

第17讲：C语言内存函数

目录：

1. memcpy使用和模拟实现
2. memmove使用和模拟实现
3. memset函数的使用
4. memcmp函数的使用

正文开始

1. memcpy 使用和模拟实现

```
1 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指向的内存位置。
- 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。

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main()
5 {
6     int arr1[] = { 1,2,3,4,5,6,7,8,9,10 };
7     int arr2[10] = { 0 };
8     memcpy(arr2, arr1, 20);
9     int i = 0;
10    for (i = 0; i < 10; i++)
11    {
12        printf("%d ", arr2[i]);
13    }
14    return 0;
15 }
```

对于重叠的内存，交给memmove来处理。

memcpy函数的模拟实现:

```
1 void * memcpy ( void * dst, const void * src, size_t count)
2 {
3     void * ret = dst;
4     assert(dst);
5     assert(src);
6     /*
7      * copy from lower addresses to higher addresses
8      */
9     while (count-->0) {
10         *(char *)dst = *(char *)src;
11         dst = (char *)dst + 1;
12         src = (char *)src + 1;
13     }
14
15     return(ret);
16 }
```

2. memmove 使用和模拟实现

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

- 和memcpy的差别就是memmove函数处理的源内存块和目标内存块是可以重叠的。
- 如果源空间和目标空间出现重叠，就得使用memmove函数处理。

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main()
5 {
6     int arr1[] = { 1,2,3,4,5,6,7,8,9,10 };
7     memmove(arr1+2, arr1, 20);
8     int i = 0;
9     for (i = 0; i < 10; i++)
10     {
11         printf("%d ", arr1[i]);
12     }
```

```

13     return 0;
14 }

```

输出的结果：

```

1      1 2 1 2 3 4 5 8 9 10

```

memmove的模拟实现：

```

1 void * memmove ( void * dst, const void * src, size_t count)
2 {
3     void * ret = dst;
4     if (dst <= src || (char *)dst >= ((char *)src + count)) {
5         /*
6          * Non-Overlapping Buffers
7          * copy from lower addresses to higher addresses
8          */
9         while (count-- > 0) {
10             *(char *)dst = *(char *)src;
11             dst = (char *)dst + 1;
12             src = (char *)src + 1;
13         }
14     }
15     else {
16         /*
17          * Overlapping Buffers
18          * copy from higher addresses to lower addresses
19          */
20         dst = (char *)dst + count - 1;
21         src = (char *)src + count - 1;
22
23         while (count-- > 0) {
24             *(char *)dst = *(char *)src;
25             dst = (char *)dst - 1;
26             src = (char *)src - 1;
27         }
28     }
29
30     return(ret);
31 }

```

3. memset 函数的使用

比特就业课主页：<https://m.cctalk.com/inst/s9yewhfr>

```
1 void * memset ( void * ptr, int value, size_t num );
```

memset是用来设置内存的，将内存中的值以字节为单位设置成想要的内容。

```
1 #include <stdio.h>
2 #include <string.h>
3 int main ()
4 {
5     char str[] = "hello world";
6     memset (str, 'x', 6);
7     printf(str);
8     return 0;
9 }
```

输出的结果：

```
1 xxxxxxworld
```

4. memcmp 函数的使用

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

- 比较从ptr1和ptr2指针指向的位置开始，向后的num个字节
- 返回值如下：

Return Value

Returns an integral value indicating the relationship between the content of the memory blocks:

return value	indicates
<0	the first byte that does not match in both memory blocks has a lower value in <i>ptr1</i> than in <i>ptr2</i> (if evaluated as <i>unsigned char</i> values)
0	the contents of both memory blocks are equal
>0	the first byte that does not match in both memory blocks has a greater value in <i>ptr1</i> than in <i>ptr2</i> (if evaluated as <i>unsigned char</i> values)

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main()
5 {
6     char buffer1[] = "DWga0tP12df0";
7     char buffer2[] = "DWGA0TP12DF0";
8     int n;
9     n = memcmp(buffer1, buffer2, sizeof(buffer1));
10
11     if (n > 0)
12         printf("'s' is greater than 's'.\n", buffer1, buffer2);
13     else if (n < 0)
14         printf("'s' is less than 's'.\n", buffer1, buffer2);
15     else
16         printf("'s' is the same as 's'.\n", buffer1, buffer2);
17
18     return 0;
19 }
```

完