C++ Primier Plus Stephen Prata

第一章 开始学习C++

绪

1.函数原型：只描述函数接口。描述的发送给函数的信息和返回的信息。  
2.函数定义：包含了函数的代码。应在首次使用函数之前提供其原型，通常的做法是把原型放在main()函数定义的前面。  
3.在C和Pescal中，所有的变量声明通常都位于函数或过程的开始位置，但C++没有这种限制，C++可以在首次使用变量前声明它。

using

要访问名称空间中的方法（例如cout）  
1.将using namespace std;放在函数定义之前，让文件中的所有函数都能使用名称空间std中的所有元素。  
2.将using namespace std;放在特定的函数定义中，让该函数能够使用名称空间std中的所有元素。  
3.在特定的函数中使用类似using std::cout;这样的编译指令，而不是using namespace std;让该函数能够使用指定的元素，如cout。  
4.完全不使用编译指令using,而在需要使用名称空间std中的元素时，使用前缀std::,如下所示：std::cout << “i” << std::endl;

#include<iostream>  
using namespace std;

//该文件的所有函数均可使用空间名称std中的所有元素   
int main()  
{  
 int a;  
 cout << "Enter the a: "<< endl;  
 cin >> a;  
 cout << a;  
 **return** 0;  
}

第二章 处理数据

无符号类型

#include<iostream>  
using namespace std;  
#define ZERO 0  
  
int main()  
{  
 short Sam = SHRT\_MAX; //#define SHRT\_MAX 32767  
 unsigned short Sue = Sam; //unsigned = unsigned int  
 Sam += 1;  
 Sue += 1;  
 cout << "Sam= " << Sam << endl;//-32768  
 cout << "Sue= " << Sue << endl;//32768  
 Sam = Sue = ZERO;  
 Sam -= 1;  
 Sue -= 1;  
 cout << "Sam= " << Sam << endl;//-1  
 cout << "Sue= " << Sue << endl;//65535  
 //0-> +32767 -> (+32768不存在) -32768 重置点 -> -1 -> 0  
 **return** 0;  
}

第三章 复合类型

数组

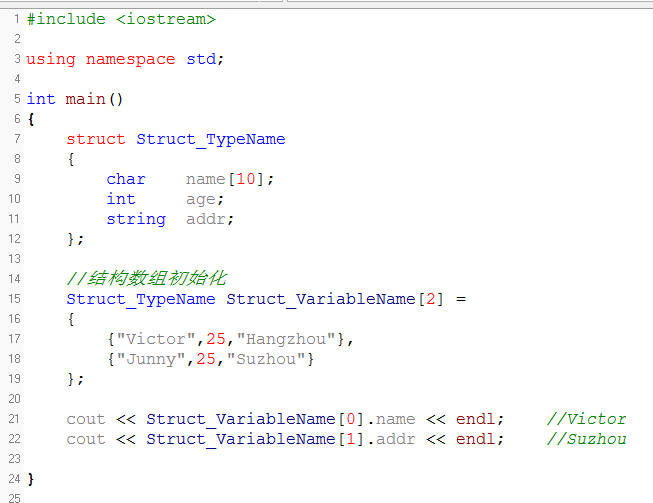
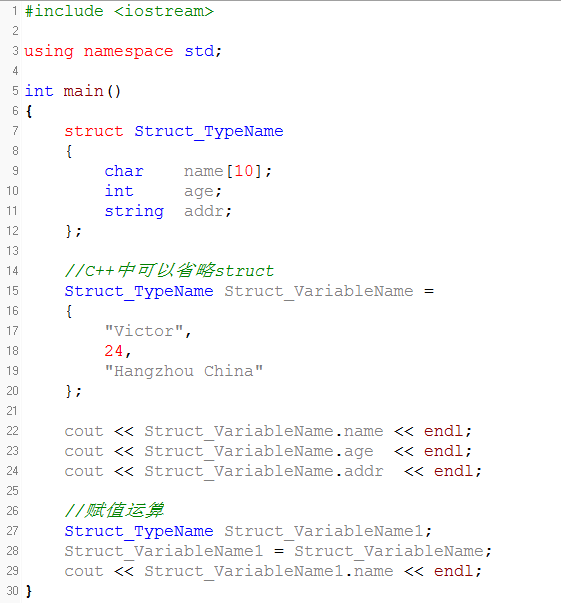
#include<iostream>  
using namespace std;  
#define ZERO 0

int main()  
{  
 int cards[4] = {1,2,3,4};  
 int totalZero[100] = {0};//所有的数组元素都是0  
 int onlyOne[100] = {1}; //onlyOne[0]=1;其余都是0  
 short things[] = {1,2,3,4};  
 cout << sizeof(things)/sizeof(short)<< endl;//4  
 //C++11数组初始化方法  
 //可省略=  
 unsigned int counts[10] = {}; //所有元素设置为0   
 double earnings[4]{1.0.20.5,4.0,4.0};  
 int earnings[4]{};  
 **return** 0;  
}

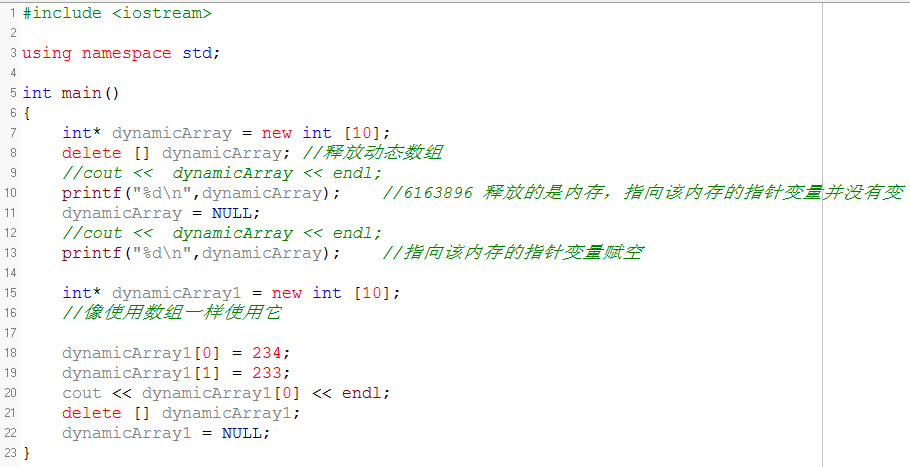
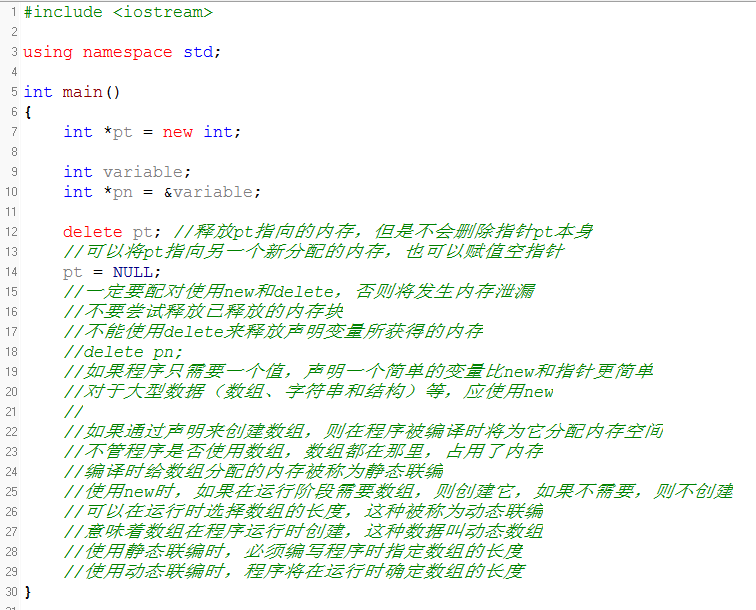
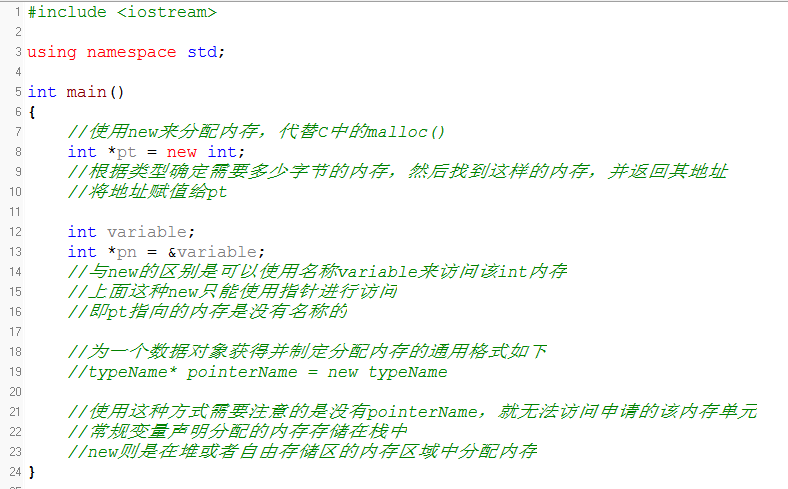
String类

#include<iostream>  
using namespace std;  
  
int main()  
{  
 string str1;  
 string str2 = "VICTOR HUGO ";  
  
 //string对象和字符数组相同  
 //可以使用数组表示法来访问存储在string对象中的字符  
 //可以将string对象声明为简单变量，而不是数组   
 cout << str2 << endl; //VICTOR HUGO   
 cout << str2[2] << endl; //C  
 //数组不能简单赋值，但是string对象可以   
 str1 = str2;  
 cout << str1 << endl; //VICTOR HUGO   
 cout << str1[4] << endl; //O  
 //还可以+-操作   
 str1 += str2;  
 cout << str1 << endl; //VICTOR HUGO VICTOR HUGO   
 str1 += "JUNNY";  
 cout << str1 << endl; //VICTOR HUGO VICTOR HUGO JUNNY  
 **return** 0;  
}

结构体

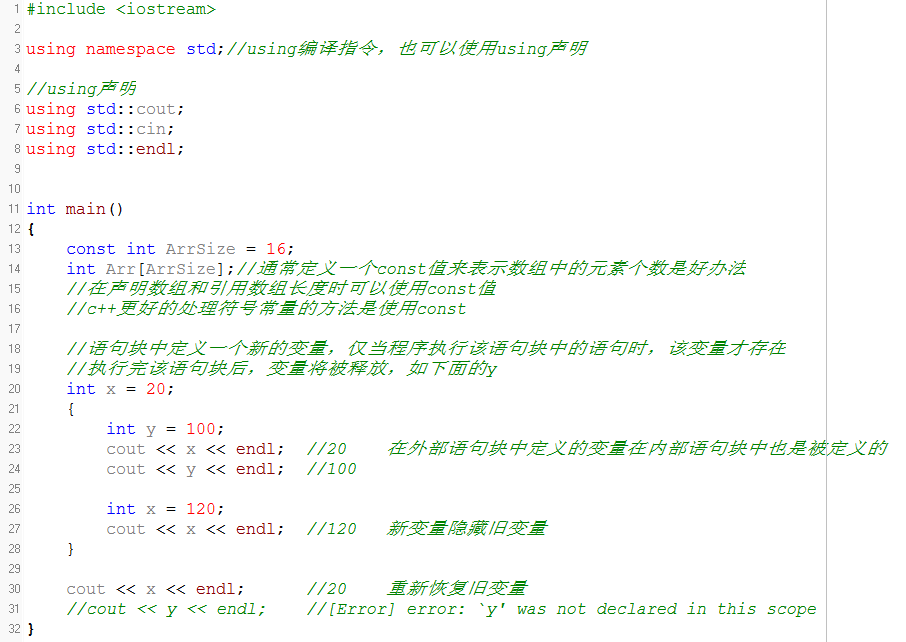


指针

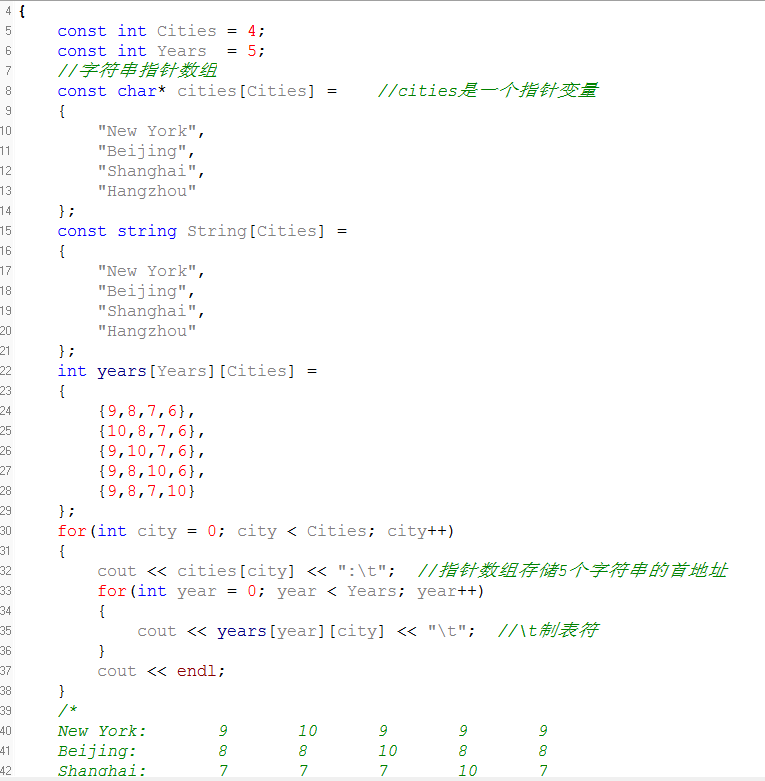
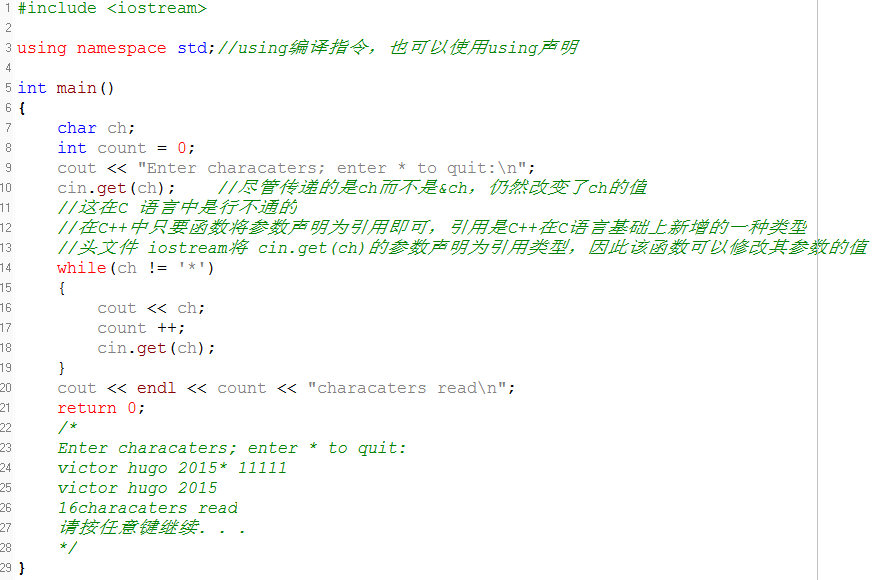
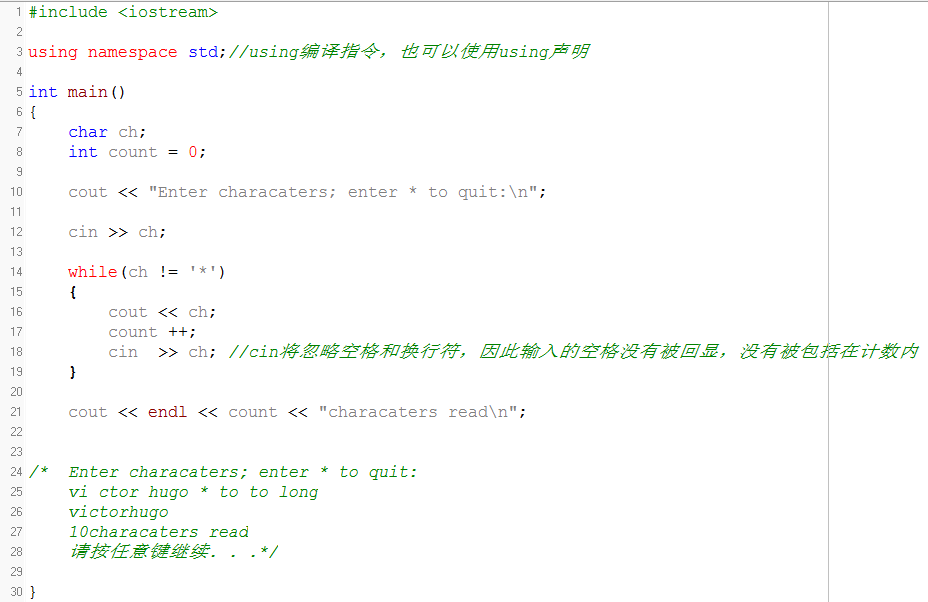
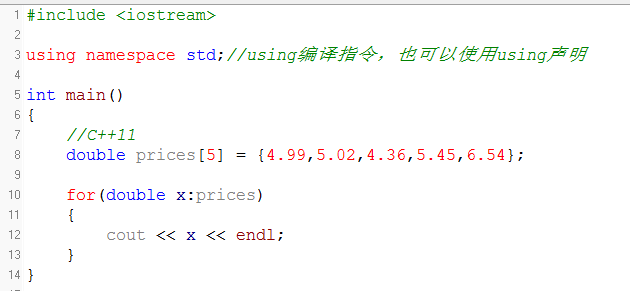


第四章 循环和关系表达式

复合语句（语句块）

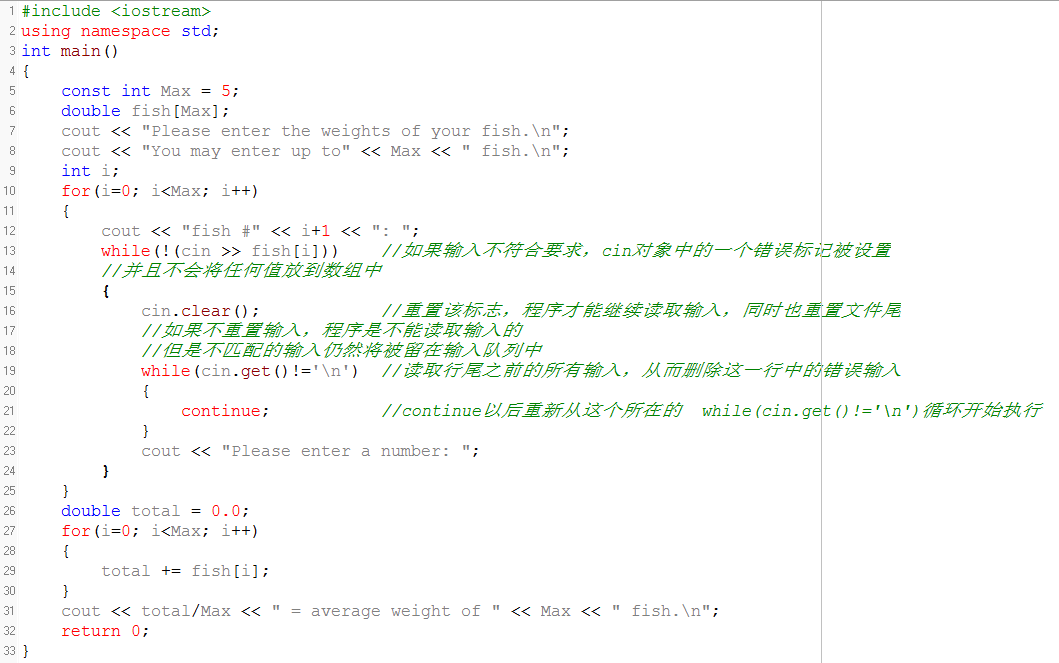
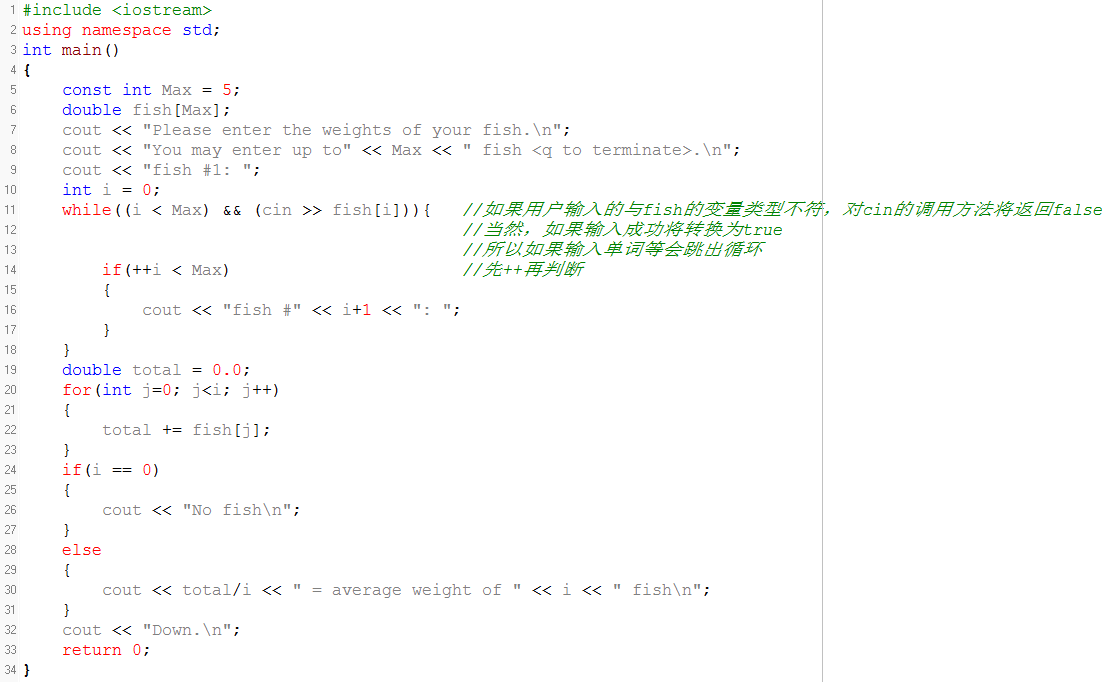


循环和文本输入

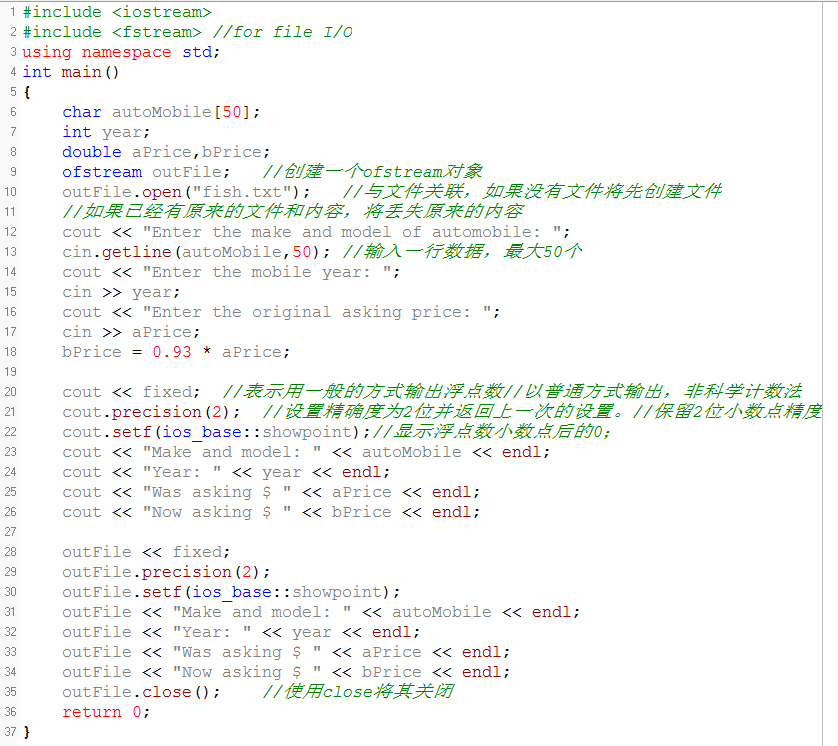


第五章 分支语句和逻辑运算符

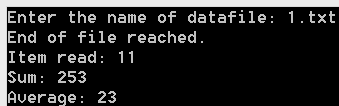
读取数字的循环



简单的文件输入/输出



#include <iostream>  
#include <fstream> //for file I/O  
#include <cstdlib> //support for exit()   
using namespace std;  
int main()  
{  
 char fileName[50];  
 ifstream inFile; //创建一个ifstream对象   
 cout << "Enter the name of datafile: ";  
 cin.getline(fileName,50);  
 inFile.open(fileName); //除非在输入的文件名中包含路径，否则程序将在可执行文件所属的文件夹中寻找   
 **if**(!inFile.is\_open()) //打开失败   
 {  
 cout << "Could nont open the file " << fileName << endl;  
 cout << "Program terminating.\n";  
 exit(EXIT\_FAILURE);  
 }  
 double value;  
 double sum = 0.0;  
 int count = 0;  
  
 inFile >> value;  
 **while**(inFile.good()) //输入正确并且不是EOF   
 {  
 ++count;  
 sum += value;  
 inFile >> value; //获取下一个输入值   
 }  
 **if**(inFile.eof()) //读取文件的EOF,将返回true   
 {  
 cout << "End of file reached.\n";  
 }  
 **else if**(inFile.fail()) //如果读取的类型不匹配，方法fail()将返回true，如果遇到了EOF，该方法也将返回true   
 {  
 cout << "Input terminated by data mismatch.\n";  
 }  
 **else** {  
 cout << "Input terminated for unkown reason.\n";  
 }  
  
 **if**(count == 0)  
 {  
 cout << "No data processed.\n";  
 }  
 **else** {  
 cout << "Item read: " << count << endl;  
 cout << "Sum: " << sum << endl;  
 cout << "Average: " << sum/count << endl;  
 }  
 inFile.close();  
 **return** 0;  
}

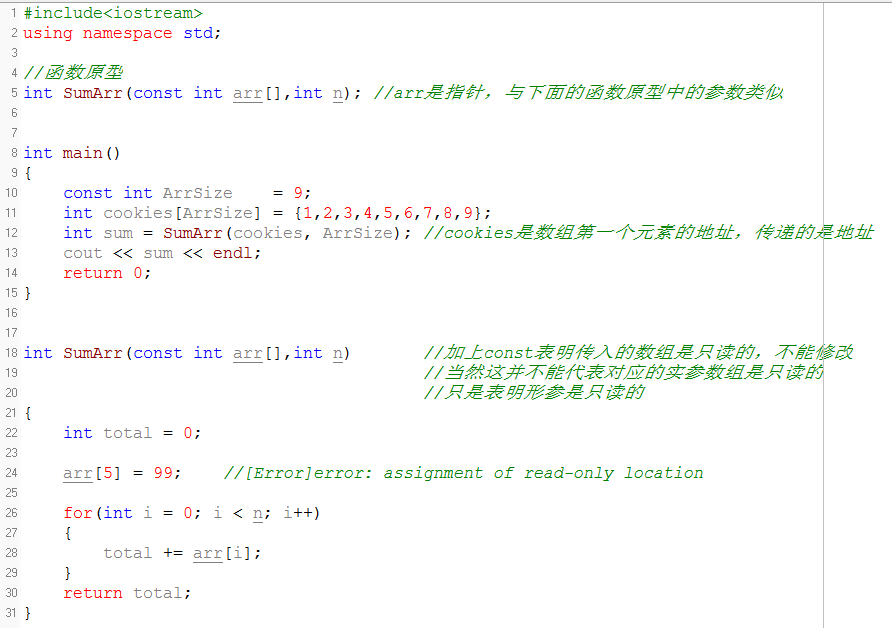


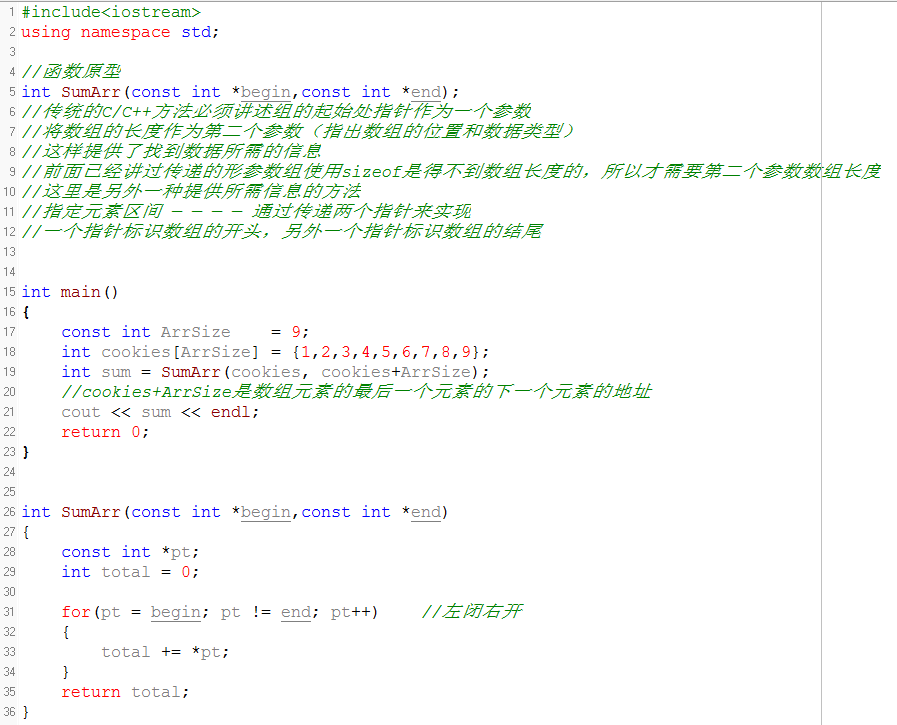
第六章 函数——C++的编程模块

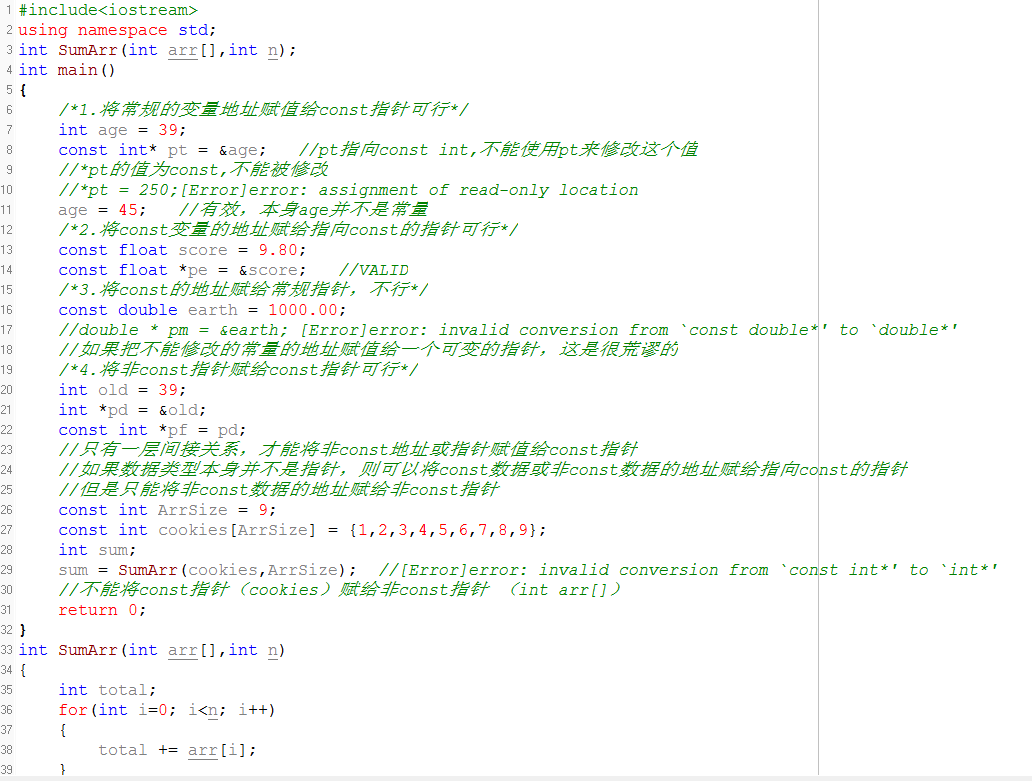
函数和数组

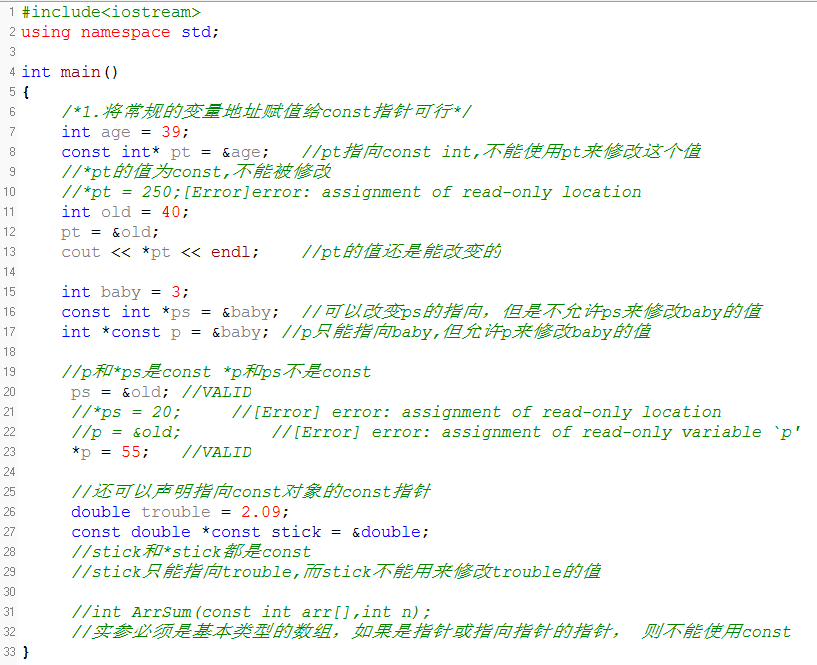






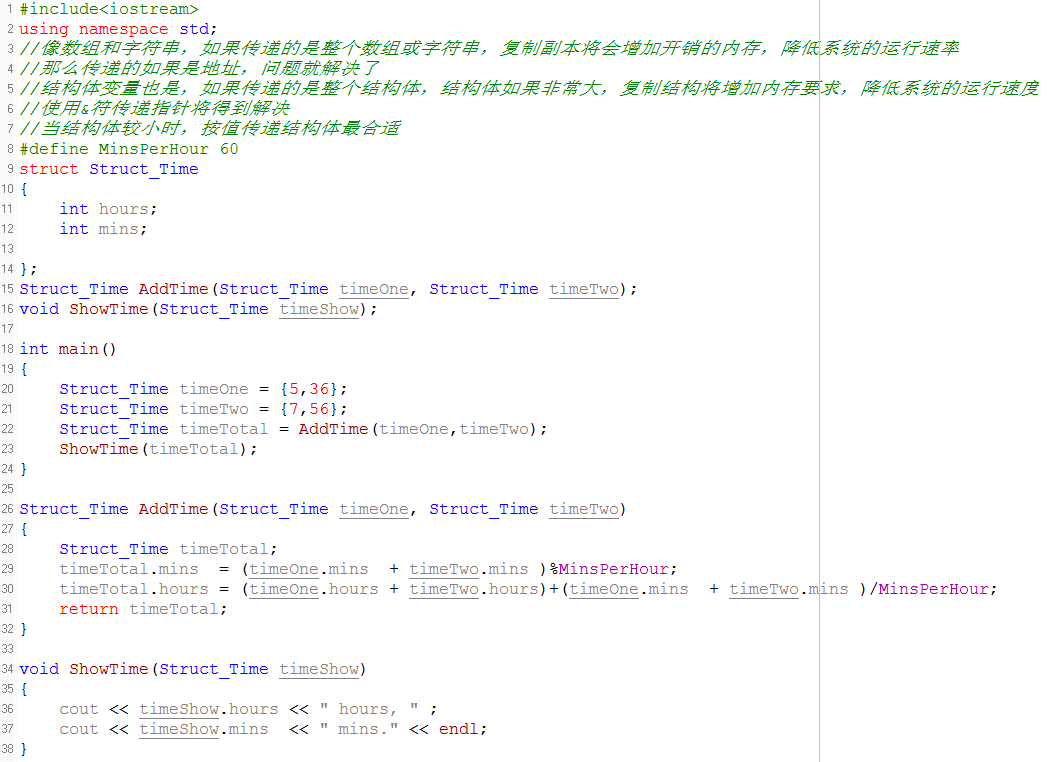


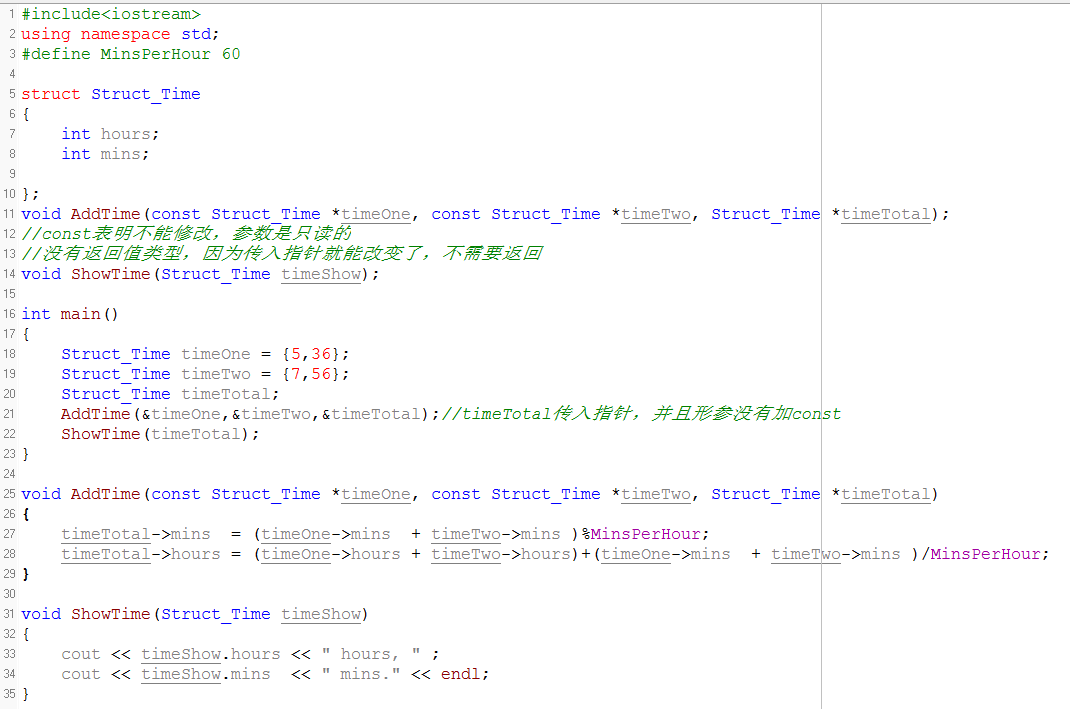




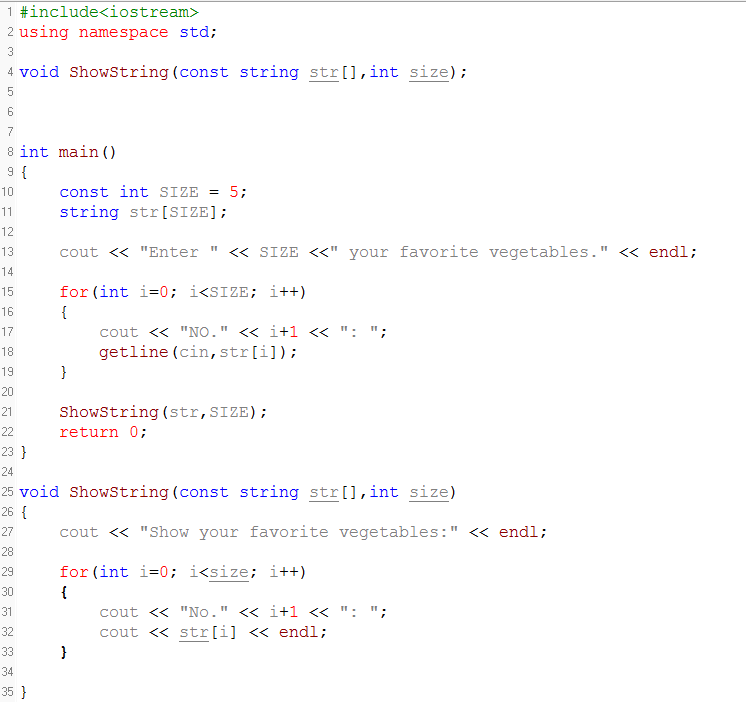


函数和结构

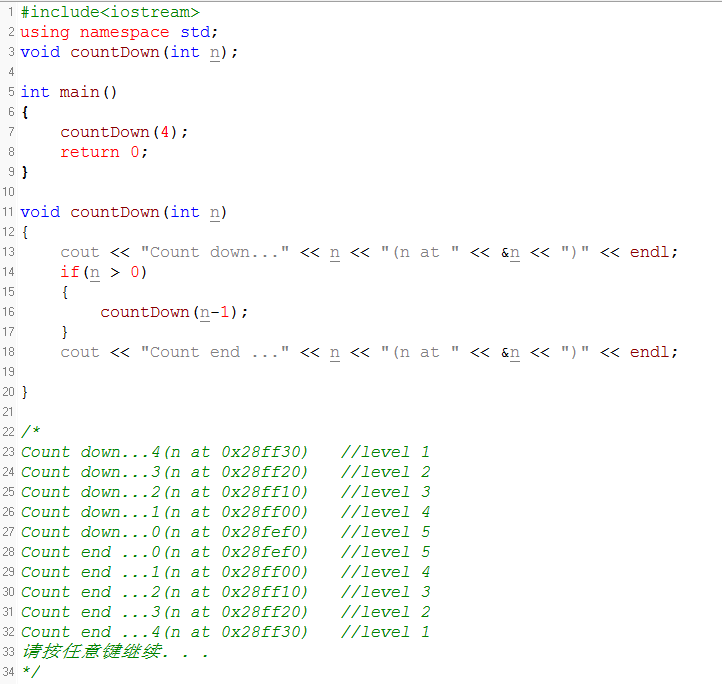




函数和string对象

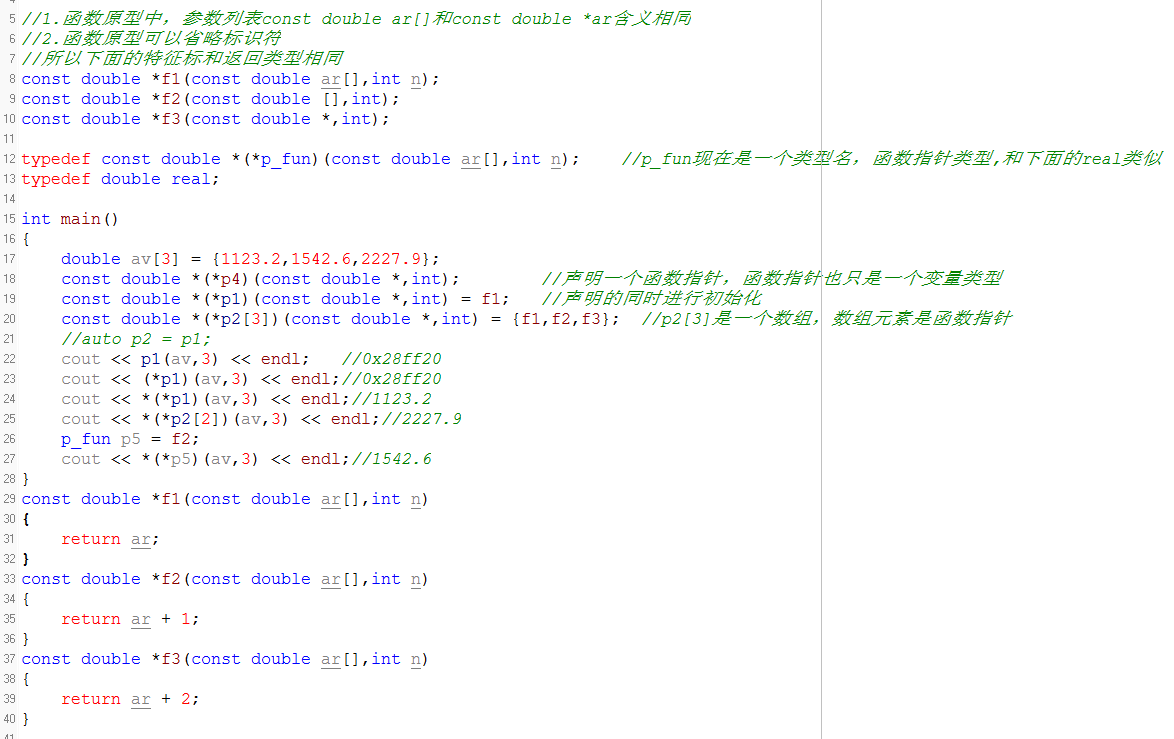


递归



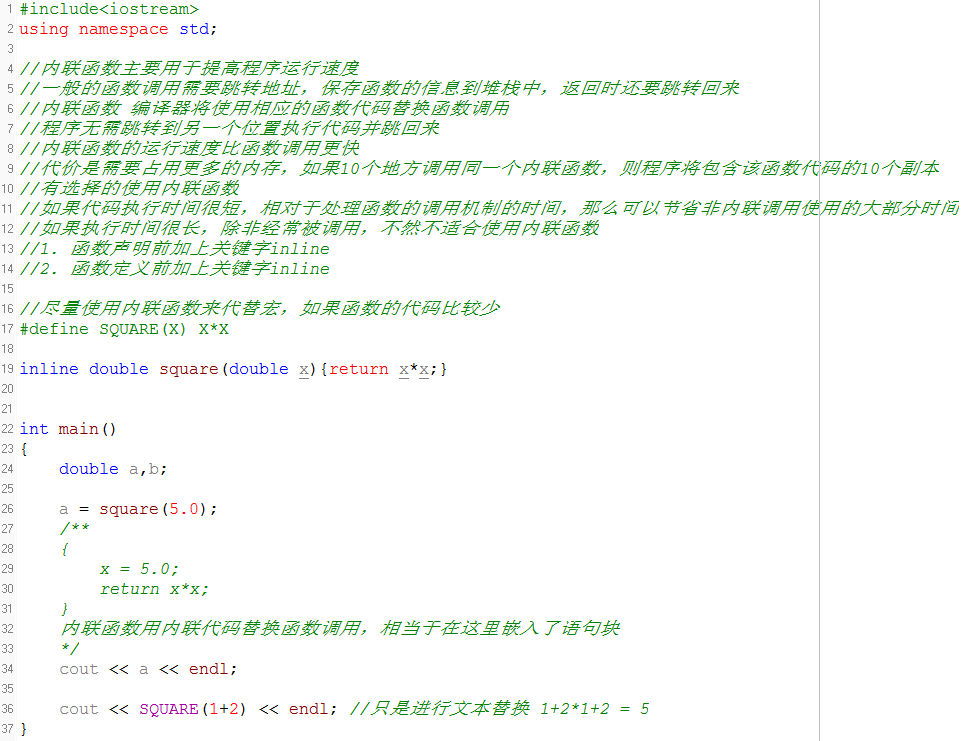
函数指针



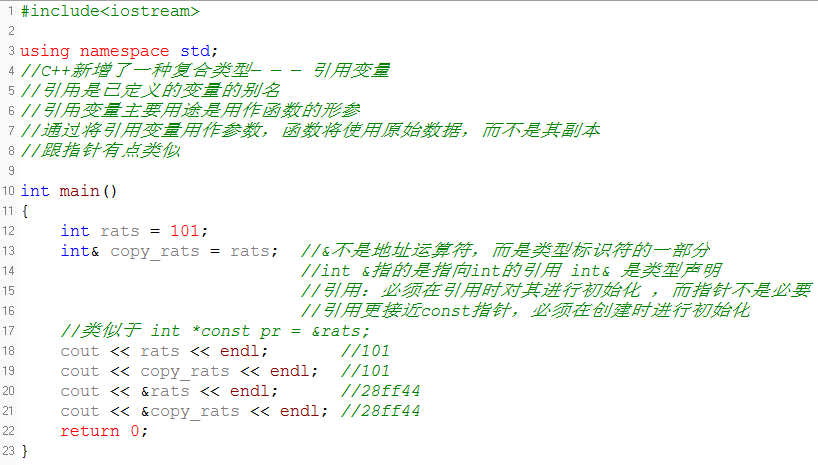


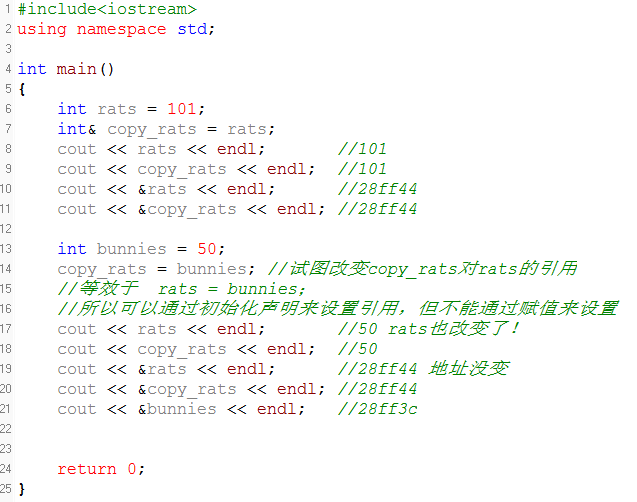
第七章 函数探幽

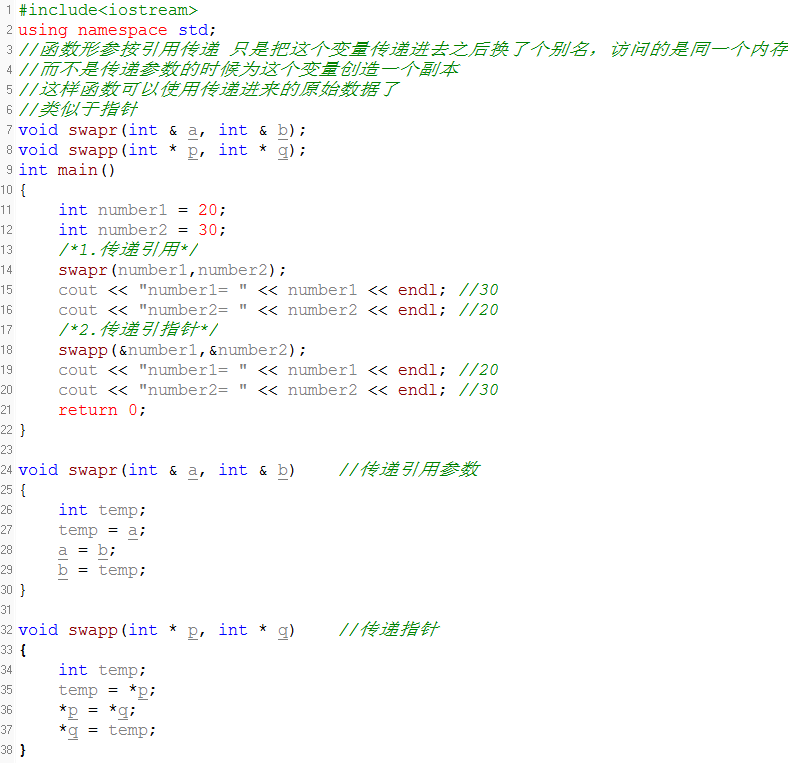
C++内联函数

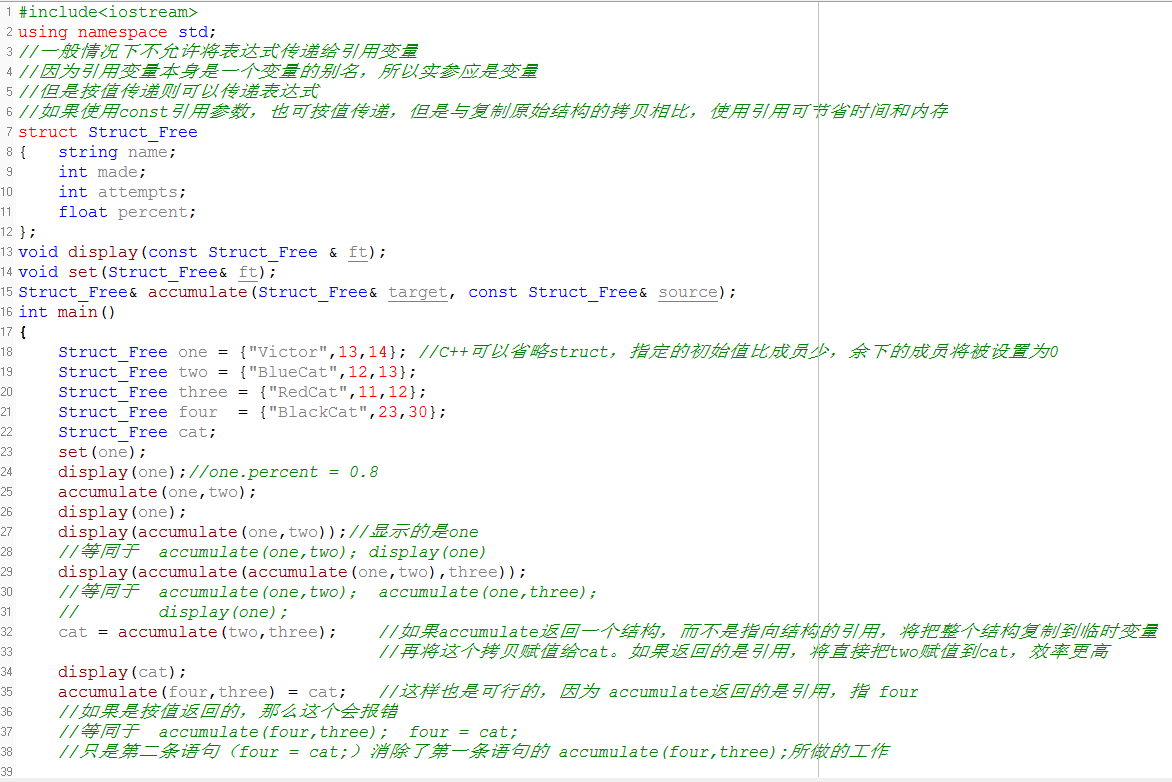


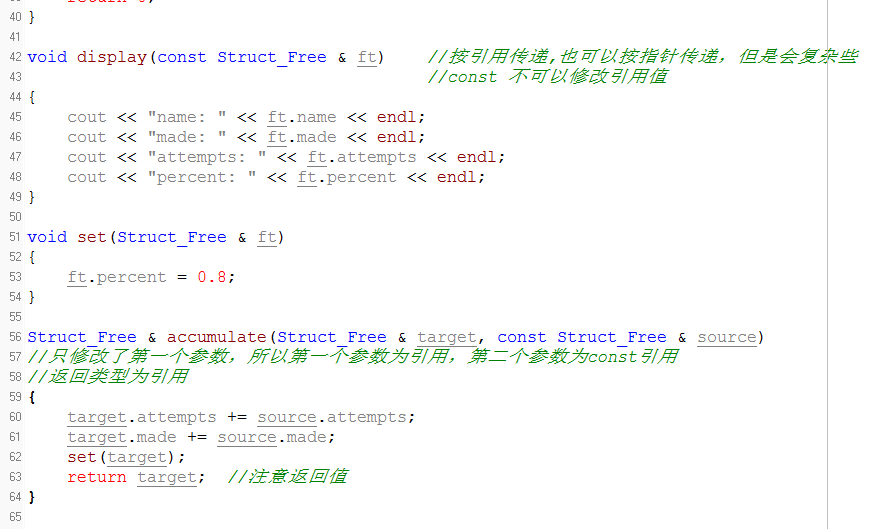
引用变量

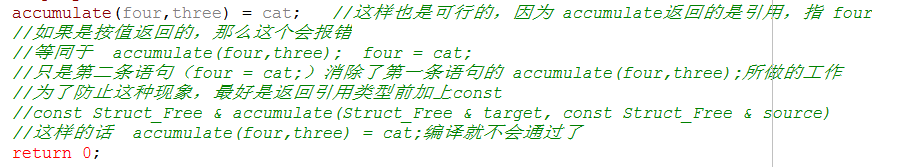












什么时候该使用引用参数，什么时候应使用指针，什么时候应使用按值传递？  
一、如果参数不作修改  
1.如果数据对象很小，如内置数据类型或小型结构，则按值传递  
2.如果数据类型是数组，则使用指针， 这是唯一的选择，并将指针声明为指向const的指针  
3.如果是较大的结构，则使用const指针或const引用，以提高程序效率，可以节省复制结构所需的时间和空间  
4.如果数据对象是类对象，则使用const引用，传递类对象的标准方式是按引用传递  
二、如果需要修改调用函数中的数据的函数  
1. 如果数据对象很小，如内置数据类型或小型结构，则使用指针  
2. 如果数据对象是数组，则只能使用指针  
3. 如果数据对象是结构，则使用引用或指针  
4. 如果数据对象是类对象，则使用引用

默认参数

#include<iostream>  
using namespace std;  
**const** int SIZE = 20;  
char \*left(**const** char \* str, int n = 1); //如果不传递n，默认是1  
//void fun(int k=1, int m=6, int n=8);  
//fun(3,5) = fun(3,5,8)  
//fun(3, , 5) INVALID   
  
int main()  
{  
 char Arr[SIZE];  
 int LeftCount;  
 cout <<"Enter the String: ";  
 cin.get(Arr,SIZE);  
 cout << "Enter the num ch you want: ";  
 cin >> LeftCount;  
 char \*p;  
 p = left(Arr,LeftCount);  
 cout << "The new String is "<< p << endl;  
 **delete** [] p;  
 **return** 0;  
}  
  
  
char \*left(**const** char \* str, int n)  
{  
 **if**(n < 0)  
 {  
 n = 0;  
 }  
 char \*p = **new** char[n+1];//分配动态数组   
 int i;  
 **for**(i=0; i<n&&str[i]; i++)

//for(init-expression;test-expression;update-exression)  
 //1.设置初始值 2.执行测试 3.执行循环操作 4.更新用于测试的值   
 {  
 p[i] = str[i];  
 }  
 **while**(i <= n)

//如果Arr本身比较小，而n比较大，那么浪费了存储空间，多余的空间都被加上了'\0'   
 {  
 p[i++] = '\0';  
 }  
 **return** p;  
}

优化一：

#include<iostream>  
using namespace std;  
**const** int SIZE = 20;  
char \*left(**const** char \* str, int n = 1); //如果不传递n，默认是1  
  
int main()  
{  
 char Arr[SIZE];  
 int LeftCount;  
 cout <<"Enter the String: ";  
 cin.get(Arr,SIZE);  
 cout << "Enter the num ch you want: ";  
 cin >> LeftCount;  
 char \*p;  
 p = left(Arr,LeftCount);  
 cout << "The new String is "<< p << endl;  
 **delete** [] p;  
 **return** 0;  
}  
  
  
char \*left(**const** char \* str, int n)  
{  
 int len = strlen(str);  
 n = (n < len) ? n : len; //避免确保n值过大的问题   
 char \*p = **new** char[n+1];  
 int i;  
 **for**(i=0; i<n&&str[i]; i++)  
 {  
 p[i] = str[i];  
 }  
 **while**(i <= n)  
 {  
 p[i++] = '\0';  
 }  
 **return** p;  
}

优化二：

#include<iostream>  
using namespace std;  
**const** int SIZE = 20;  
char \*left(**const** char \* str, int n = 1); //如果不传递n，默认是1  
  
int main()  
{  
 char Arr[SIZE];  
 int LeftCount;  
 cout <<"Enter the String: ";  
 cin.get(Arr,SIZE);  
 cout << "Enter the num ch you want: ";  
 cin >> LeftCount;  
 char \*p;  
 p = left(Arr,LeftCount);  
 cout << "The new String is "<< p << endl;  
 **delete** [] p;  
 **return** 0;  
}  
  
  
char \*left(**const** char \* str, int n)  
{  
 //int len = strlen(str); 导致运行效率降低   
 int m = 0;  
 **while**((str[m]!='\0') &&(m<n)) //while(m<n && str[m])  
 {  
 m++;  
 }  
 char \*p = **new** char[m+1];  
 int i;  
 **for**(i=0; i<m&&str[i]; i++)  
 {  
 p[i] = str[i];  
 }  
 **while**(i <= m)  
 {  
 p[i++] = '\0';  
 }  
 **return** p;  
}

函数重载

#include<iostream>  
using namespace std;  
//函数重载主要用于基本上执行相同的任务，但使用不同形式的数据时才应采用函数重载   
//函数重载的关键是函数的参数列表——函数特征标（使得可以对函数进行重载，而不是返回类型）   
//1.函数名称可以相同，但是特征标必须不同  
//2.将类型引用和类型本身视为同一个特征标  
//double cube(double x) 和 double cube(double & x)  
//3.并不区分const和非const变量   
//因为将非const转化为const是合法的，反之则是非法的  
**const** int SIZE = 40;  
unsigned long left(unsigned long num, unsigned ct);  
char\* left(**const** char\* str, int n=1); //函数重载left   
  
int main()  
{  
 char Str[SIZE];  
 unsigned long Num;  
 unsigned LeftCount;  
 char \*p;  
 cout << "Enter the string: ";  
 cin.get(Str,SIZE);  
 cout << "Enter the num of left start string for you want: ";  
 cin >> LeftCount;  
 p = left(Str, LeftCount);  
 cout << "The left String is: " << p << endl;  
  
 cout << "Enter the num: ";  
 cin >> Num;  
 cout << "Enter the num of left start Num for you want: ";  
 cin >> LeftCount;  
 cout << "The left Num is: " << left(Num,LeftCount) << endl;  
}  
  
/\*取整数num左边ct个数字组成的整数\*/  
unsigned long left(unsigned long num, unsigned ct) //unsigned = unsigned int  
{  
 unsigned digitCount = 1;  
 unsigned long numCopy = num;  
 **if**(num == 0 || ct == 0)  
 {  
 **return** 0;  
 }  
  
 **while**(num/=10)  
 {  
 digitCount++; //共有多少个数字   
 }  
 **if**(digitCount > ct) //如果需要左边的位数小于整数的总位数   
 {  
 ct = digitCount - ct; //总共的数字 - 需要的左边的位数 = 需要去掉的位数   
 **while**(ct--) //ct-- 先使用ct的当前计算表达式，然后将ct的值-1 --ct   
 //先将ct的值-1，然后使用新的值来计算表达式   
 {  
 numCopy = numCopy/10;  
 }  
 **return** numCopy;  
 }  
 **else** //如果需要的左边的位数大于整数总位数，直接输出   
 {  
 **return** numCopy;  
 }  
}  
  
/\*取字符串str的左边n个字符\*/  
char\* left(**const** char\* str, int n)  
{  
 //int len = strlen(str); 导致运行速度降低  
 int m = 0;  
 **while**((m < n) && (str[m]!= '\0')) //while((m < n) && str[m]) 这样也行  
 {  
 m++;  
 }  
 char \*p = **new** char[m+1];  
 int i;  
 **for**(i=0; i<m&&str[i];i++)  
 {  
 p[i] = str[i];  
 }  
 **while**(i<=m)  
 {  
 p[i++] = '\0';  
 }  
 **return** p;  
}

函数模板

#include<iostream>  
using namespace std;  
//函数模板：需要多个将同一种算法用于不同类型的函数，请使用模板  
//需要注意的是最终仍将是两个独立的函数定义,并不能缩短可执行程序   
//使用模板只是为了缩减代码量，让函数定义更简单可靠

//模板只是一个用于生成函数定义的方案，模板并非函数定义  
template <typename AnyType> //typename向后兼容，函数模板原型,类型命名为AnyType  
**void** Swap(AnyType &a, AnyType &b);  
//template <class T1> //class比较旧   
//void Swap(T1 &a, T1 &b);  
int main()  
{  
 int i = 20;  
 int j = 30;  
 Swap(i,j); //模板函数中的 AnyType被替换成了int   
 cout << i << endl; //30  
 cout << j << endl; //20  
  
 char a = 'a';  
 char b = 'b';  
 Swap(a,b); //模板函数中的 AnyType被替换成了char  
 cout << a << endl; //b  
 cout << b << endl; //a  
}  
  
template <typename AnyType> //函数模板定义  
**void** Swap(AnyType &a, AnyType &b) //这里的参数是引用类型，参数数据类型是根据传入什么类型就是什么类型，这就是模板   
{  
 AnyType temp;  
 temp = a;  
 a = b;  
 b = temp;  
}

模板重载：

#include<iostream>  
using namespace std;  
//模板重载   
template <typename AnyType>  
**void** Swap(AnyType &a, AnyType &b);  
template <typename AnyType>  
**void** Swap(AnyType \*a, AnyType \*b, int n);  
  
**const** int SIZE = 5;  
int main()  
{  
 int i = 20;  
 int j = 30;  
 Swap(i,j); //模板函数中的 AnyType被替换成了int   
 cout << i << endl; //30  
 cout << j << endl; //20  
  
 char a[SIZE] = "ABCD";  
 char b[SIZE] = "EFGH";  
 Swap(a,b,SIZE); //模板函数中的 AnyType被替换成了char  
 cout << a << endl; //EFGH  
 cout << b << endl; //ABCD  
}  
  
  
  
template <typename AnyType>  
**void** Swap(AnyType &a, AnyType &b) //参数为引用类型  
{  
 AnyType temp;  
 temp = a;  
 a = b;  
 b = temp;  
}  
  
template <typename AnyType>  
**void** Swap(AnyType a[], AnyType b[], int n) //参数为指针  
{  
 AnyType Temp;  
 int i;  
 **for**(i=0; i<n; i++)  
 {  
 Temp = b[i];  
 b[i] = a[i];  
 a[i] = Temp;  
 }  
}

第八章 内存模式和名称空间

自动存储持续性

#include<iostream>  
using namespace std;  
**void** oil(int x);  
  
int main()  
{  
 int Var = 5;  
 cout << "Main\_Var=" << Var << ",Address=" << &Var << endl;  
 {  
 int Var = 100; //隐藏了值为5的Var   
 cout << "Block\_Main\_Var=" << Var << ",Address=" << &Var << endl;//100  
 }  
 cout << "Main\_Var=" << Var << ",Address=" << &Var << endl; //5  
 oil(Var);  
}  
  
**void** oil(int x)  
{  
 int Var = 110;  
 cout << "Oil\_Var=" << Var << ",Address=" << &Var << endl; //110   
}

静态存储持续性

//静态变量是分配固定的内存块来存储的。整个程序执行期间都在。

//如果没有显示地初始化静态变量，编译器把它设置为0。

//1.静态，无链接性（在代码块中使用，使用static，当前函数或代码块可用）

//2.静态，外部连接性（不在任何函数内，不使用static，多文件可用）

//3.静态，内部链接性（不在任何函数内，使用static，当前文件可用）

1.定义声明，简称定义，给变量分配存储空间

2.引用声明，简称声明，不给变量分配存储空间

引用声明使用关键字extern，且不进行初始化，否则声明变成了定义，导致分配内存空间

作用域解析运算符:: 放在变量名前面该