strncpy：

```
//模拟实现strncpy
char* my_strncpy(char* str1,const char* str2,size_t num)
{
    assert(str1&&str2);
    char* tmp = str1;
    while ((*str1++ = *str2++) && --num)
        ;
    if (num)
        while (num--)
        {
            *str1++ = '\0';
        }
    return tmp;
}
```

strncat：

```
//模拟实现strncat
char* my_strncat(char* str1, char* str2, size_t num)
{
    assert(str1&&str2);
    char* tmp = str1;
    while (*str1!='\0')
        str1++;
    while ((*str1++ = *str2++) && --num)
        ;
    if (num)
        while (num--)
            *str1++ = '\0';
    return tmp;
}
```

strstr：

```
//模拟实现strstr
char * my_strstr (const char * str1, const char * str2)
{
    char *cp = (char *) str1;
    char *s1, *s2;
    if ( !*str2 )
        return((char *)str1);
    while (*cp)
    {
        s1 = cp;
        s2 = (char *) str2;
        while ( *s1 && *s2 && !(*s1-*s2) )
        s1++, s2++;
        if (!*s2)
            return(cp);
        cp++;
    }
    return(NULL);
}
```

strtok：

```
char * strtok ( char * str, const char * sep);
```

sep参数指向一个字符串，定义了用作分隔符的字符集合
• 第一个参数指定一个字符串，它包含了0个或者多个由sep字符串中一个或者多个分隔符分割的标记。
• strtok函数找到str中的下一个标记，并将其用 \0 结尾，返回一个指向这个标记的指针。（注：strtok函数会改变被操作的字符串，所以在使用strtok函数切分的字符串一般都是临时拷贝的内容并且可修改。）
• strtok函数的第一个参数不为 NULL ，函数将找到str中第一个标记，strtok函数将保存它在字符串中的位置。
• strtok函数的第一个参数为 NULL ，函数将在同一个字符串中被保存的位置开始，查找下一个标记。
• 如果字符串中不存在更多的标记，则返回 NULL 指针。

strerror：

```
#include
char * strerror ( int errnum );
```

strerror函数可以把参数部分错误码对应的错误信息的字符串地址返回来。
在不同的系统和C语言标准库的实现中都规定了一些错误码，一般是放在 errno.h 这个头文件中说明的，C语言程序启动的时候就会使用一个全面的变量errno来记录程序的当前错误码，只不过程序启动的时候errno是0，表示没有错误，当我们在使用标准库中的函数的时候发生了某种错误，就会讲对应的错误码，存放在errno中，而一个错误码的数字是整数很难理解是什么意思，所以每一个错误码都是有对应的错误信息的。strerror函数就可以将错误对应的错误信息字符串的地址返回。

在Windows VS2022环境下：

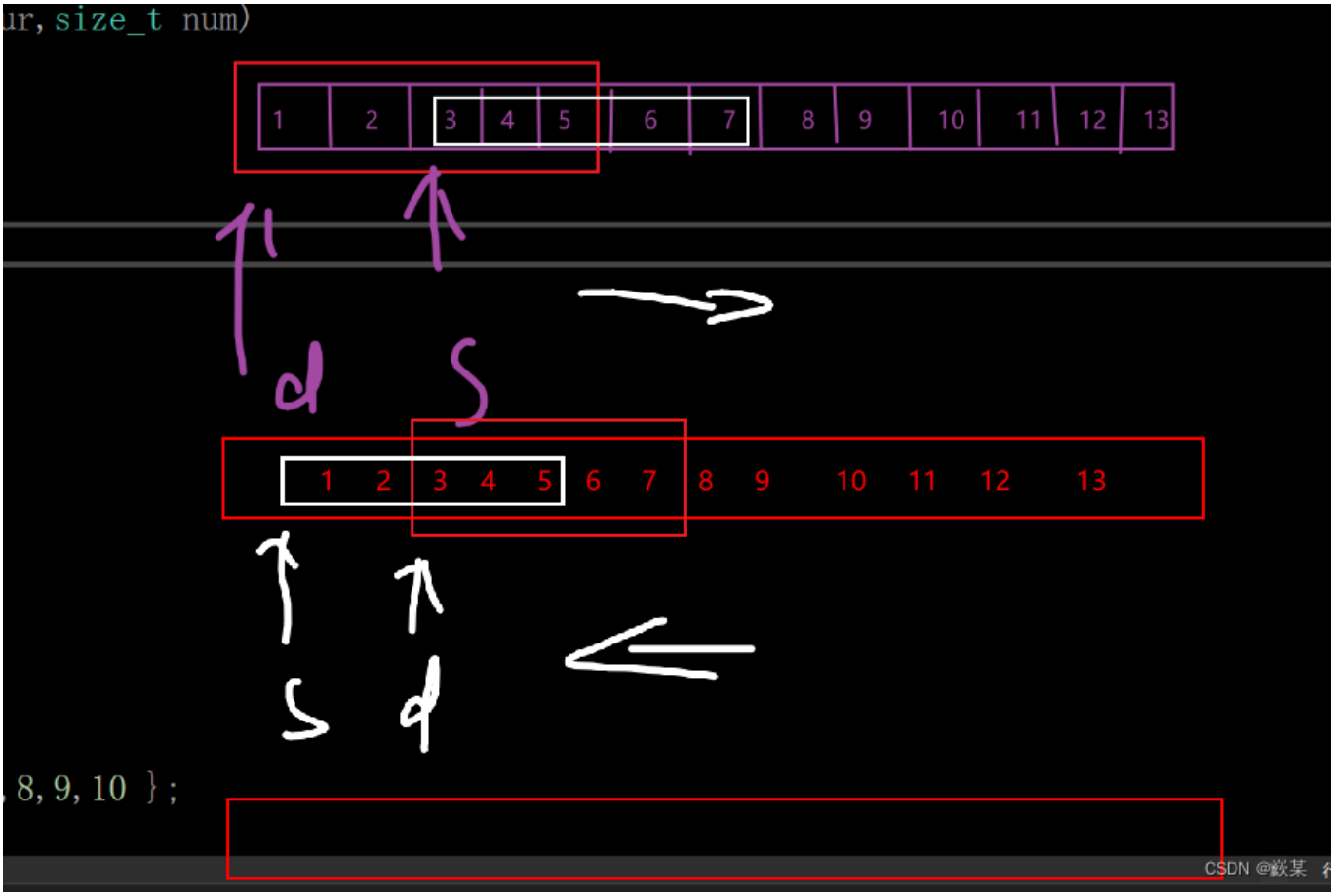
1	**No error**
2	Operation not permitted
3	No such file or directory
4	No such process
5	Interrupted function call
6	Input/output error
7	No such device or address
8	Arg list too long
9	Exec format error
10	Bad file descriptor
11	No child processes

memcpy:

```
//模拟实现memcpy
void * my_memcpy ( void * dst, const void * src, size_t count)
{
    void * ret = dst;
    assert(dst);
```

```
    assert(src);
    while (count--)
    {
        *(char *)dst = *(char *)src;
        dst = (char *)dst + 1;
        src = (char *)src + 1;
    }
    return(ret);
}
```

memove（这里注意根据dest和sour地址的高低分情况）：




```
//模拟实现memmove
void* my_memmove(void* dest,void* sour,size_t num)
{
    assert(dest && sour);
    char* tmp = (char*)dest;
    if (dest == sour)
        return dest;
    else if (dest < sour)
```

```
    {
        while (num--)
        {
            *(((char*)dest)++) = *(((char*)sour)++);
        }
    }
    else
    {
        while (num--)
        {
            *(((char*) dest) + num) = *(((char*) sour) + num);
        }
    }
    return tmp;
}
```

memset:

```
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。

memcmp:

```
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（和strcmp差不多，memcmp可以比较任何内容）。

本期博客到这里就结束了，如果有什么错误，欢迎指出，如果对你有帮助，请点个赞，谢谢！