第18讲：C语言内存函数

目录：

正文开始

1. memcpy

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

功能：

- `memcpy` 是完成内存块拷贝的，不关注内存中存放的数据是啥
- 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。
- 如果 `source` 和 `destination` 有任何的重叠，复制的结果都是未定义的。
  - （内存重叠的情况使用 `memmove` 就行）
- `memcpy` 的使用需要包含 `<string.h>`

参数：

`destination`：指针，指向目标空间，拷贝的数据存放在这里

`source`：指针，指向源空间，要拷贝的数据从这里来

`num`：要拷贝的数据占据的字节数

返回值：

拷贝完成后，返回目标空间的起始地址

1.1 代码演示

```c
#include <stdio.h>
#include <string.h>

int main()
{
    int arr1[] = { 1,2,3,4,5,6,7,8,9,10 };
    int arr2[10] = { 0 };
    memcpy(arr2, arr1, 20);
    int i = 0;
    for (i = 0; i < 10; i++)
    {
        printf("%d ", arr2[i]);
    }
    return 0;
}
```

1.2 模拟实现

```c
void * memcpy ( void * dst, const void * src, size_t count)
{
    void * ret = dst;
    assert(dst);
    assert(src);
    /*
     * copy from lower addresses to higher addresses
     */
    while (count--) {
            *(char *)dst = *(char *)src;
            dst = (char *)dst + 1;
            src = (char *)src + 1;
    }

    return(ret);
}
```

2. memmove

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

功能：

- memmove函数也是完成内存块拷贝的
- 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。
- memmove的使用需要包含<string.h>

参数：

`destination`：指针，指向目标空间，拷贝的数据存放在这里

`source`：指针，指向源空间，要拷贝的数据从这里来

`num`：要拷贝的数据占据的字节数

返回值：

拷贝完成后，返回目标空间的起始地址

2.1 代码演示

```c
#include <stdio.h>
#include <string.h>

int main()
{
    int arr1[] = { 1,2,3,4,5,6,7,8,9,10 };
    memmove(arr1 + 2, arr1, 20);
    int i = 0;
    for (i = 0; i < 10; i++)
    {
        printf("%d ", arr1[i]);
    }
    return 0;
}
```

输出的结果：

```
1 2 1 2 3 4 5 8 9 10
```

2.2 模拟实现

```c
void * memmove ( void * dst, const void * src, size_t count)
```

```
{
    void * ret = dst;
    if (dst <= src || (char *)dst >= ((char *)src + count)) {
        /*
         * Non-Overlapping Buffers
         * copy from lower addresses to higher addresses
         */
        while (count--) {
                *(char *)dst = *(char *)src;
                dst = (char *)dst + 1;
                src = (char *)src + 1;
        }
    }
    else {
        /*
         * Overlapping Buffers
         * copy from higher addresses to lower addresses
         */
        dst = (char *)dst + count - 1;
        src = (char *)src + count - 1;

        while (count--) {
                *(char *)dst = *(char *)src;
                dst = (char *)dst - 1;
                src = (char *)src - 1;
        }
    }

    return(ret);
}
```

3. memset

代码块

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

功能：

- `memset` 函数是用来设置内存块的内容的，将内存中指定长度的空间设置为特定的内容。
- `memset` 的使用需要包含 `<string.h>`

参数：

`ptr`：指针，指向要设置的内存空间，也就是存放了要设置的内存空间的起始地址。

`value`：要设置的值，函数将会把 `value` 值转换成 `unsigned char` 的数据进行设置的。 也就是以字节为单位来设置内存块的。

`num`：要设置的内存长度，单位是字节。

返回值：返回的是要设置的内存空间的起始地址。

3.1 代码演示

代码块

```c
#include <stdio.h>
#include <string.h>
int main ()
{
    char str[] = "hello world";
    memset (str,'x',6);
    printf(str);
    return 0;
}
```

输出的结果：

代码块

```
xxxxxxworld
```

3.2 总结：

当有一块内存空间需要设置内容的时候，就可以使用memset函数，值得注意的是memset函数对内存单元的设置是以字节为单位的。

4. memcmp

代码块

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

功能：

比较指定的两块内存块的内容，比较从ptr1和ptr2指针指向的位置开始，向后的num个字节

`memcmp` 的使用需要包含 `<string.h>`

参数：

`ptr1`：指针，指向一块待比较的内存块

`ptr2`：指针，指向另外一块待比较的内存块

`num`：指定的比较长度，单位是字节

返回值：

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 *ptr1* than in *ptr2* (if evaluated as *unsigned char* 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 *ptr1* than in *ptr2* (if evaluated as *unsigned char* values)

4.1 代码演示

代码块

```c
#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' 大于 '%s'.\n", buffer1, buffer2);
        else if (n < 0)
            printf("'%s' 小于 '%s'.\n", buffer1, buffer2);
        else
            printf("'%s' 和 '%s' 一样.\n", buffer1, buffer2);

        return 0;
}
```

4.2 总结

如果要比较2块内存单元的数据的大小，可以使用 `memcmp` 函数，这个函数的特点就是可以指定比较长度。

`memcmp` 函数是通过返回值告知大小关系的。

完