

★ 1. memcpy

```
void * memcpy ( void * destination, const void * source, size_t num );
```

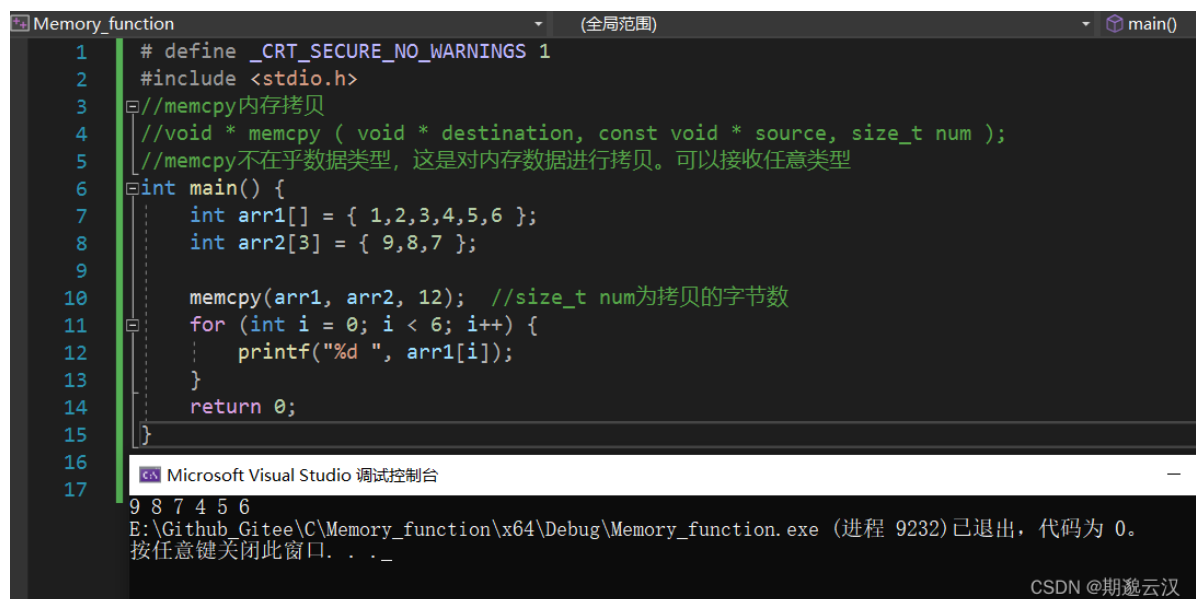
函数memcpy从source的位置开始向后复制num个字节的数据到destination的内存位置。

这个函数在遇到 '\0' 的时候并不会停下来。

如果source和destination有任何的重叠，复制的结果都是未定义的。

```
#include <stdio.h>
//memcpy内存拷贝
//void * memcpy ( void * destination, const void * source, size_t num );
//memcpy不在乎数据类型，这是对内存数据进行拷贝。可以接收任意类型
int main() {
    int arr1[] = { 1,2,3,4,5,6 };
    int arr2[3] = { 9,8,7 };

    memcpy(arr1, arr2, 12); //size_t num为拷贝的字节数
    for (int i = 0; i < 6; i++) {
        printf("%d ", arr1[i]);
    }
    return 0;
}
```



```
Memory function (全局范围) main()
1 # define _CRT_SECURE_NO_WARNINGS 1
2 #include <stdio.h>
3 //memcpy内存拷贝
4 //void * memcpy ( void * destination, const void * source, size_t num );
5 //memcpy不在乎数据类型，这是对内存数据进行拷贝。可以接收任意类型
6 int main() {
7     int arr1[] = { 1,2,3,4,5,6 };
8     int arr2[3] = { 9,8,7 };
9
10    memcpy(arr1, arr2, 12); //size_t num为拷贝的字节数
11    for (int i = 0; i < 6; i++) {
12        printf("%d ", arr1[i]);
13    }
14    return 0;
15 }
16
17 Microsoft Visual Studio 调试控制台
9 8 7 4 5 6
E:\Github_Gitee\C\Memory_function\x64\Debug\Memory_function.exe (进程 9232)已退出，代码为 0。
按任意键关闭此窗口。 . . .
CSDN @期遛云汉
```

在监视窗口可以看到根据我们指定拷贝多少字节，内存布局是显然的。



★实现memcpy函数

```
//实现Mymemcpy
void* Mymemcpy(void * dest, void *src,int size_num) {
    void* ret = dest;
    while (size_num--) {
        *(char*)dest = *(char*)src;

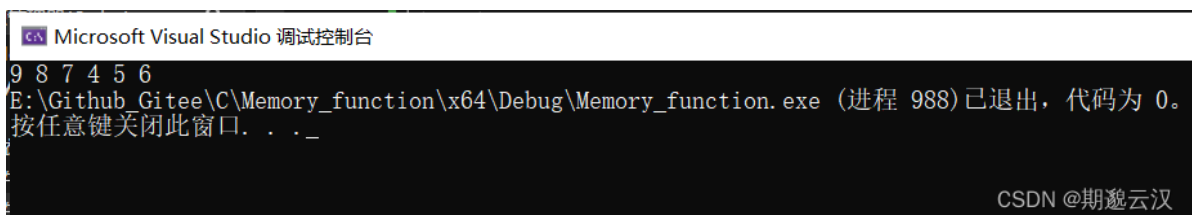
        ((char*)dest)++;
        ((char*)src)++;
    }
    return ret; //返回目标初始指针
}

int main() {
    int arr1[] = { 1,2,3,4,5,6 };
    int arr2[3] = { 9,8,7 };

    //memcpy(arr1, arr2, 12); //size_t num为拷贝的字节数
    Mymemcpy(arr1, arr2, 12);
    for (int i = 0; i < 6; i++) {
        printf("%d ", arr1[i]);
    }
    return 0;
}

//如果实现重叠内存的拷贝，不需要memcpy，重叠内存拷贝要使用memmove，但VS编译器的memcpy实现了重叠内存的拷贝
```

测试一下：



★2. 实现memmove

```

//实现memmove (实现重叠内存的拷贝)
//void* memmove(void* destination, const void* source, size_t num);
#include <assert.h>

void* myMemmove(void* dest, void* src, size_t num)
{
    void* ret = dest;
    assert(dest);
    assert(src);

    if (dest < src) //1 前->后
    {
        while(num--)
        {
            *(char*)dest = *(char*)src;
            dest = (char*)dest + 1;
            src = (char*)src + 1;
        }
    }
    else //2 3 后->前
    {
        while (num--)
        {
            *((char*)dest + num) = *((char*)src + num);
        }
    }
    return ret;
}

//memcpy只需要实现不重叠的拷贝就可以了 - 60 100
//memmove是需要实现重叠内存的拷贝的

int main()
{
    int arr1[] = { 1,2,3,4,5,6,7,8,9,10 };
    int arr2[10] = { 0 };
    myMemmove(arr2, arr1, 20);

    for (int i = 0; i < 10; i++)
    {
        printf("%d ", arr2[i]);
    }
    return 0;
}

```

3. memcpy内存比较

内存比较函数

```

int memcmp ( const void * ptr1,
             const void * ptr2,
             size_t num );

```

比较从ptr1和ptr2指针开始的num个字节

```
/* memcmp example */
#include <stdio.h>
#include <string.h>
int main ()
{
    char buffer1[] = "DWgaOtP12df0";
    char buffer2[] = "DWGAOTp12DF0";
    int n;
    n=memcmp ( buffer1, buffer2, sizeof(buffer1) );
    if (n>0) printf ("%s' is greater than '%s'.\n",buffer1,buffer2);
    else if (n<0) printf ("%s' is less than '%s'.\n",buffer1,buffer2);
    else printf ("%s' is the same as '%s'.\n",buffer1,buffer2);
    return 0;
}
```

```
//memcpy内存比较函数
/* memcmp example */
#include <string.h>
int main()
{
    char buffer1[] = "DWgaOtP12df0";
    char buffer2[] = "DWGAOTp12DF0";
    int n;
    n = memcmp(buffer1, buffer2, sizeof(buffer1));
    if (n > 0)
        printf("%s' is greater than '%s'.\n", buffer1, buffer2);
    else if (n < 0)
        printf("%s' is less than '%s'.\n", buffer1, buffer2);
    else
        printf("%s' is the same as '%s'.\n", buffer1, buffer2);
    return 0;
}
```

Microsoft Visual Studio 调试控制台

'DWgaOtP12df0' is greater than 'DWGAOTp12DF0'.

CSDN @期趣云

4. memset

将指定字节大小的内存段设置为期望的值

```
void *memset(void *a, int ch, size_t length);
```

代码如下（示例）：可以将一段内存空间全部设置为特定的值，所以经常用来初始化字符数组。

```
//memset
//以内存字节数来设置值
int main() {
    int arr[] = { 1,2,3,4,5 };
    memset(arr, 0, 12);
    return 0;
}
```

结果显然：

名称	值	类型
arr	0x00000023a913f9c8 {0, 0, 0,...	int[5]
[0]	0	int
[1]	0	int
[2]	0	int
[3]	4	int
[4]	5	int
添加要监视的项	CSDN @期邈云汉	

总结

内存操作函数以每个基本的内存字节数量进行操作，通常需要指定操作的内存字节大小