memset()（memory set）

函数原型：extern void \*memset(void \*buffer, int c, int count)

头文件：<string.h>或<memory.h>

功能：把buffer所指内存区域的前count个字节设置成字符c。

说明：返回指向buffer的指针(即内存地址)

buffer：为指针或是数组,

c：是赋给buffer的值,

count：是buffer的长度.

memset可以方便的清空一个聚合量（数组和结构）和指针。

数组：

struct sample\_struct TEST[10];

memset(TEST,0,sizeof(struct sample\_struct)\*10);

如果用malloc分配的内存，一般只能使用memset来初始化了。

字符串：

char a[100];memset(a, '/0', sizeof(a)); //’\0’和0其实是一样的

结构：

struct sample\_struct{

char csName[16];

int iSeq;

int iType;

};

对于变量：

struct sample\_strcut stTest;

一般情况下，清空stTest的方法：

stTest.csName[0]='/0';

stTest.iSeq=0;

stTest.iType=0;

用memset就非常方便：

memset(&stTest,0,sizeof(struct sample\_struct));

注意点：

1. memset中的第三个参数一定要使用sizeof操作符，因为每个系统下对类型长度的定义可能不一样。
2. memset中的第一个参数一定要是一个已知的、已经被分配内存的地址，否则会出错。
3. 返回一个void\*的指针去指向这个地址的用处：为了实现链式编程，所谓链式编程

memcpy(cBuf1, memset(cBuf, 'a', sizeof(char) \* 10), sizeof(char) \* 10);//很少用

（4）最后一点，也是最重要的一点。一定要注意，memset是按照字节对待初始化空间进行初始化的，也就是说，函数里面的第二个参数的那个初值（一般为0）是按照一个一个字节往第一个参数所指区域赋值的，所以，对于单字节数据类型（char）可以初始化为任意支持的值，都没有问题，但是对于非多字节数据类型只能初始化为0，而不能初始化成别的初值，因为对所有字节按任意顺序赋值0的结果都是0，而如果初始化为其他的值，就会一个字节一个字节的进行赋值，从而出现奇怪的结果。比如说，上面的例3之所以没有出错就是因为初始化为0，但是如果初始化为1，那么因为int一般是4个字节，那么相当于将一个int元素初始化成了0000 0001 0000 0001 0000 0001 0000 0001，这样对于一个int元素肯定不是1，而是一个很大的数，结果出乎意料，所以一定要记住这一点，非常重要！！！

memcpy()（memory copy）

原型：extern void \*memcpy(void \*dest, void \*src, unsigned int count);

用法：#include <string.h>

功能：由src所指内存区域复制count个字节到dest所指内存区域。

说明：src和dest所指内存区域不能重叠，函数返回指向dest的指针

char \*s="Golden Global View";

char d[20];

memcpy(d,s,strlen(s));

https://blog.csdn.net/qq\_35040828/article/details/71123521