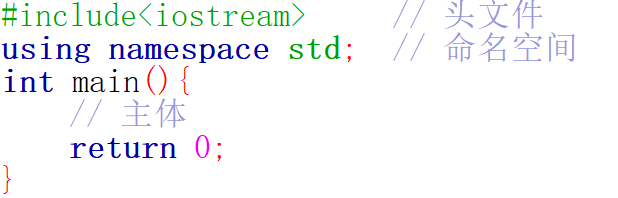
**基本框架：**



**常用的头文件：**

头文件的作用：包含了头文件就可以使用其提供的功能，例如想使用求根号的运算，就需要包含<math.h>

(先简要介绍下，后面详细介绍具体的使用)

# include<iostream> 有了它就可以用基本的输入输出，例如 cin,cout

# include<vector> 使用vector容器 常用

# include<queue> 使用队列 常用

# include<algorithm> 使用一些算法，常用的就是排序sort()

# include<math.h> 使用一些数学函数，如求根号，求x的y次方等

# include<map> 现在不常用了，常用unordered\_map替代

# include<set> 常用来做去重和自动排序

# include<unordered\_map> 哈希表 leetcode 刷题很常用

# include<deque> 双端队列

# include<fstream> 读写文件使用，平常刷题不用，但是如果需要读txt, csv,excel文件需要

# include<string> 处理字符串很方便， 常用

# include<cstdio> 如果想用C语言的输入输出，scanf printf

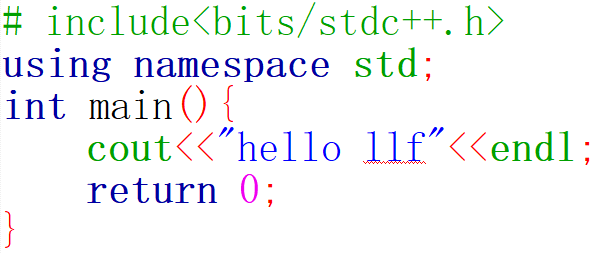
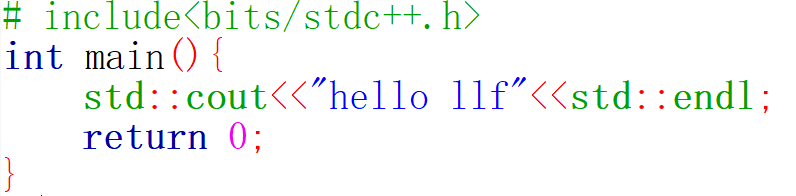
**万能头：**

# include<bits/stdc++.h>

可以只包含这一个，就包含上述提及或者没有提及的所有头文件了，优点在于不用敲那么多头文件，缺点就是编译速度慢，有的比赛可能不支持。

**命名空间：**

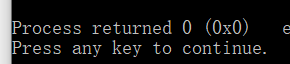
using namespace std; 但是如果不包含这行代码：

cout 和 endl 就前面需要加上std 这个作用域，很麻烦，因此就记住加上这段代码就可以了，不用深入了解命名空间具体指什么。

**return 0 ：**

如果程序正常运行，没有语法错误，那么可以从控制台中看到return 0。



其他的数字1或者-1就表明程序有问题。

**常用的数据类型：**

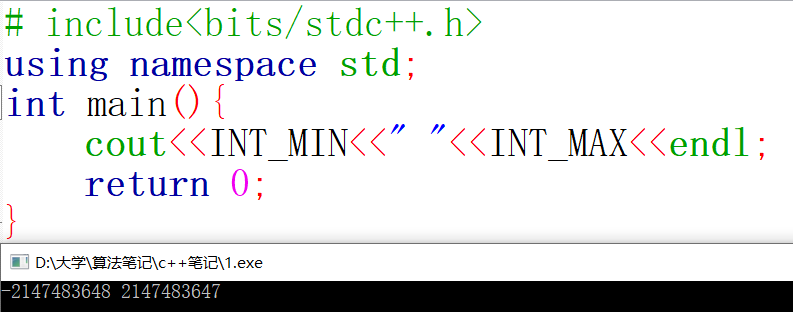
int, double, char, string, long long, bool

float, unsigned int, unsigned long long 较少使用

**整数：**

常用int，如果数字很大，考虑用long long, 否则会溢出，导致运算错误。

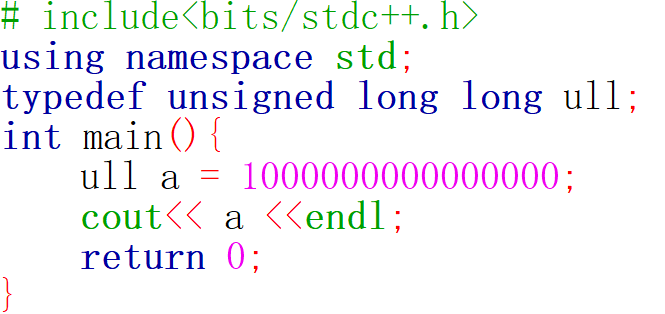
具体数据在什么范围内用int？

大致就是21亿，通常就是以内用int就可以，超过用long long

如果有时候数据还是很大，用long long 也不够，可以考虑用 unsigned long long ,其中unsigned 表示无符号，也就是不能表示负数，只能表示正数，正数表示范围\*2，另外为了简化书写，常用typedef unsigned long long ull; 意思是用ull表示unsigned long long这一长串：

具体例子：



为了更好的理解这一问题，再用unsigned int 举例：

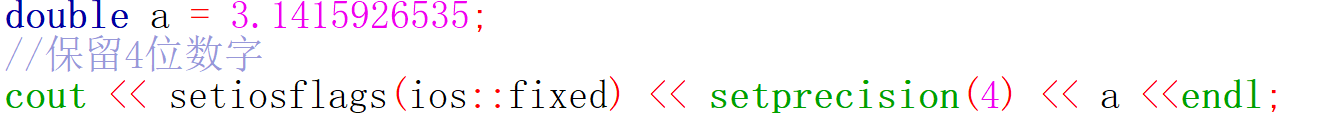
int 如今基本用32位表示数字，但是其中首位是表示正负，后面15位才表示数字，因此表示范围, 因为多表示了0，所以正数少一个。但如果用unsigned int，意思就是最高位不再表示正负，直接表示,

注意：c++和c语言中使用某个变量之前必须要定义（或者说声明），否则会报错，但是c++不同于c语言变量必须在一开始定义，c++定义变量相对比较灵活，随用随定义。

**小数：**

涉及到小数的运算，直接用double

这里面涉及到最后输出显示小数点位数的问题



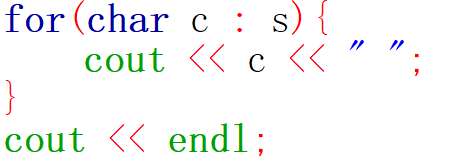
**字符型和字符串型：**

字符用‘’，字符串用“”，难点在于两者的混用

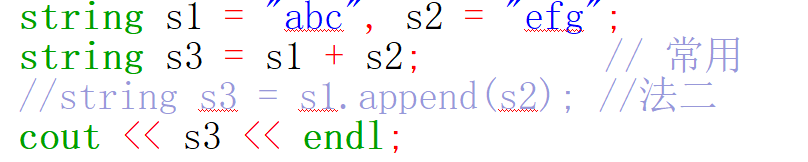
字符串string本质是一个字符数组，可以像访问数组一样访问字符串。

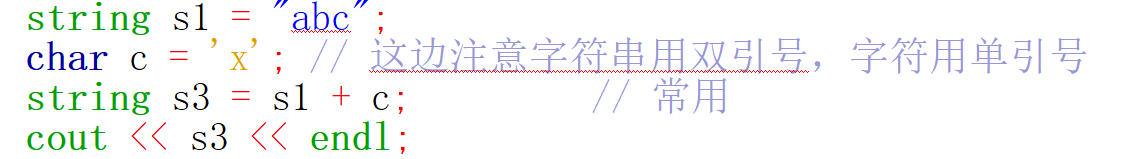
C++11特性，简化的遍历版本：

上面传统的用于理解，平常用这个版本，更简洁

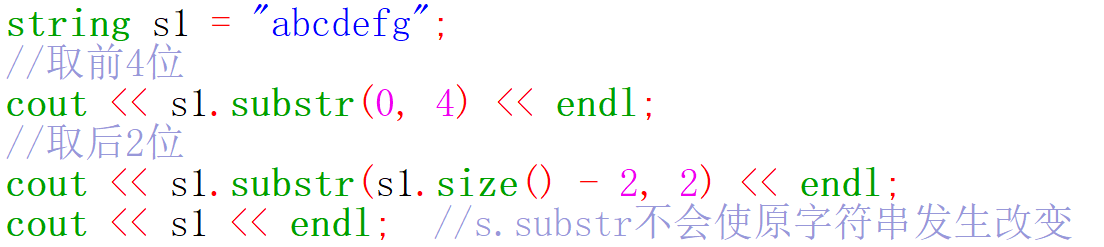
字符串后追加字符串：

结果：abcefg

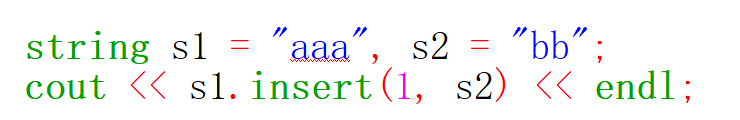
字符串后追加字符： （也可以直接使用+）



字符串截取：s.substr(k,len) 从s下标k开始，截取长度len, 如果len> s.size(), 截取到最后。

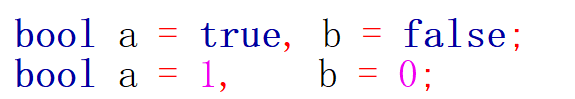
 

字符串插入： s1.insert(k,s2) , 在s1的第k位插入字符串s2

其他的操作遇到了百度即可，通常就是用上面几种就够了。

**bool 类型：**

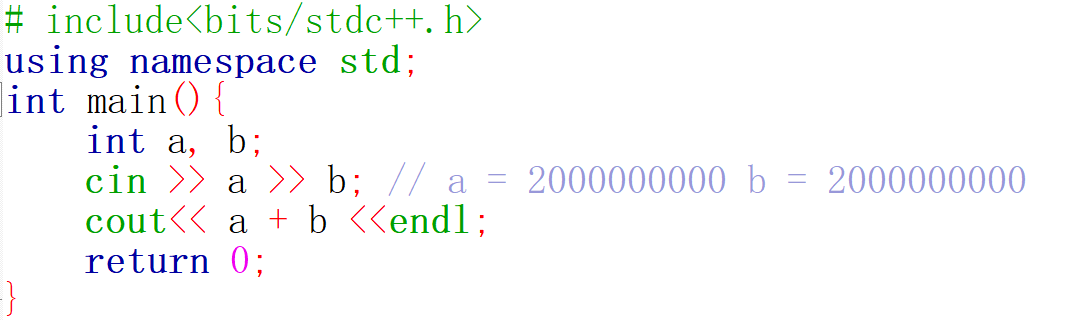


两种方法等效，所以通常就是用1表示对，0表示不对，不同特意去写True和False

**基本的输入输出： cin, cout**

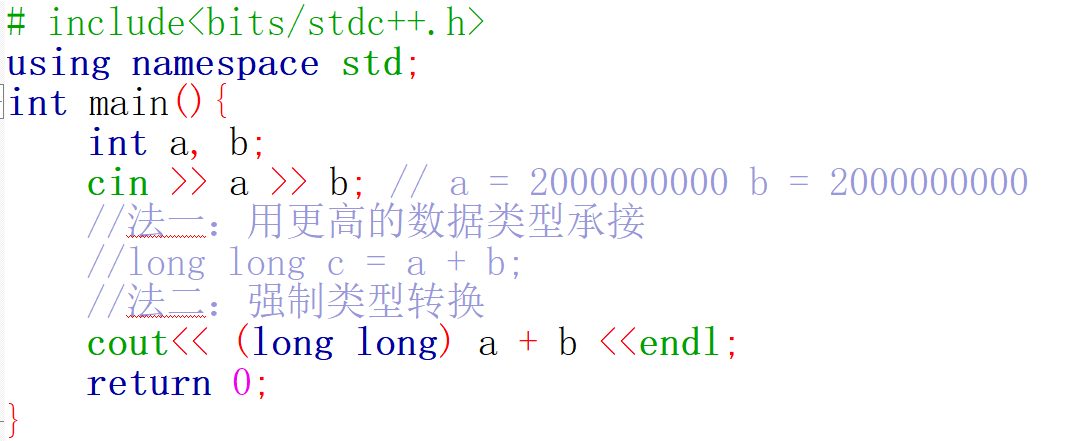
1. 输入两个数，然后求出两数之和：（数据量较大）（直接a+b感觉你肯定会了）

a ,b 都是, 单独都在int 范围内，但是加起来超过了int



 结果溢出。

解决方法：





**常用的运算：**

**（1）自增自减： a++,++a, b--,--b**

区分a++和++a: 两者都是使a增加一，但是a++使得（a++）这个整体数值不变，++a使得（++a）这个整体数值加一。





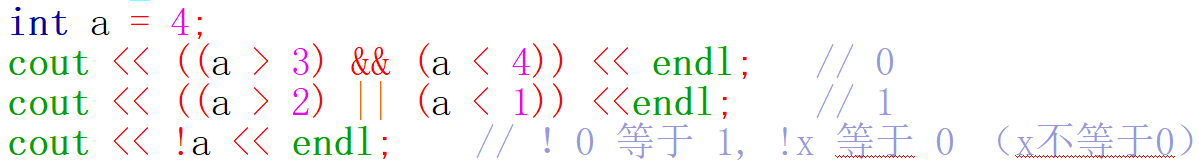
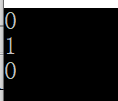
**（2）取余数和取整数运算： （leetcode刷题常用）**

取整 a/b 取余 a%b

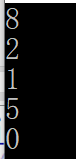


即 11 = 3 \* 3 + 2

**（3）逻辑运算符: 与 &&， 或 || ， 非 ！**

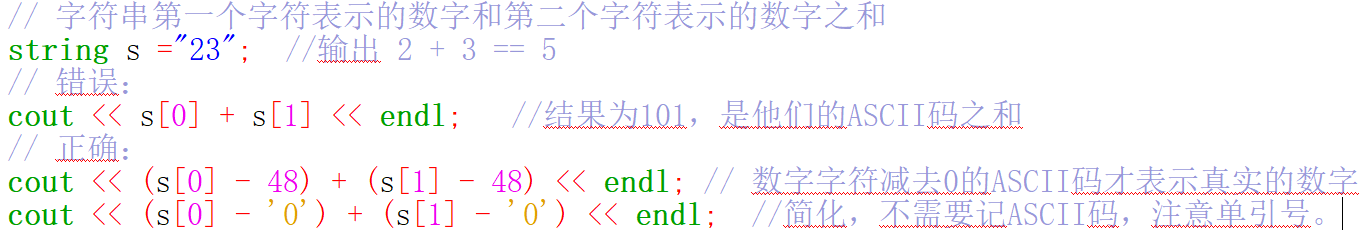
**（4）位运算：**

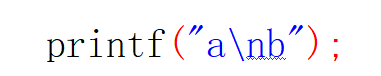
**一些需要记得点：**

ASCII码：（字符转数字需要用）

0：48 A：65 a：97

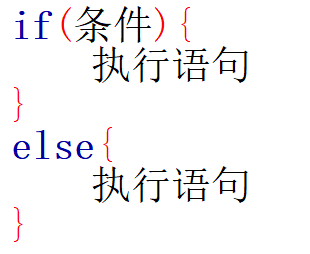
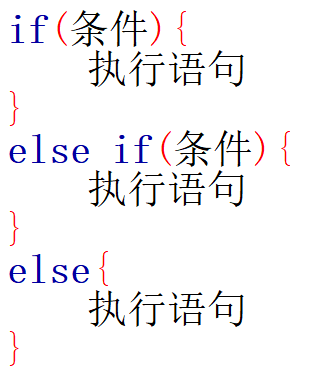
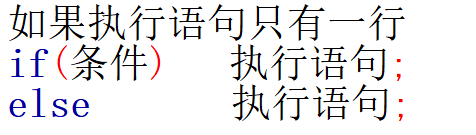
 

换行的转义字符： ‘\n‘ 记得是右斜杠

**基本语句：条件，循环等**

**条件语句：**

**switch语句**

switch(表达式)

{

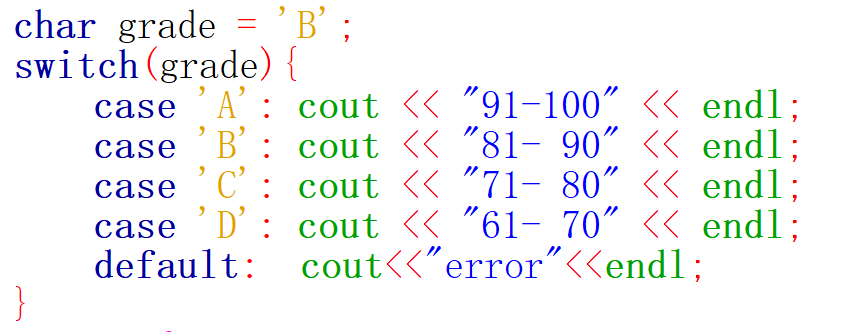
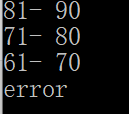
case 常量表达式1： 语句

case 常量表达式2： 语句

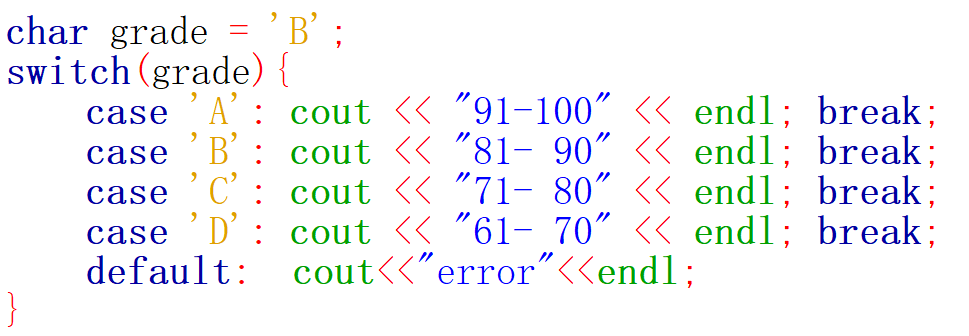
case 常量表达式3： 语句

}

当switch表达式的值与某个case子句中的常量表达式的值想匹配的时候，就从该case子句往下一直执行。如下面的例子，输出并不是仅仅输出81-90，而是从case B开始一直执行。

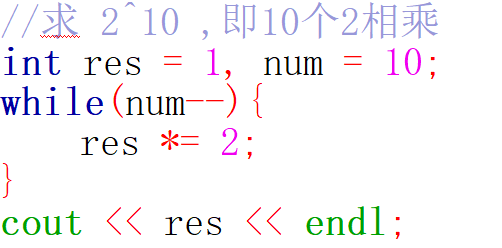
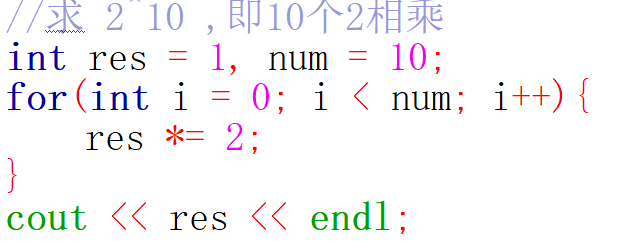
若要想达到理想的效果，应该在每个case语句后加break

另外经过测试，switch(表达式)中的表达式只能是数值类型的数据(数字或者字符(字符转化为其ASCII码,因此也是个数字))，不能是字符串等其他类型。

**循环语句：** while 和 for （感觉不用说具体的语法，直接上例子）

例如：求 2的10次方

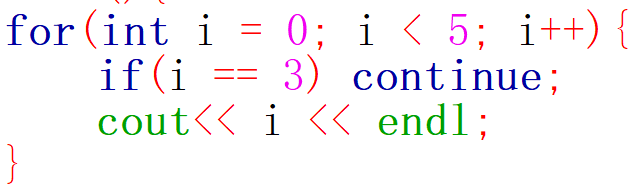
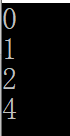
  

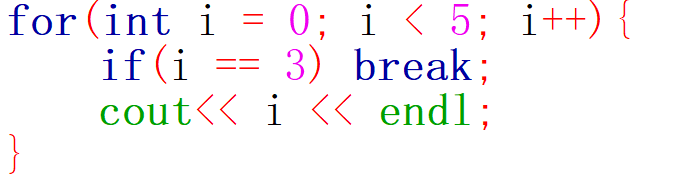
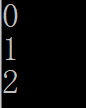
循环语句中注意continue 和 break 的区别：

continue指的是仅跳过本次循环，即本次循环continue后面的语句都不再执行，直接跳到下一次循环。

break是跳出本层循环，即其所属的for或者while语句均不在执行，注意只能跳出一层。

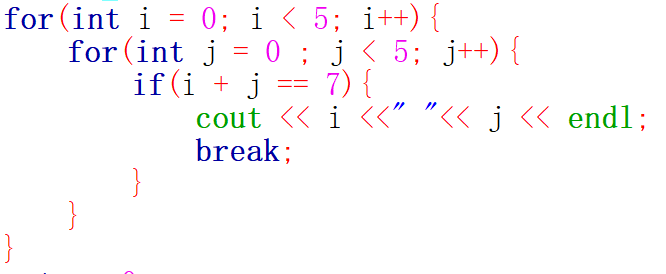
例如：

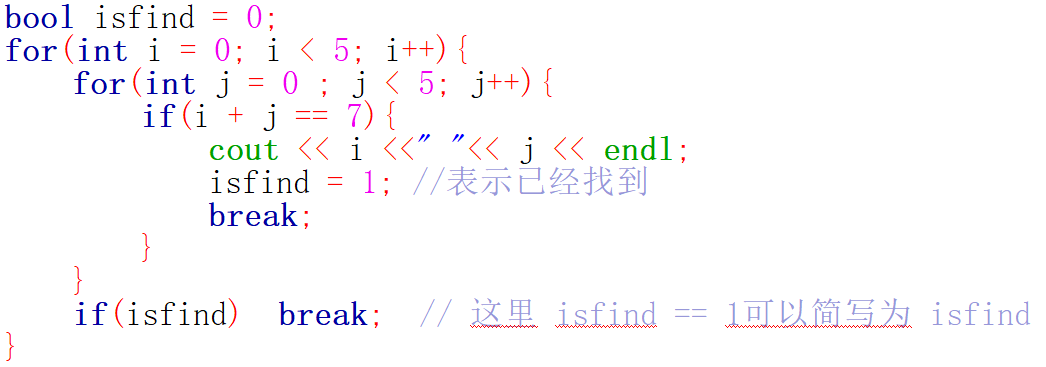
 

注意break只能跳出一层循环，这个地方容易忽略。

例如：想找到5以内的两个数相加等于7，找到一个即可。如果直接里层break，仅仅是跳出了for(j=0;j<5;j++),而有关i的循环仍会执行。

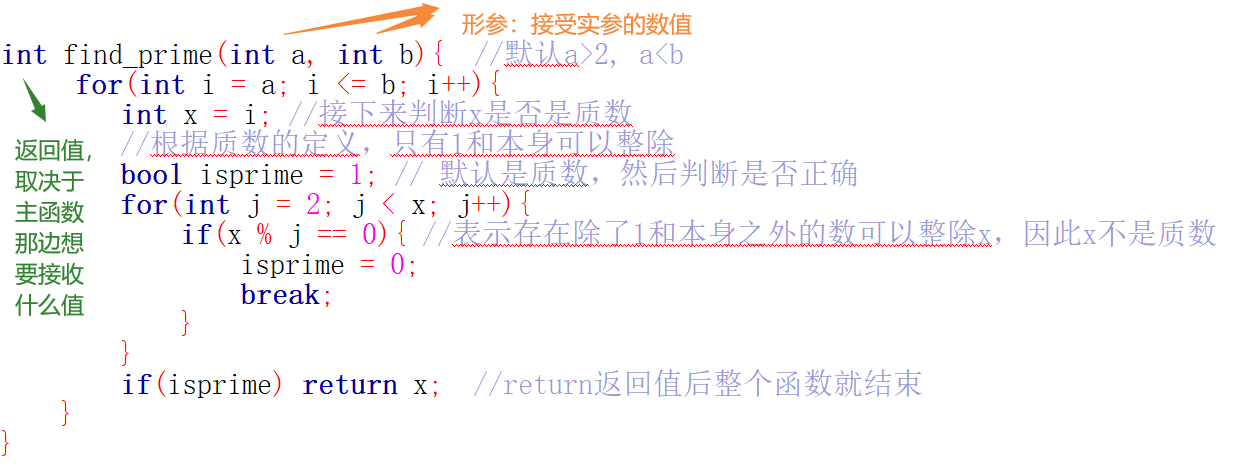
常用解决方法：（在不改变for循环的写法的情况下）

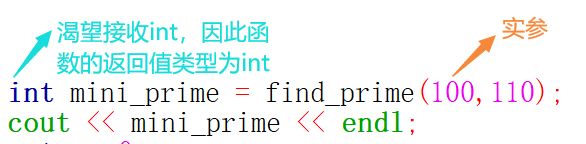
 

**函数：**

使用函数的作用：大致就是为了将某个功能封装为一个函数，从而有助于频繁调用和精简主函数。

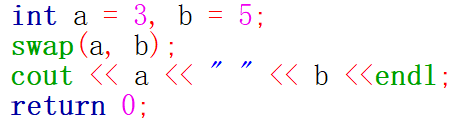
例如：编写一个函数find\_prime(a,b)，找到区间[a,b]内最小的质数



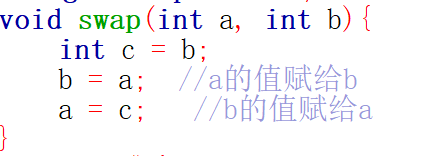
 

一个经典的问题：编写一个函数，实现交换两个数，

swap(a,b),交换a,b的值。

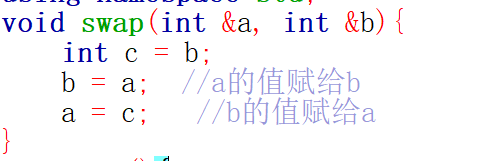
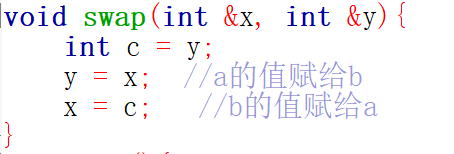


错误做法：

结果表明并没有实现交换，原因在于形参的a,b和实参的a,b不是一个东西，也就是它们并不指向同一内存，形参a,b的作用域仅仅是定义的swap函数，因此当函数执行完后，它们的内存空间就释放，虽然函数内部实现了形参a,b的交换，但是并没有返还给实参，因此实参的数值并未发生变化。

正确做法: 用引用 & 只需要小小的一点改动

就是直接在形参a,b前加上&这个符号，这个东西类似指针。 int &x = y, 表示给y起一个别名x，它们是同一个东西，仅仅是名字不同，通过不同的名字均可以访问到这同一个变量的内容。左图就表示此时实参的a,b和形参的a,b指向同一内存，因此，形参的交换就意味着实参的交换。右图刻意把a换成了x，b换成了y，表示这仅仅是一个别名，别名可以与原名相同，即都叫a,也可以取不同的名字，但是都表示一块内存，因为上面两个写法等效。都是对的

结果确实实现了形参和实参的交换。

通常函数只有一个返回值(int, double)或者没有返回值(void), 如果有多个返回值，怎么办？

第一种方法就是使用上述的传引用&，形参和实参表示的是同一个东西，形参修改了，实参也就跟着修改了，但是用这种方法一定要小心避免无意的修改。

第二种方法就是用数组或者vector或者自定义的类型返回多个值。

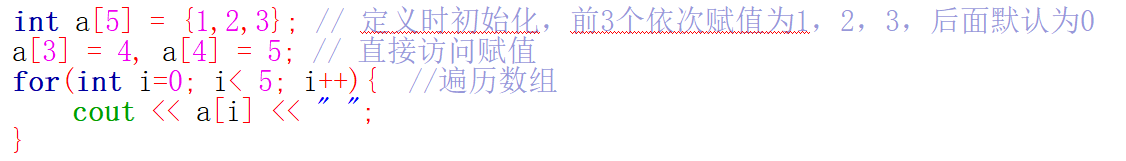
**数组：**

（后面有了stl中的vector后，数组就不经常用了。）

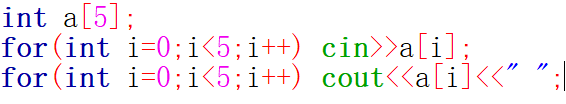
**一维数组：**

int a[5];

初始化和遍历方式：（遍历的意思就是将数组从头到尾访问一遍，每个地方都走一遍）

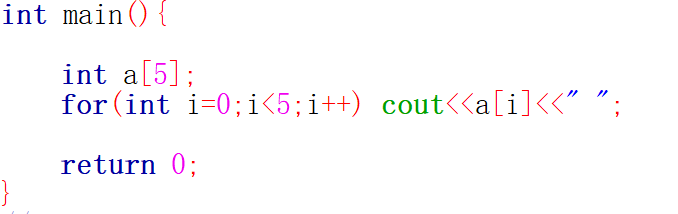
 

也可以通过输入初始化，多输入的无效。例如多输入了6，但是没影响，程序只读前5个。

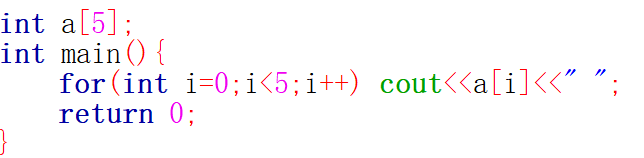


小细节：

如果数组是在函数中定义的，就是局部变量，那么其数值是随机的，并不是想当然的0

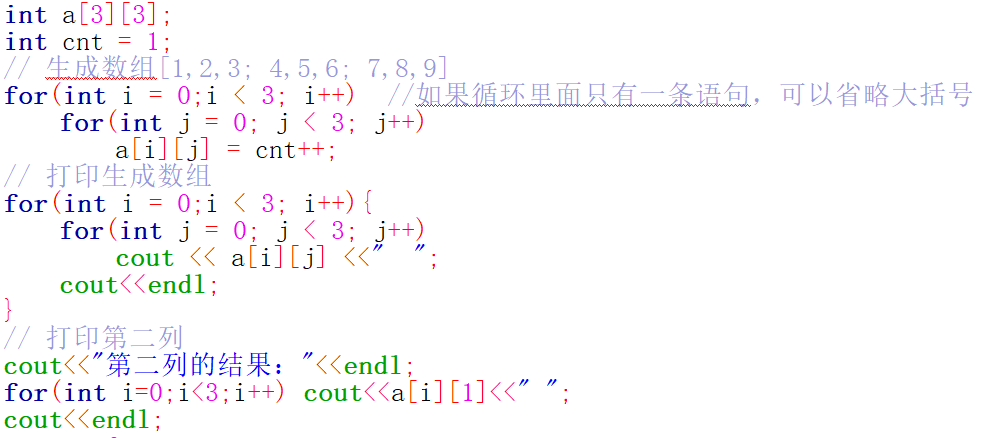
 

但如果是定义在函数外的全局变量，那么默认初始化为0

**二维数组：**一般就是两重循环处理

例如：生成一个二维数组[1,2,3; 4,5,6; 7,8,9] (;表示换行)，然后打印第二列。

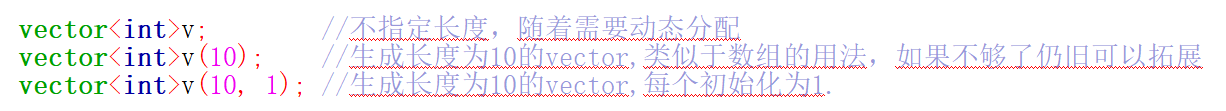


指针：不用看，基本不用

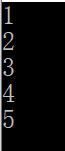
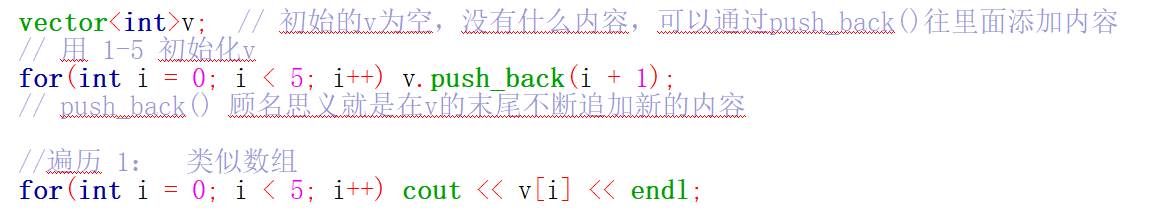
STL

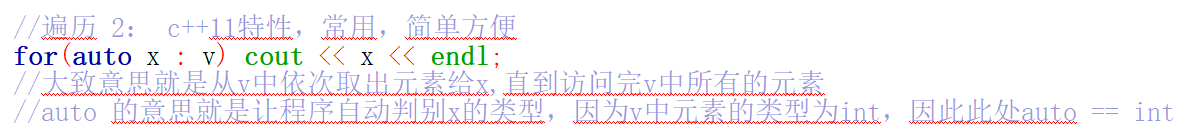
一、vector: 向量，不定长数组（重要，基本数组都被vector取代了）

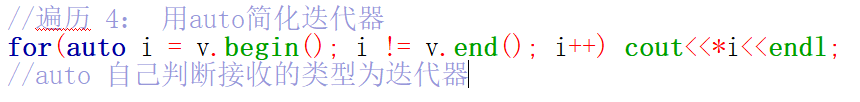
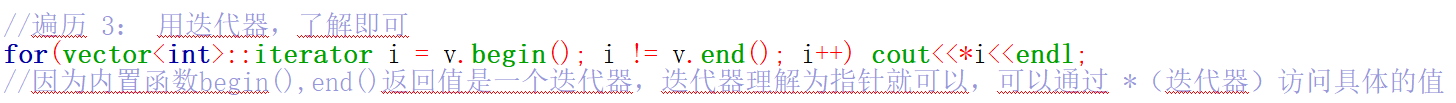
包含头文件： # include<vector>

初始化：  


遍历vector:

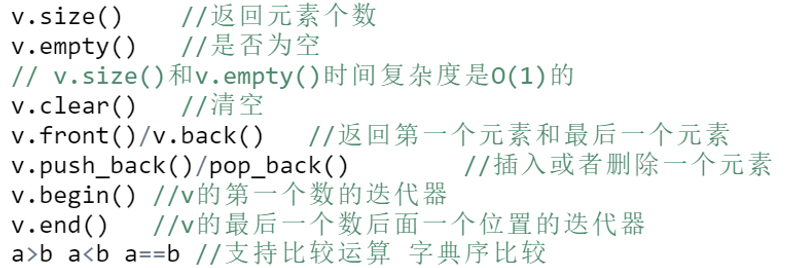




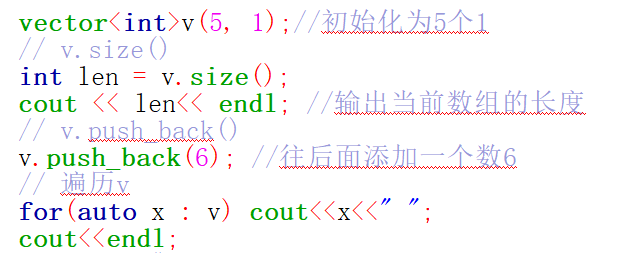


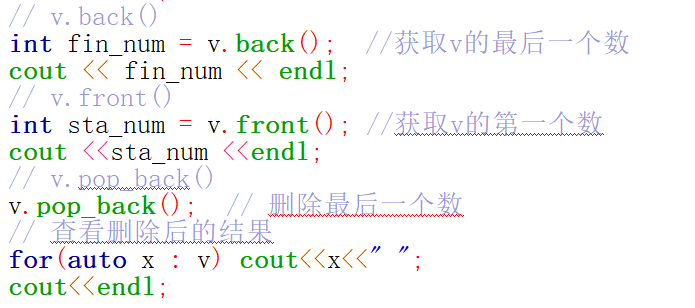
四种遍历方式的结果都是1，2，3，4，5

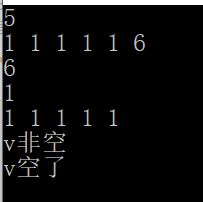
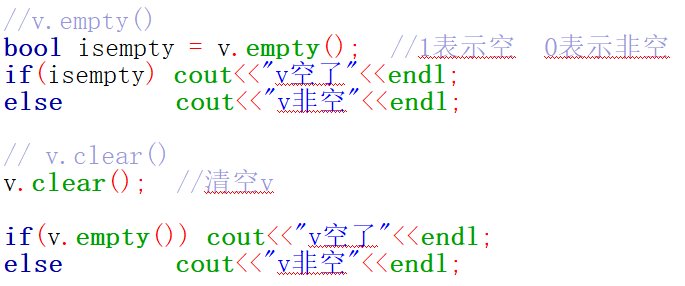
其他常用的操作：



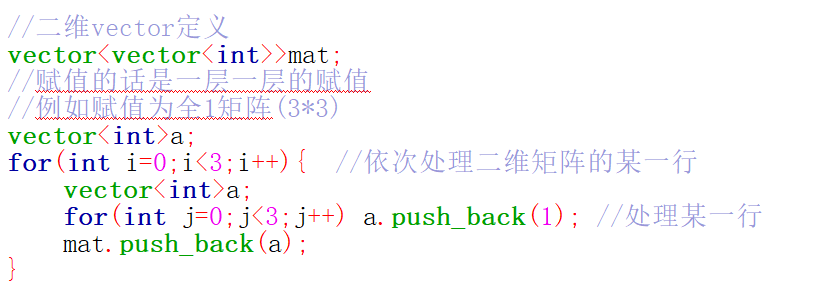
具体例子：



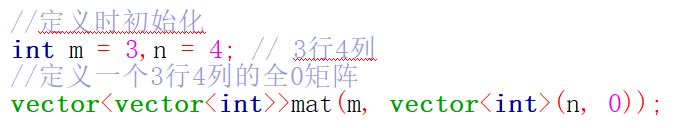




二维vector



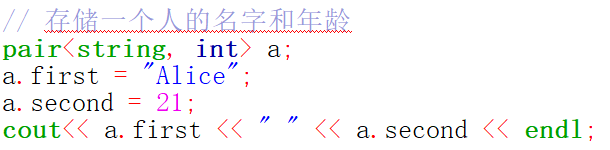
leetcode常用定义方法

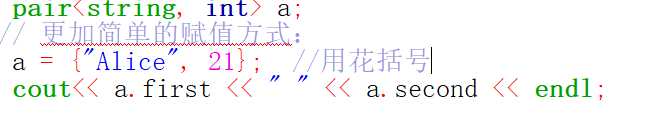


**二、pair**

需要包含的头文件：#include<utility> （有的编译器不包含也没问题）

pair 用来存储成对出现的东西。

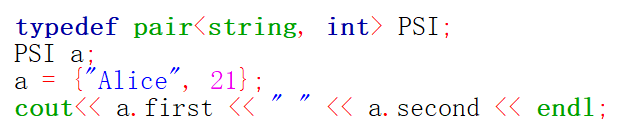




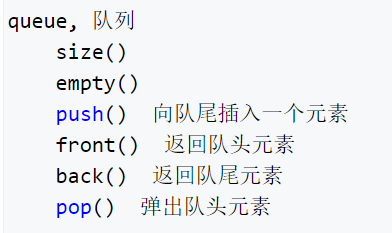
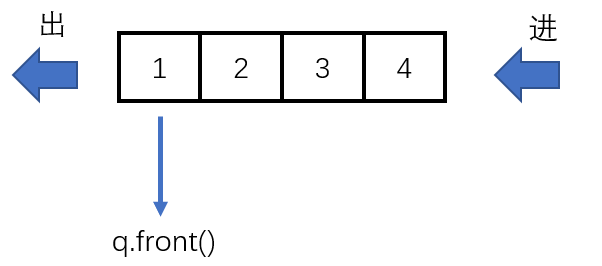
如果需要处理很多人，结合vector

vector<pair<string,int>>v

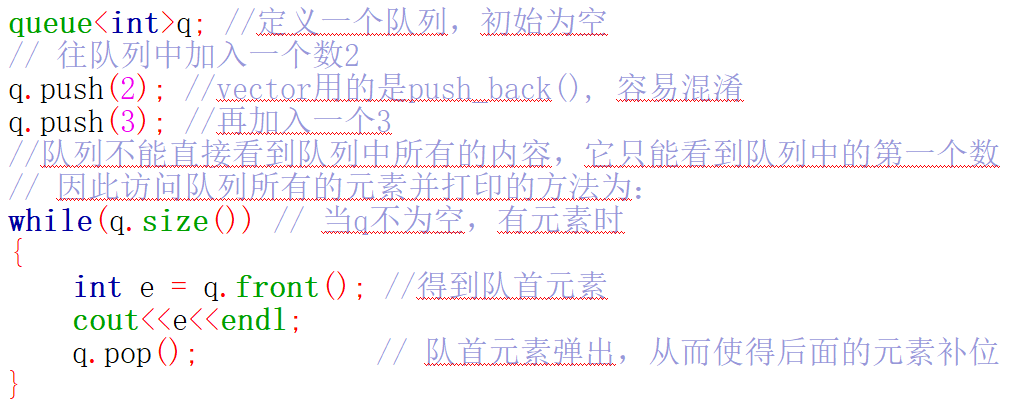
考虑到pair一般需要很长，可以用typedef简化：



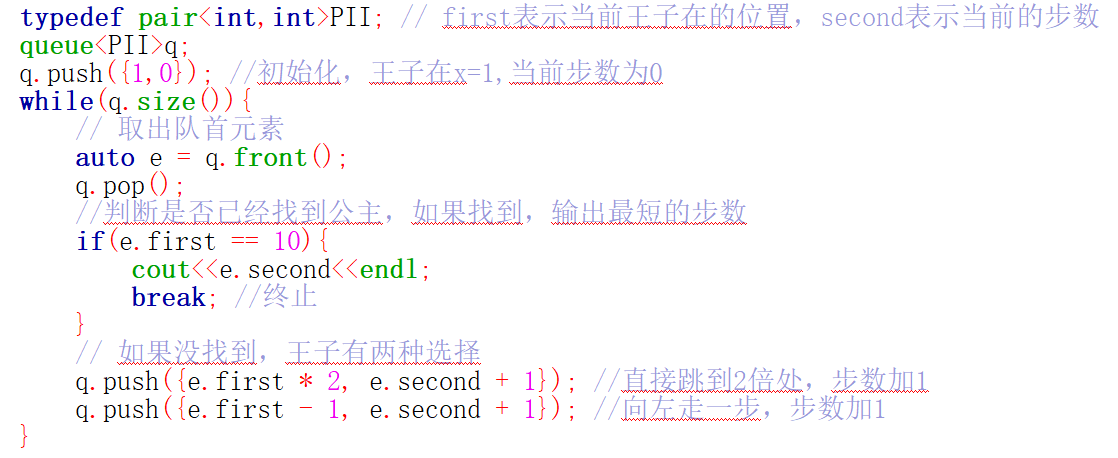
**三、队列：**

需要包含头文件： # include<queue>



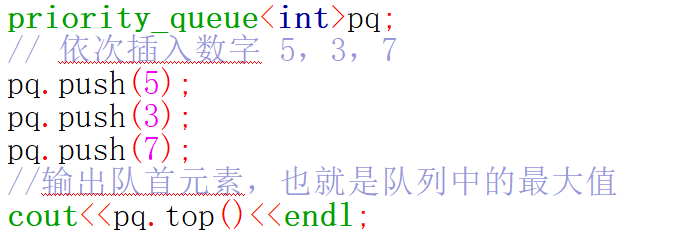
一个小例子：假设考虑一维空间，公主在x = 10，王子在 x = 1, 王子每次有两种选择， 直接跳到当前位置的两倍处（例如当前在x=4, 可以直接跳到8），或者向左移动一步，问王子最短多少步可以找到他的公主。（queue + pair）



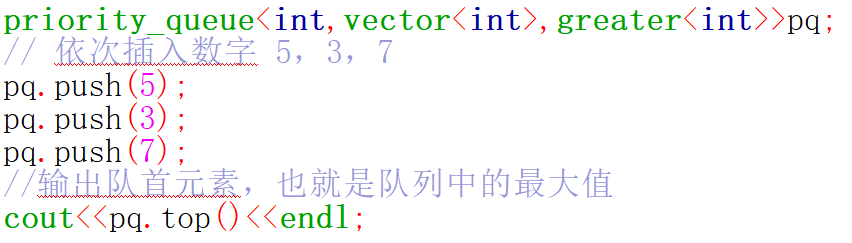
四、优先队列

需要包含头文件： # include<queue>

你尽管往队列中添加元素，队列自动将当前“最大”的元素放在队首，因为队列只能访问队首元素，因此每次访问的都是队列中的“最大”值，此处，“最大”可以自己定义什么是大，自定义排序规则。

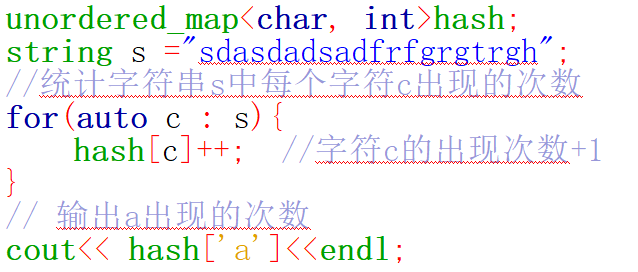
如果想自定义排序规则，定义小的数在前面：

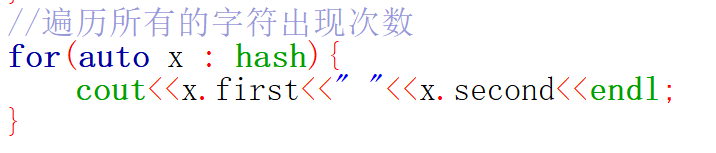
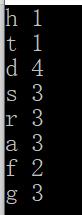
 

五、哈希表 unordered\_map

需要包含头文件： # include<unordered\_map>

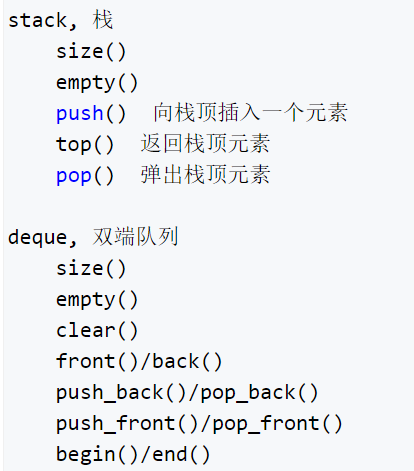
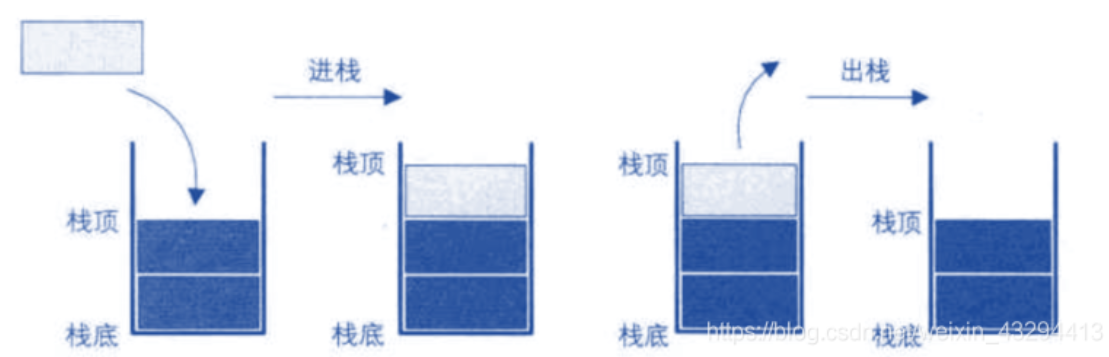
功能：给定一串字符，问每个字符出现的次数

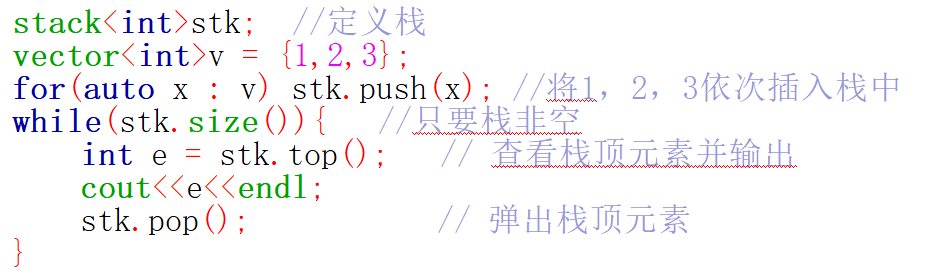
结果：3

**六、栈： stack**

需要包含头文件： # include<stack> 特点：后进先出

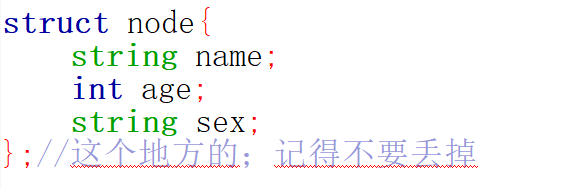


**七：map, set, multiset, multimap, list, deque感觉用的不多，需要用到的时候再学。**

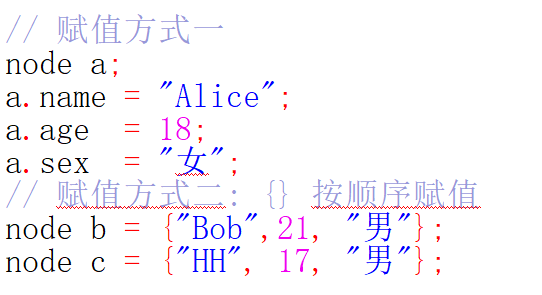
**结构体：**

大致意思就是把有相互关系的一些变量（可以是不同类型的）反正一起处理。

例如定义一个（姓名，年龄，性别）的结构体



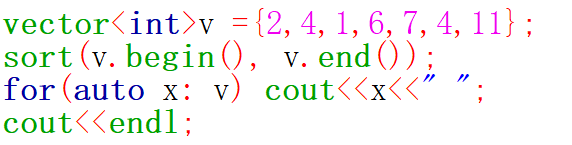
赋值：



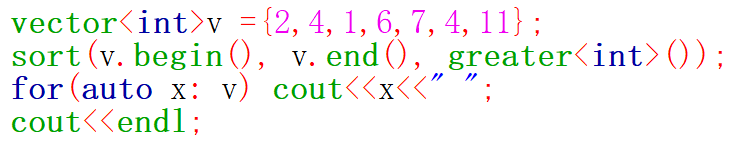
**排序： （太常用了，sort） 一定要会自定义排序规则，利用结构体**

需要包含头文件： # include<algorithm>

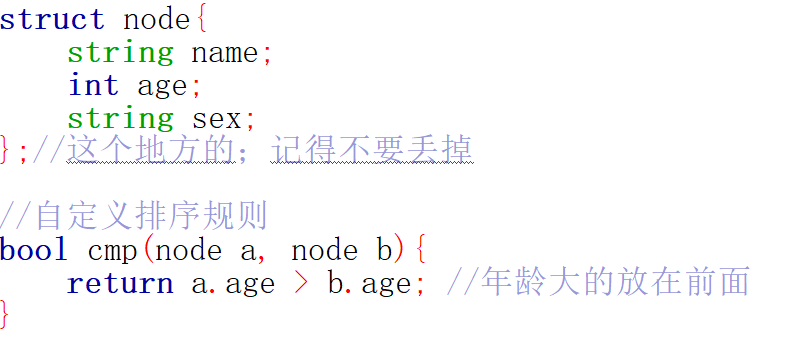
简单常规排序(默认从小到大)

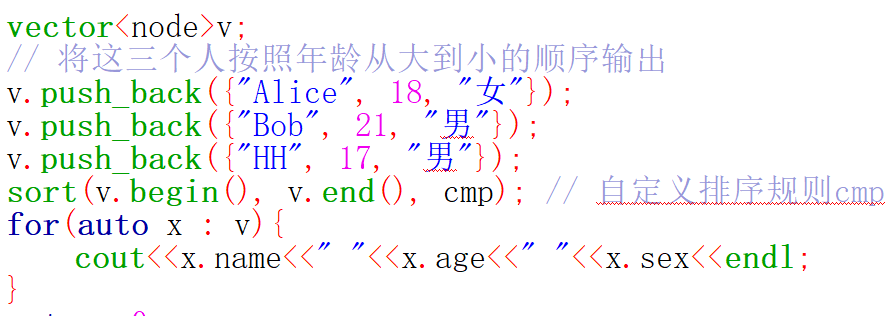
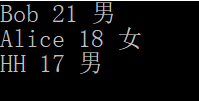
 

从大到小

自定义排序规则：以结构体样例为例，按照每个人的年龄顺序从大到小输出



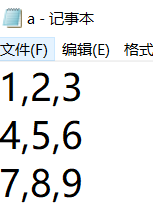
**文件读写：（有时需要从txt，csv文件中读数据）**

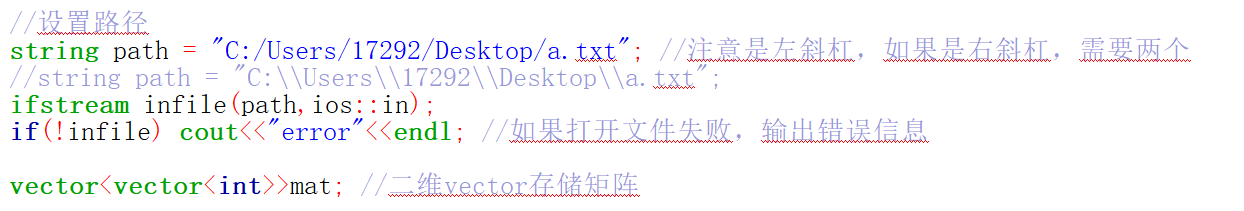
需要包含头文件： # include<fstream>

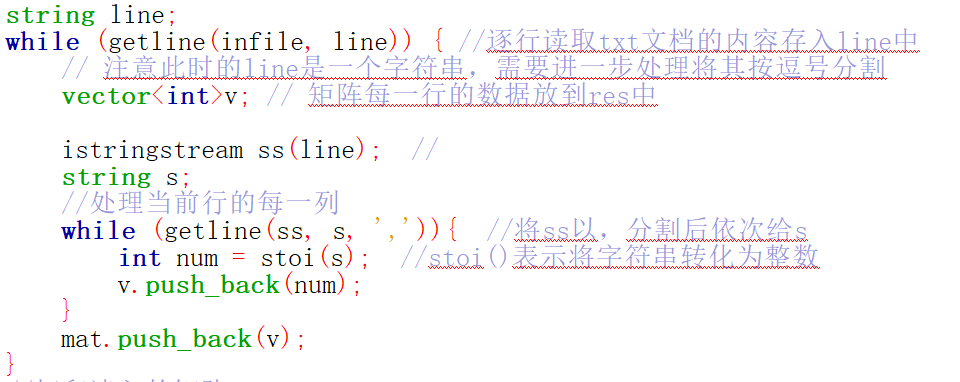
txt,csv文件类似，一样的读法

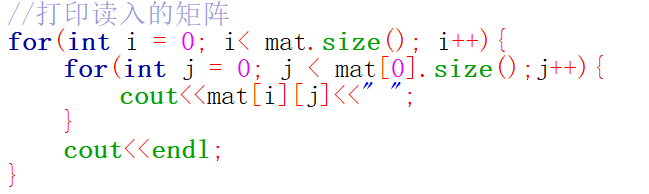
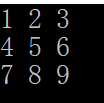
**读文件：**

从txt文档中读取信息，存放到上述的二维数组中。

txt文档内容如左图所示：





**注意点：**

（1）分隔符可以根据实际情况修改，空格” ”，逗号“，“或者其他符号

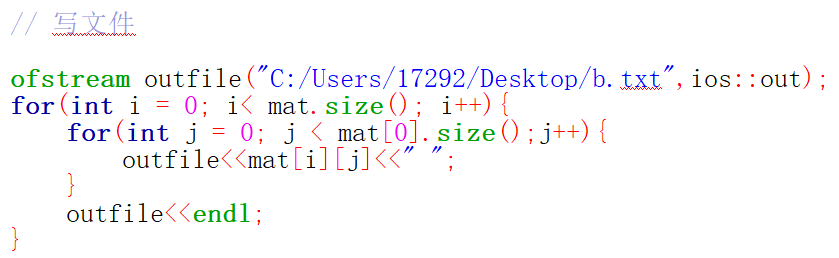
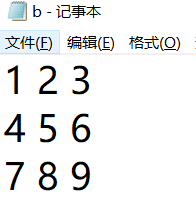
（2）用istringstream需要包含头文件 #include <sstream>

（3）不是固定的读法，网上版本很多，看自己喜欢哪个

**写文件：（比读文件简单）**

就是把内存中的数据写入文档中

例：将上述矩阵输出到另一个b.txt中。

leetcode



下面可以选择按类型刷题，通常按类型刷题提升较大，效率高，具体的顺序参照这个网站：https://programmercarl.com/

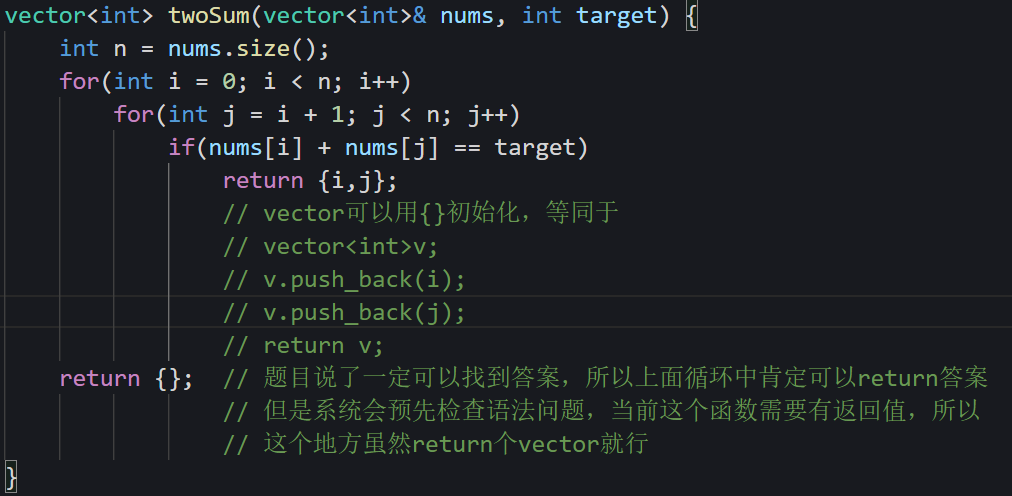


以第一题为例：经典两数之和：

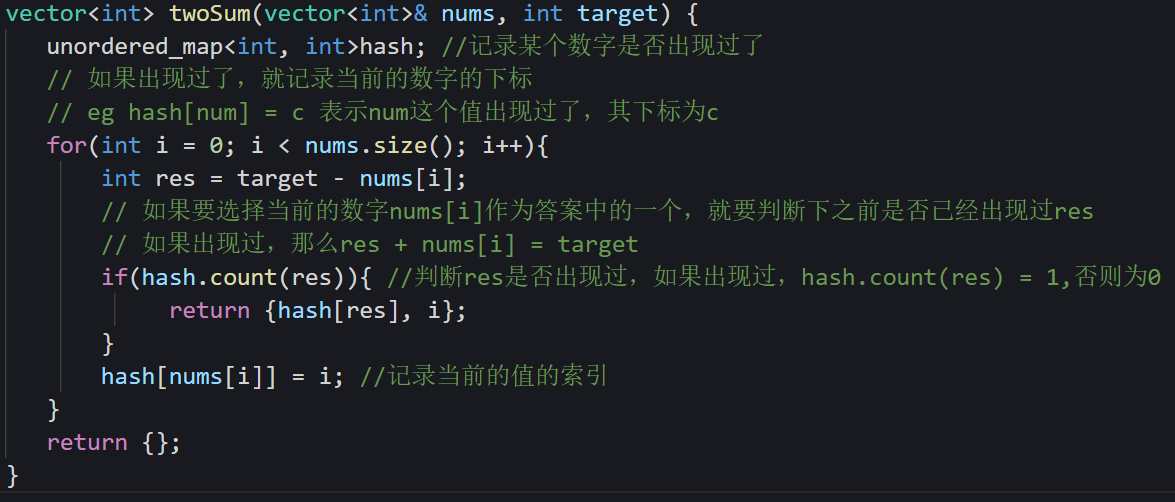


问题的基本意思就是给定nums，然后在nums中找到两个数，它们的和为target,返回这两个数的下标，因为涉及到返回多个值，因此题目返回值用了vector。

方法一：两重循环



方法二：哈希表







结果可以看到运行时间，也可以自己备注是什么方法，方便今后复习。

我刚开始刷题前也是十分恐惧，觉得应该把语言掌握的很好，各个点都掌握好才能开始刷题，但后来才知道，其实没太大必要，基本的语法会了，其实就可以大胆的开始写了。编程的核心在于思维而不是语言的语法，而思维这个东西是需要不断做题训练的，而不是通过看别人的代码获取的，所以自己动手去敲一敲效果好于看别人的，（哪怕是照着别人的代码自己动手敲一遍，也会很有收获）。

leetcode刷题时，一道题自己想了5分钟没思路就直接看题解，然后看懂题解，自己照着敲一篇，积累多了，渐渐就知道什么类型的题目用什么方法了，而且过程中看别人的代码，也可以学习到很多东西，语言语法等内容在实战的过程中就顺便复习了，如果有的点忘了，比如我一开始一直记不住二维vector初始化，就直接百度就找答案，不需要一定要记住所有的东西，不会的点，忘了的点直接查找就可以了。

上面的c++语法我仅仅总结了常用的一些知识点（严谨点：应该是我目前常用的一些知识），肯定会不全面，好多知识也可能较为零散，不成体系，或者跳跃性较大。但我都尽可能写了较为详细的注释帮助理解，希望能够对你有所帮助，加油！