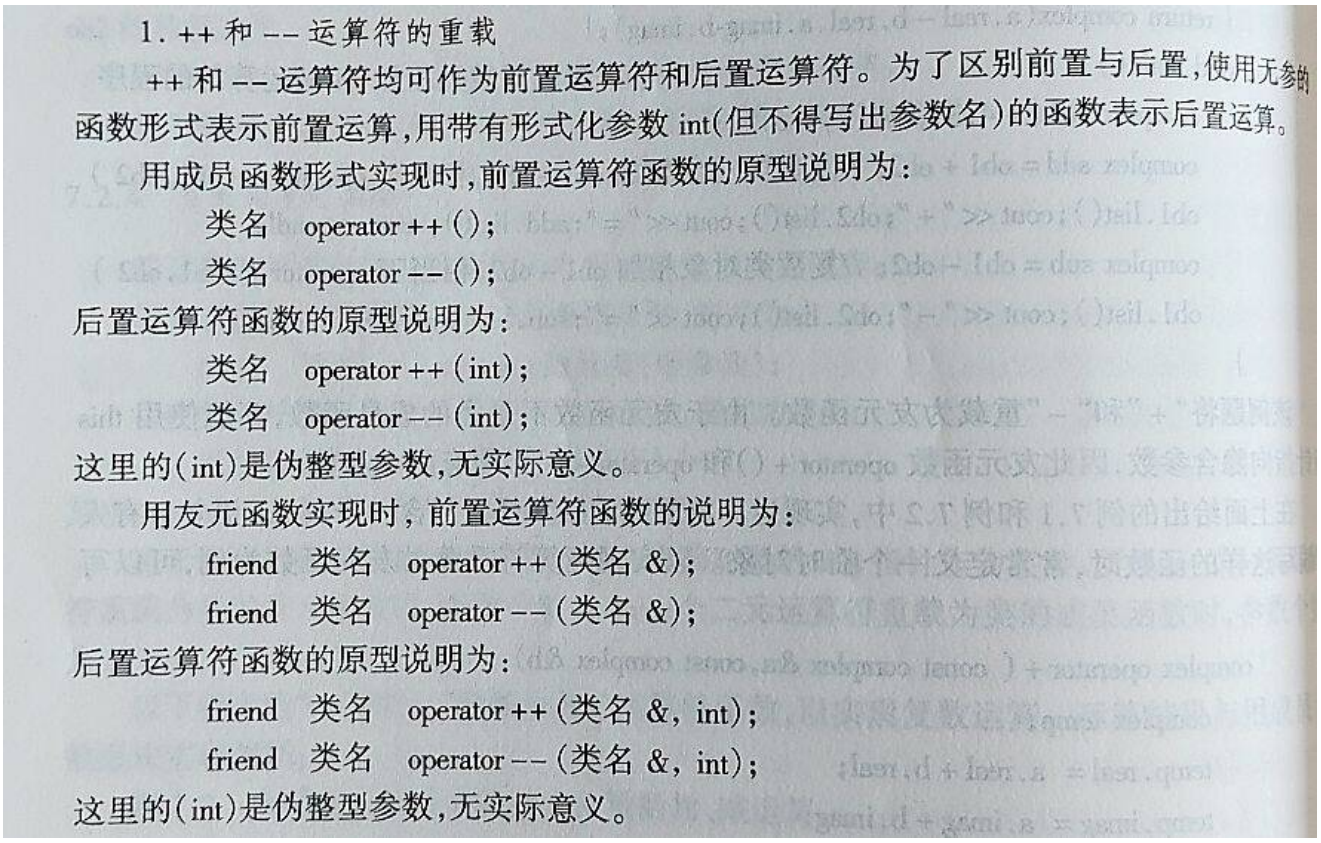
```
#include <iostream>
using namespace std;

class Point{
public:
    Point(){};
    Point (int x, int y): x(x),y(y) {};
    friend Point operator+(const Point &, const Point &);    //声明类的友元函数
    int x,y;
};

Point operator+(const Point &a,const Point &b){//类外重载,运算符重载函数作为类的友元函数
    Point ret;
    ret.x = a.x + b.x;
    ret.y = a.y + b.y;
    return ret;
}

int main() {
     Point a(2,4),b(5,3);
    Point c = a + b;
        cout<< "x :" << c.x << endl;
    cout<<"y :" << c.y << endl;
}
```

一元运算符重载

（见课本P194）

1.插入运算符重载>> and 提取运算符重载<<

2.前置运算符重载++ and 后置运算符重载++



1. ++ 和 -- 运算符的重载

++ 和 -- 运算符均可作为前置运算符和后置运算符。为了区别前置与后置，使用无参的函数形式表示前置运算，用带有形式化参数 int(但不得写出参数名)的函数表示后置运算。

用成员函数形式实现时，前置运算符函数的原型说明为：

类名 operator ++ ();

类名 operator -- ();

后置运算符函数的原型说明为：

类名 operator ++ (int);

类名 operator -- (int);

这里的(int)是伪整型参数，无实际意义。

用友元函数实现时，前置运算符函数的说明为：

friend 类名 operator ++ (类名 &);

friend 类名 operator -- (类名 &);

后置运算符函数的原型说明为：

friend 类名 operator ++ (类名 &, int);

friend 类名 operator -- (类名 &, int);

这里的(int)是伪整型参数，无实际意义。

注意：使用成员函数进行重载的时候，常常返回当前对象（*this表示），用友元函数进行重载的时候，常常返回被操作的参数对象。

3 =等号运算符重载

不能重载的运算符!

```
(1)"."（类成员访问运算符）
(2)" .*"（类成员指针访问运算符）
(3) "::"（域运算符）
(4)"siezof"（长度运算符）
(5) " ?:"（条件运算符）
------不能重载为友元函数的运算符------
(1)=
(2)→
(3)[]
(4)()
```

重载运算符的规则

- 1.只能重载C++中已有的可重载的运算符，不能建立新的运算符
- 2.重载运算符时不能改变原运算符操作数的个数、原有运算符的优先级和结合性
- 3.C++内部已经对所以对象重载了两个运算符，即赋值运算符"="和取地址运算符"&"

- 4.如果重载了某个运算符（如"="）并不意味着重载了相关的运算符（如"+="、"-="）

函数模板

1.定义：模板是支持参数化的工具，即将一段程序所处理的对象的类型参数化，分为函数模板和类模板
2.定义形式：

```
template <class T>
类型 函数名(参数表)
        函数体
```

T代表在函数模板中使用的通用类型，既可以代表系统提供的基本数据类型，也可以代表用户自定义的类型。
class 表示模板参数的类型
3.注意：函数模板不是一个完全的函数，，而是一类函数的抽象，使用时将模板参数T实例化即可

4.代码：

```
template <typename T>
T max(T a, T b)
{
    return a > b ? a : b;
}
//上述代码实现了一个可以接受任意类型的函数模板`max`，其返回两个参数中的最大值。

//使用函数模板时，用指定的类型替换函数模板中的模板参数即可。例如，以下代码使用函数模板`max`来查找两个整数中的最大值：
int x = 10, y = 20;
int max_val = max(x, y);  // 使用函数模板max查找两个整数中的最大值

double a=10.0,b=20.0;
double max_val1 = max(a,b);

//-----分界线-------
//[函数模板实例化]可以让编译器自动推导，也可以在调用的代码中显式的指定。
int x = 10, y = 20;
int max_val<int> = max(x, y);  // 使用函数模板max查找两个整数中的最大值

double a=10.0,b=20.0;
double max_val1<double> = max(a,b);
```

内联函数

使用内联函数主要是为了提高程序的运行效率

在类内定义的成员函数默认为内联成员函数，但是不是所有的成员函数都是内联函数

作用域、生存期与可见性

编译预处理