第6章 函数

——C++程序设计

对外经济贸易大学 雷擎
leiqing@uibe.edu.cn

内容

- 6.1 函数基础
- 6.2 参数传递
- 6.3 返回类型和return语句
- 6.4 函数重载
- 6.5 特殊用途语言特性
- 6.7 函数指针
- 6.8 递归函数 //补充

6.1 函数基础

- 定义无参函数的一般形式为：

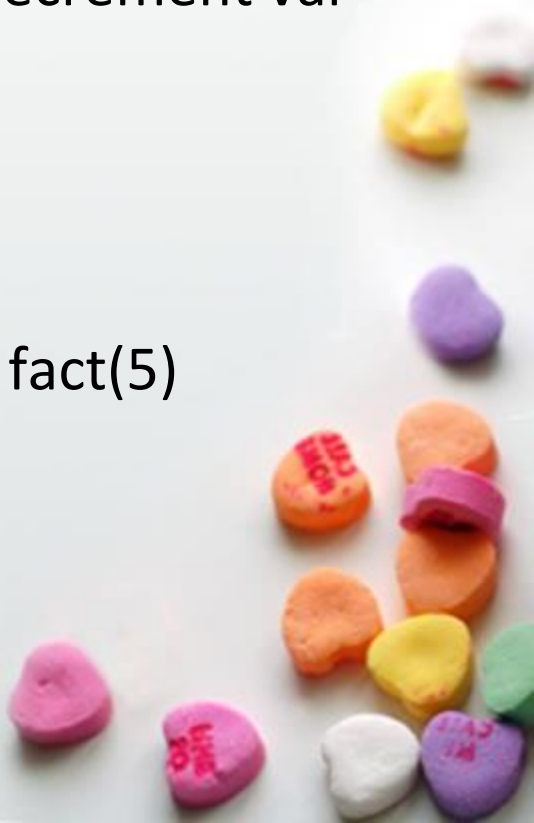
  类型标识符 函数名([void]) {
  　声明部分
  　语句
  }

- 定义有参函数的一般形式为：

  类型标识符 函数名(形式参数表列) {
  　声明部分
  　语句
  }

编写和调用函数

```
int fact(int val){
    int ret = 1; // local variable to hold the result as we calculate it
    while (val > 1)
        ret *= val--; // assign ret * val to ret and decrement val
    return ret; // return the result
}

int main(){
    int j = fact(5); // j equals 120, i.e., the result of fact(5)
    cout << "5! is " << j << endl;
    return 0;
}
```



形参与实参

- 在定义函数时函数名后面括号中的变量名称为形式参数(formal parameter，简称形参)，

- 在主调函数中调用一个函数时，函数名后面括号中的参数称为实际参数(actual parameter，简称实参)。

形参与实参

形参

```
int fact(int val){
    int ret = 1; // local variable to hold the result as we calculate it
    while (val > 1)
        ret *= val--; // assign ret * val to ret and decrement val
    return ret; // return the result
}
```

实参

```
int main(){
    fact("hello"); // error: wrong argument type
    fact(); // error: too few arguments
    fact(42, 10, 0); // error: too many arguments
    fact(3.14); // ok: argument is converted to int
    return 0;
}
```

形参和实参一定要一一对应

形参列表

```
void f1(){ /* ... */ } // implicit void parameter list
void f2(void){ /* ... */ } // explicit void parameter list
```

both are ok

```
int f3(int v1, v2) { /* ... */ } // error
int f4(int v1, int v2) { /* ... */ } // ok
```

形参中必须每个变量都明确类型

6.1.1 局部对象

- 名字的作用域
  - 名字有效的程序区域
- 对象的生命周期
  - 是程序执行过程中该对象存在的一段时间
- 局部变量
  - 形参和函数体内定义的变量，称为局部变量
- 自动对象
  - 只存在于函数体程序块执行期间的对象
- 局部静态对象
  - 使用static类型定义的对象
  - 在程序第一次执行对象定义语句时初始化
  - 生命周期贯穿函数调用及之后的时间
  - 程序终止时才被销毁

6.1.2 函数声明

- 和其他的名字一样，函数的名字也必须在使用之前声明
- 建议在头文件中声明，在源文件中定义

  float add(float x,float y);
- 在函数声明中也可以不写形参名，而只写形参的类型

  float add(float, float);

```
int main(){
void getMax(double x, double y,double z); // 函数的声明
// void getMax(double, double, double); //另一种写法
double a,b,c;
cin>>a>>b>>c;
getMax(a,b,c);
return 0;
}

void getMax(double x, double y,double z){
.....
}
```

6.2 参数传递

- 值传递
  - 当实参的值被拷贝给形参时，形参和实参是两个独立的对象。这样称为实参被值传递(passed by value) 或者函数被传值调用(called by value)。
- 引用传递
  - 当形参是引用类型时,对应的实参被引用传递(passed by reference)或者函数被传引用调用(called by reference)。

数组名作为参数传递

- 如果函数定义时，形参为数组名(或指针变量)，实参则是数组名。实参数组与形参数组类型应一致，如不一致，结果将出错。

- 数组名代表数组首元素的地址，并不代表数组中的全部元素。因此用数组名作函数实参时，不是把实参数组的值传递给形参，而只是将实参数组首元素的地址传递给形参。

- 用数组名作函数实参时，改变形参数组元素的值将同时改变实参数组元素的值。

用多维数组名作函数参数

- 用二维数组名作为实参和形参，在对形参数组声明时，必须指定第二维(即列)的大小，且实参第二维的大小必须与其相同。第一维的大小可以指定，也可以不指定。

```
int array[3][10]; //形参数组的两个维都指定
```

或

```
int array[][10]; //第一维大小省略
```

- 二者都合法而且等价。但是不能把第二维的大小省略。下面的形参数组写法不合法：

```
int array[][]; //不能确定数组的每一行有多少列元素
int array[3][]; //不指定列数就无法确定数组的结构
```

用多维数组名作函数参数

- 在第二维大小相同的前提下，实参数组的第一维可以与形参数组不同。例如，形参数组定义为：

```
int score[5][10];
```

- 而形参数组可以声明为：

```
int array[3][10]; //列数与形参数组相同，行数不同
```

```
int array[8][10];
```

- 如果是三维或更多维的数组，处理方法类似。

6.3 返回类型和return语句

- 无返回值函数：返回类型是void的函数
  - 可以省略return语句，系统会隐式执行return
- 有返回值函数：返回类型不是void的函数
  - 函数内每条return语句必须返回一个值，并且返回值得类型必须与函数的返回类型相同。
- 返回数组指针
  - 因为数组不能被拷贝，所以函数不能返回数组，但可以返回数组的指针或引用。

6.4 函数重载

- C++允许用同一函数名定义多个函数，这些函数的参数个数和参数类型不同。这就是函数的重载(function overloading)。即对一个函数名重新赋予它新的含义，使一个函数名可以多用。

定义重载函数

```
Record lookup(const Account&); // find by Account
Record lookup(const Phone&); // find by Phone
Record lookup(const Name&); // find by Name
```

```
Account acct;
Phone phone;
Record r1 = lookup(acct); // call version that takes an Account
Record r2 = lookup(phone); // call version that takes a Phone
```

调用重载函数

- 函数匹配(function matching)
  - 是指一个过程，把函数调用与一组重载函数中的某一个关联起来，又叫重载确定(overload resolution)

6.5.1 默认实参

- 在同一函数调用时，很多次调用中同一形参都被赋予相同的值，C++提供简单的处理办法，给此形参一个默认值，这样形参就不必一定要从实参取值了。这个反复出现的值，称为函数的默认实参(default argument)。
- 调用含有默认实参的函数时，可以包含该实参，也可以省略该实参。
- 可以为一个或多个形参定义默认值。注意：一旦某个形参被赋予了默认值，它后面的形参都必须有默认值。
- typedef string::size_type sz;
- string screen(sz ht = 24, sz wid = 80, char backgrnd = ' ');

使用默认实参调用函数

- 如果要使用默认实参，只要在调用函数的时候省略实参就可以了。

```
string window;
window = screen(); // equivalent to screen(24,80,' ')
window = screen(66);// equivalent to screen(66,80,' ')
window = screen(66, 256); // screen(66,256,' ')
window = screen(66, 256, '#'); // screen(66,256,'#')
```

默认实参声明

默认参数和函数重载不要一起用

- 默认参数可以放在函数声明或者定义中，但只能放在二者之一

- 通常我们都将默认参数放在函数声明中，因为如果放在函数定义中，那么将只能在函数定义所在地文件中调用该函数。

- 在给定作用作用域中，一个形参只能被赋予一次默认实参
- string screen(sz, sz, char = ' ');
- string screen(sz, sz, char = '*'); // error: redeclaration
- string screen(sz = 24, sz = 80, char); // ok: adds default

6.5.2 内联函数和constexpr函数

- C++提供一种提高效率的方法，即在编译时将所调用函数的代码直接嵌入到主调函数中，而不是将流程转出去。这种嵌入到主调函数中的函数称为内联函数(inline function)，又称内嵌函数。

- 指定内置函数的方法很简单，只需在函数首行的左端加一个关键字inline。

6.7 函数指针

- 函数指针的定义格式：

数据类型 (*指针变量名)(参数表);

  - 数据类型是指函数的返回值的类型

- 区分下面两个语句：

```
int (*p)(int a, int b);
//p是指向函数的指针变量，所指函数的返回值类型为整型
int *p(int a, int b);
//p是函数名，此函数的返回值类型为整型指针
```

函数指针说明

- 指向函数的指针变量不是固定指向哪一个函数的，只是表示定义了一个这样类型的变量，它是专门用来存放函数的入口地址的；在程序中把哪一个函数的地址赋给它，它就指向哪一个函数。

- 在给函数指针变量赋值时，只需给出函数名，而不必给出参数。
  - 例如，函数max的原型为：int max(int x, int y); 指针p的定义为：int (*p)(int a, int b); 则p = max;的作用是将函数max的入口地址赋给指针变量p。这时，p就是指向函数max的指针变量，也就是p和max都指向函数的开头。

函数指针说明

- 指向函数的指针变量不是固定指向哪一个函数的，只是表示定义了一个这样类型的变量，它是专门用来存放函数的入口地址的；在程序中把哪一个函数的地址赋给它，它就指向哪一个函数。

- 在给函数指针变量赋值时，只需给出函数名，而不必给出参数。
  - 例如，函数max的原型为：int max(int x, int y); 指针p的定义为：int (*p)(int a, int b); 则p = max;的作用是将函数max的入口地址赋给指针变量p。这时，p就是指向函数max的指针变量，也就是p和max都指向函数的开头。

函数指针说明

- 定义了一个函数指针，并赋值指向了一个函数后，对函数的调用可以通过函数名调用，也可以通过这个函数指针调用（即用指向函数的指针变量调用）。

- 函数指针变量常用的用途之一是把指针作为参数传递到其他函数

- 函数指针只能指向函数的入口处，而不可能指向函数中间的某一条指令。不能用*(p+1)来表示函数的下一条指令。

函数指针说明

- 在程序中，函数指针变量p可以先后指向不同的函数，但必须指向与其函数类型和参数一致的函数，一个函数不能赋给一个不一致的函数指针（即不能让一个函数指针指向与其类型、参数不一致的函数）。例如：

```
int fn1(int x, int y);
int fn2(int x);

int (*p1)(int a, int b);
int (*p2)(int a);

p1 = fn1; //正确
p2 = fn2; //正确
p1 = fn2; //产生编译错误
```

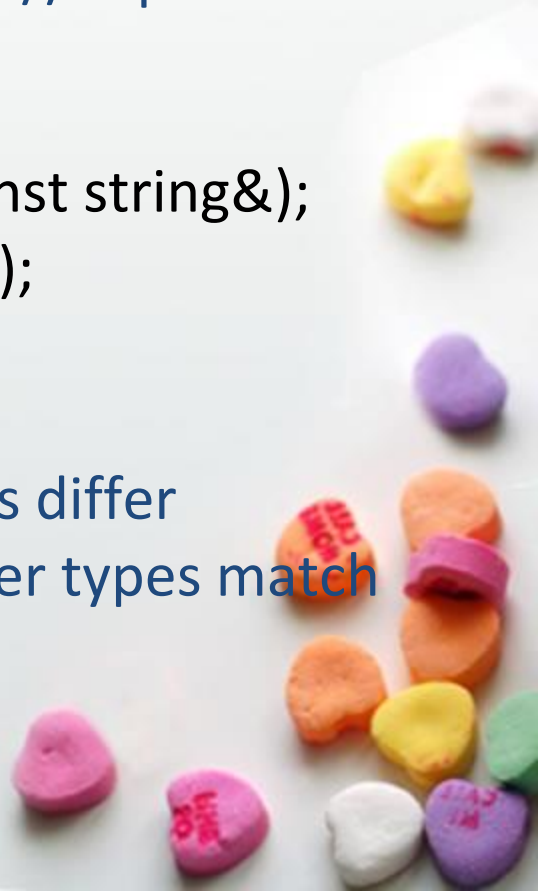
函数指针定义和赋值

```
// compares lengths of two strings
bool lengthCompare(const string &, const string &);

// pf points to a function returning bool that takes two const string
    references
bool (*pf)(const string &, const string &); // uninitialized

pf = lengthCompare; // pf now points to the function named lengthCompare
pf = &lengthCompare; // equivalent assignment: address-of operator is
    optional
```

使用函数指针

- bool b1 = pf("hello", "goodbye"); // calls lengthCompare
- bool b2 = (*pf)("hello", "goodbye"); // equivalent call
- bool b3 = lengthCompare("hello", "goodbye"); // equivalent call

- string::size_type sumLength(const string&, const string&);
- bool cstringCompare(const char*, const char*);
- pf = 0; // ok: pf points to no function
- pf = sumLength; // error: return type differs
- pf = cstringCompare; // error: parameter types differ
- pf = lengthCompare; // ok: function and pointer types match exactly



6.8 递归函数

- 在调用一个函数的过程中又出现直接或间接地调用该函数本身，称为函数的递归(recursive)调用。C++允许函数的递归调用。

```
int f(int x)
{
   int y, z;
   z=f(y); //在调用函数f的过程中，又要调用f函数
   return (2*z);
}
```

内容

- 6.1 函数基础
- 6.2 参数传递
- 6.3 返回类型和return语句
- 6.4 函数重载
- 6.5 特殊用途语言特性
- 6.7 函数指针
- 6.8 递归函数

Q & A