目录

[数组做函数参数 1](#_Toc514325737)

[熟悉字符串操作 1](#_Toc514325738)

[获取一个未知变量的类型 2](#_Toc514325739)

[的使用（组成原理，数据和地址） 2](#_Toc514325740)

[深入熟悉Scanf 和 printf 4](#_Toc514325741)

[我们的一大弱项——string.h /cstring 4](#_Toc514325742)

[广度优先 4](#_Toc514325743)

[深度优先 4](#_Toc514325744)

[冒泡排序 4](#_Toc514325745)

[选择排序【垃圾】O(n2) O(1) 4](#_Toc514325746)

[插入排序【垃圾】O(n2) O(1) 4](#_Toc514325747)

[Shell Sort(递减增量排序算法)？？ 5](#_Toc514325748)

[位操作符： 7](#_Toc514325749)

[排序概述： 8](#_Toc514325750)

[Algorithm\_reverse() 9](#_Toc514325751)

[Printf输出以空格为间距： 9](#_Toc514325752)

[读取一长串数字字符并且输出数字的和： 10](#_Toc514325753)

[C++中字符串的读入和写出 10](#_Toc514325754)

[STL\_Map（红黑树） 10](#_Toc514325755)

[Map——insert 10](#_Toc514325756)

[STL—heap—完全二叉树 12](#_Toc514325757)

## 数组做函数参数

数组作为指针传进函数作为参数，指针的长度是8bit，然后如果函数需要使用数组作为参数，必须传入长度。

这是因为C++实际上是将数组作为指针来传递，而该指针指向数组的第一个元素。至于后面数组在哪里结束，C++的函数传递机制并不负责。

上面的特性可总结为，**数组仅在定义其的域范围内可确定大小**。

## 熟悉字符串操作

重点熟悉类的使用。包括。

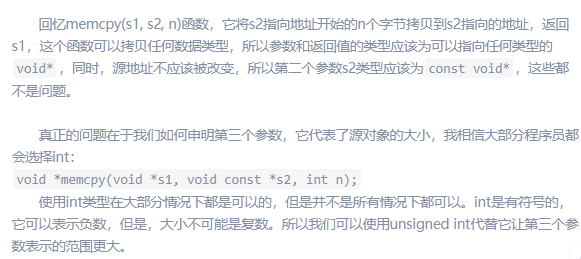
## 获取一个未知变量的类型

## 的使用（组成原理，数据和地址）

* int小于等于数据线宽度，大于等于地址线宽度。

**地址长度的发展总是比数据长度的发展要快。**

* **的取值range是目标平台下最大可能的数组尺寸（也就是最大地址，也就是寻址范围。。。）**
* **，就是大小的类型，也是的直接结果。**
* ，这里显然参数意味着“某块内存的大小”。
* X是变量，而不是类型。



**所以，这个第三个参数，由于是地址的多少，又由于不同的机器里面的大小是不一样的，所以，必须要首先获取，表示你到底要复制多少东西。最主要的原因就是int的大小，随着机器会改变，比如我64位机器，可能就是4bit int，而换一个可能是8bit int，这些一方面取决于计算机体系结构，一方面取决于C编译器。**

sizeof返回的必定是无符号整形,在标准c中通过 typedef 将返回值类型定义为size\_t。

sizeof，获取操作数占用的内存空间字节数,返回类型size\_t；多少多少个byte

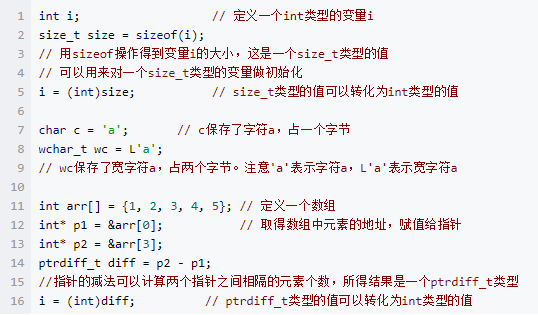
strlen，获取字符数组实际使用的字节数,不包含数组结尾符’\0’，返回类型size\_t；多少多少个byte

**但是注意，其实本质上 size\_t是的一个别名。可以转化为int并且赋值给int。**

**Size type的本质目的就是为了保存类型的大小。**

**————wide char type 就是记录一个宽字符的数据类型**

**——pointer difference type 记录两个指针之间距离的数据类型**



size\_t是全局定义的类型；size\_type是STL类中定义的类型属性，用以保存任意string和vector类对象的长度.

string::size\_type 制类型一般就是unsigned int, 但是不同机器环境长度可能不同 win32 和win64上长度差别;size\_type一般也是unsigned int

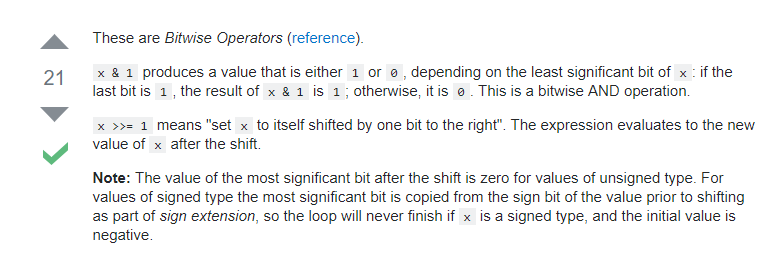
二者联系：在用下标访问元素时，vector使用vector::size\_type作为下标类型，而数组下标的正确类型则是size\_t【但是在操作的时候，使用普通的int就行了】

总结一下：基本用不到

## 深入熟悉Scanf 和 printf

## 我们的一大弱项——string.h /cstring

## 位操作符：



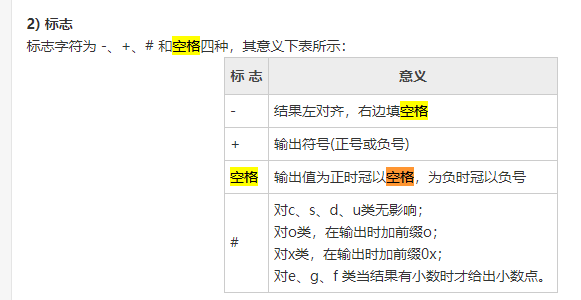
实践证明>>5 和 >>=5的效果是一样的，都是右移。也就是缩小

## Algorithm\_reverse()

* 推荐调用algorithm的泛型reverse。然后调用的方式：
* 对于vector——reverse(vectorName.begin(),vectorName.end());
* 然后也可以选择性的进行reverse，比如（vectorName.begin()+1,vectorName.end()-1）;

## Algorithm\_Swap()

## Printf输出以空格为间距：



也就是说，你想要输出空格，并不是单纯的打一个空格就行的，一旦输出的是符号，就很尴尬。

字符串处理的题，一般都是要注意EOF情况的，同时对于特殊符号的读取，char ASCII倒是没有问题，关键就是要注意能不能读取进来，所以一般使用getchar来读取。

注意一点：scanf输入的终止或者说是EOF符号，是需要使用CT+Z打印出来的。

## 读取一长串数字字符并且输出数字的和：

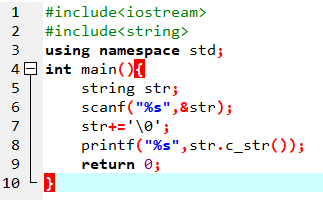
这个东西，看你怎么看，

* 看做是一个数字一个数字的读取【难以操作】
* 看做是读取一个string，然后逐个字符的-‘0’的输出。【便于操作】

然后scanf(“%s”,string\_name)需要增加&字符吗。——【需要】

不要使用printf和scanf处理任何和字符串相关的问题。

## C++中字符串的读入和写出



加了c\_str()和’\0’,完全没有任何的用处好不好。

推荐还是使用cin和cout处理C++类的string读写，因为本身C中对于字符串的读写就有着自己的问题。

## STL\_Map（红黑树）

map内部自建一颗红黑树(一种非严格意义上的平衡二叉树)，这颗树具有对数据自动排序的功能，所以在map内部所有的数据都是有序的，后边我们会见识到有序的好处。

两种方式构建字符串类型的东西，一个是string，一个是char\*。

### Map——insert

}



声明就是

map<int,string> myMap;

插入元素就是：

* myMap.insert(map<int,string>::value\_type(4,”student\_four”));
* myMap.insert(pair<int,string>(1,”student\_one”));
* 直接的mapStudent[1] = “student\_one”;这种方式

Insert可以是参考了集合的唯一性的情况进行的插入，而数组赋值的方法是覆盖的方法进行操作。

## STL—heap—完全二叉树