c语言字符串操作总结

1. strlen函数

1. **原型**：`unsigned int strlen(const char * str)`
2. **作用**：计算str的长度并返回
3. **源码：**

```
#include <stdio.h>

size_t my_strlen(const char * str)
{
    unsigned len = 0;

    while(*str != '\0')
    {
        len++;
        str++;
    }

    return len;
}

int main()
{
    char * str = "abcdefgh";
    int len = 0;

    len = my_strlen(str);

    printf("%d\n", len);

    return 0;
}
```

2、strcpy函数与strncpy函数

strcpy函数

1. **原型**：`char *strcpy(char * dest, const char * src)`
2. **作用**：把src指向的字符串拷贝到dest指向的字符串中
3. **源码：**

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

char *my_strcpy(char *dest, const char *src)
{
    assert((dest != NULL) && (src != NULL));
```

```
    char *o_dest = dest;

    while (*src != '\0')
    {
        *dest++ = *src++;
    }
    *dest = '\0';

    return o_dest;
}

int main()
{
    char dest[10] = {0};
    char *src = "abcdefgh";

    my_strcpy(dest, src);
    printf("%s\n", dest);

    return 0;
}
```

strncpy函数

4. **原型**：`char *strncpy(char *dest, char *src, int n)`
5. **作用**：把str2指向的前count个字符拷贝到str1指向的字符串中
6. **源码：**

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

char *my_strncpy(char *dest, const char *src, size_t n)
{
   size_t i;

   for (i = 0; i < n && src[i] != '\0'; i++)
   {
       dest[i] = src[i];
   }

   for ( ; i < n; i++)
   {
       dest[i] = '\0';
   }

   return dest;
}


int main()
{
    char dest[10] = {0};
    char *src = "abcdefgh";
    int n = 5;
```

```
    my_strncpy(dest, src, n);
    printf("%s\n", dest);

    return 0;
}
```

3、strcmp函数

1. **原型**：`int strcmp(const char *s1, const char *s2)`
2. **作用**：比较str1和str2，str1 > str2返回1，str1 == str2返回0
3. **源码：**

```
#include <stdio.h>
#include <assert.h>

int my_strcmp(const char *s1, const char *s2)
{
    assert(s1 != NULL && s2 != NULL);

    while (*s1 != '\0')
    {
        if (*s1 != *s2)
        {
            break;
        }

        s1++;
        s2++;
    }

    if (*s1 == *s2)
    {
        return 0;
    }else if(*s1 > *s2)
    {
        return 1;
    }else
    {
        return -1;
    }
}

int main()
{
    char *s1 = "aaaaa";
    char *s2 = "aaaab";
    char *s3 = "aaaac";
    int ret = 0;

    ret = my_strcmp(s2, s1);
    if (ret > 0)
    {
        printf("string 2 is bigger than string 1\n");
```

```
    }
    else
    {
        printf("string 2 is smaller than string 1\n");
    }

    ret = my_strcmp(s2, s3);
    if (ret > 0)
    {
        printf("string 2 is bigger than string 3\n");
    }
    else
    {
        printf("string 2 is smaller than string 3\n");
    }

    return 0;
}
```

4、strncmp函数

1. **原型**：`int strncmp(const char * s1, const char * s2, int n)`
2. **作用**：比较str1和str2的前n个字符
3. **源码**：

```
#include <stdio.h>
#include <assert.h>

int my_strncmp(const char * s1, const char * s2, int n)
{
    assert(s1 != NULL &&s2 != NULL & n > 0);

    while(--n && *s1 != 0)
    {
        if (*s1 != *s2)
        {
            break;
        }
        s1++;
        s2++;
    }

    if (*s1 == *s2)
    {
       return 0;
    }else if(*s1  > *s2)
    {
       return 1;
    }else
    {
       return -1;
    }
}
```

```
int main()
{
    char *str1 = "China is a nation!";
    char *str2 = "French is a nation!";
    int count = 5;
    int ret = 0;

    ret = my_strncmp(str1, str2, count);
    if(ret != 0)
        printf("str1 is not equal to str2!\n");

    return 0;
}
```

5、strcasecmp、strncasecmp与stricmp函数

1. **原型**：`#include <strings.h>`

   `int strcasecmp(const char *s1, const char *s2);`

   `int strncasecmp(const char *s1, const char *s2, size_t n);`

   `int stricmp(const char *s1, const char *s2);` **备注： 仅在windows环境中使用**

2. **作用**：不区分大小写的比较str1和str2

3. **源码：**

```
#include <stdio.h>
#include <strings.h>

int my_strcasecmp(const char *s1, const char *s2)
{
    char ch1;
    char ch2;

    while(*s1 != '\0')
    {
        ch1 = *s1++;
        if(ch1 >= 'A' && ch1 <= 'Z')
        {
            ch1 += 'a' - 'A';
        }

        ch2 = *s2++;
        if(ch2 >= 'A' && ch2 <= 'Z')
        {
            ch2 += 'a' - 'A';
        }

        if (ch1 != ch2)
        {
            break;
        }
    }

    if (ch1 == ch2)
    {
        return 0;
```

```
    }else if (ch1 > ch2)
    {
        return 1;
    }else
    {
        return -1;
    }
}

int main()
{
    char *s1= "ammana";
    char *s2 = "bibi";
    char *s3 = "AMMANA";
    int ret1 = 0;
    int ret2 = 0;

    ret1 = my_strcasecmp(s1, s2);
    if(ret1 > 0)
    {
        printf("str1 bigger than str2!\n");
    }
    else
    {
        printf("str1 smaller than str2!\n");
    }

    ret2 = my_strcasecmp(s1, s3);
    if(ret2 > 0)
    {
        printf("str1 bigger than str3!\n");
    }
    else if(ret2 < 0)
    {
        printf("str1 smaller than str3!\n");
    }
    else
    {
        printf("str1 equal to str3!\n");
    }

    return 0;
}
```

6、strcat函数

1. **原型**：`char *strcat(char *dest, const char *src)`
2. **作用**：连接两个字符串，将src连接到dest
3. **源码：**
4. ```
   #include <stdio.h>

   char * my_strcat(char* dest, const char *src)
   {
   ```
   ```

```
    char * dest_o = dest;

    while(*dest != '\0')
    {
        dest++;
    }

    while(*src != '\0')
    {
        *dest++ = *src++;
    }
    *dest = '\0';

    return dest_o;
}

int main()
{
    char dest[25];
    char *space = " ";
    char *s1 = "Hello";
    char *s2 = "world!";

    strcpy(dest, s1);
    my_strcat(dest, space);
    my_strcat(dest, s2);

    printf("%s\n", dest);

    return 0;
}
```

7、strncat函数

1. **原型**：`char *strncat(char *dest, const char *src, size_t n);`
2. **作用**：连接两个字符串，将src连接到dest
3. **源码：**

```
#include <stdio.h>
#include <assert.h>

char *my_strncat(char *dest, const char *src, int n)
{
   assert((dest != NULL) && (src  != NULL));

   char *dest_o = dest;

   while (*dest != '\0')
   {
       dest++;
   }

   while (n-- && *src != '\0' )
   {
```

```
        *dest++ = *src++;
    }
    *dest = '\0';

    return dest_o;
}

int main()
{
    char dest[25];
    char *space = " ";
    char *s1 = "Hello";
    char *s2 = "world!";

    strcpy(dest, s1);
    my_strncat(dest, space, 4);
    my_strncat(dest, s2, 5);

    printf("%s\n", dest);

    return 0;
}
```

8、strchr函数

1. **原型**：`char * strchr(char * str, const char c)`
2. **作用**：查找str中c首次出现的位置，并返回位置或者NULL
3. **源码：**

```
#include <stdio.h>

char * my_strchr(char *str, const char c)
{
    while(*str != '\0')
    {
        if (*str == c)
        {
            return str;
        }
        str++;
    }

    return NULL;
}

int main()
{
    char *s1 = "abcdefgh";
    char c = 'd';
    char *ptr = NULL;

    ptr = my_strchr(s1, c);
    if(ptr != NULL)
    {
        printf("%c %c\n", *ptr, c);
    }
```

```
    else
    {
        printf("Not Found!\n");
    }

    return 0;
}
```

9、strrchr函数

1. **原型**：`char * strrchr(char * str, const char c)`
2. **作用**：查找str中c最后一次出现的位置，并返回位置或者NULL
3. **源码**：

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

char * my_strrchr(char * str, const char c)
{
    char * end = str + strlen(str);

    while(str != end)
    {
        if (*end == c)
        {
            return end;
        }
        end--;              //向前进行移动
    }

    return NULL;
}

int main()
{
    char * string = "abcdefdefgh";
    char c = 'd';
    char *p = NULL;

    p = my_strrchr(string, c);
    if(p != NULL)
    {
        printf("%s\n", p);
    }
    else
    {
        printf("Not Found!\n");
    }

    return 0;
}
```

10、strrev函数

1. **原型**：`char * strrev(char * str)`
2. **作用**：翻转字符串并返回字符串指针
3. **源码**：

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

char *my_strrev(char * str)
{
    assert(str != NULL);

    char *end = str;
    char *head = str;
    char tmp;

    while (*end != '\0')
    {
        end++;
    }
    end--;              /* 回跳过结束符'\0' */

    /* 当head和end未重合时，交换它们所指向的字符 */
    while (head < end)
    {
        tmp = *head;
        *head= *end;    /* head向尾部移动 */
        *end= tmp;      /* end向头部移动 */

        head++;
        end--;
    }

    return str;
}

#if 0
/*第二种实现方式*/
char * my_strrev(char * str)
{
    assert(str != NULL);

    char *end = strlen(str) + str - 1;
    char temp = '0';

    while(str < end)
    {
        temp = *str;
        *str = *end;
        *end = temp;

        str++;
        end--;
    }
```

```
    return str;
}
#endif


int main()
{
    char str[] = "Hello World";

    printf("Before reversal: %s\n", str);
    my_strrev(str);
    printf("After reversal:  %s\n", str);

    return 0;
}
```

11、strdup函数

1. **原型**：`char * strdup(const char * str)`
2. **作用**：拷贝字符串到新申请的内存中返回内存指针，否则返回NULL
3. **源码**：

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>

char * my_strdup(const char * str)
{
    assert(str != NULL);

    char * str_t = (char *)malloc(strlen(str) + 1);
    if (str_t == NULL)
    {
        printf("malloc failed\n");
        exit(1);
    }

    strcpy(str_t, str);             //进行拷贝

    return str_t;                   //返回的内存在堆中需要手动释放内存
}

int main()
{
    char *str1 = "abcde";
    char *str2 = NULL;

    str2 = my_strdup(str1);
    printf("%s\n", str2);
    free(str2);  //释放内存

    return 0;
}
```

12、strstr函数

1. **原型**：`char * my_strstr(const char * haystack, char * needle)`
2. **作用**：查找str2在str1中出现的位置，找到返回位置，否则返回NULL
3. **源码**：

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>

char * my_strstr(const char * haystack, char * needle)
{
    assert(haystack != NULL & needle != NULL);

    int len1 = strlen(haystack);
    int len2 = strlen(needle);

    while(len1 >= len2)
    {
        len1--;
        if(strncmp(haystack, needle, len2) == 0)
        {
            return haystack;
        }
        haystack++;
    }

    return NULL;
}

int main()
{
    char *str1 = "China is a nation, a great nation!";
    char *str2 = "nation";
    char *p = NULL;

    p = my_strstr(str1, str2);
    printf("The string is: %s\n", p);

    return 0;
}
```

13、strpbrk函数

1. **原型**：`char *strpbrk(const char *str1, const char *str2)`
2. **作用**：从str1的第一个字符向后检索，直到'\0'，如果当前字符存在于str2中，那么返回当前字符的地址，并停止检索.
3. **源码**：

```
#include <stdio.h>
#include <stdlib.h>
```

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

/*strpbrk是在源字符串（s1）中找出最先含有搜索字符串（s2）中任一字符的位置并返回，若找不到则返回空指针。*/
char *my_strpbrk(const char *s, const char *accept)
{
    assert((s != NULL) && (accept != NULL));

    const char *t;

    while (*s != '\0')
    {
        t = accept;
        while (*t != '\0')
        {
            if (*s == *t)
            {
                return (char *) s;
            }
            ++t;
        }

        ++s;
    }

    return NULL;
}

int main()
{
    char s1[] = "www.unigress.com";
    char s2[] = "esu";
    char *p = my_strpbrk(s1, s2);
    if(p != NULL)
    {
        printf("The result is: %s\n",p);
    }
    else
    {
        printf("Sorry!\n");
    }
    return 0;
}
```

14、strspn函数

1. **原型**：`size_t strspn(const char *s, const char *accept);`
2. **作用**：检索字符串 **s** 中第一个不在字符串 **accpet** 中出现的字符下标。
3. 源码:

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

```c
/*检索字符串 s 中第一个不在字符串 accept 中出现的字符下标。*/
int my_strspn(const char *s, const char *accept)
{
    assert((s != NULL) && (accept != NULL));

    const char *t;
    const char *o_s = s;
    int count = 0;

    while (*s != '\0')
    {
        t = accept;
        while (*t != '\0')
        {
            if (*s == *t)
            {
                break;
            }
            ++t;
        }

        if (*t == '\0')
        {
            return s - o_s;
        }

        s++;
    }

    return 0;
}

int main ()
{
   int len;
   const char str1[] = "ss1ABCDEFGAA019874";
   const char str2[] = "s1BA";

   len = my_strspn(str1, str2);
   printf("str1 %d\n", len );

   len = strspn(str1, str2);
   printf("str1 %d\n", len );

   return 0;
}
```

15、strcspn函数

1. **原型**：`size_t strcspn(const char *s, const char *reject);`
2. **作用**：从参数s 字符串的开头计算连续的字符，而这些字符都完全不在参数reject 所指的字符串中。

若strcspn()返回的数值为n，则代表字符串s 开头连续有n 个字符都不含字符串reject 内的字符。
返回值：返回字符串s 开头连续不含字符串reject 内的字符数目。

3. **源码:**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>

size_t my_strcspn(const char *str1, const char *reject)
{
    assert((str1 != NULL) && (reject != NULL));
    const char *t;
    const char *s = str1;

    while (*s != '\0')
    {
        t = reject;
        while (*t != '\0')
        {
            if (*s == *t)
            {
                return s - str1;
            }
            ++t;
        }
        ++s;
    }

    return 0;
}

int main()
{
   int len;
   const char str1[] = "ABCDEFGHI123456780";
   const char str2[] = "EF";

   len = my_strcspn(str1, str2);
   printf("%d\n", len);

   len = strcspn(str1, str2);
   printf("%d\n", len);

   return(0);
}
```

16、strset 与 strnset函数

1. 原型：`char *strset(char *str, int ch)`
2.

```
#include <stdio.h>
#include <assert.h>
```

```
//功能：strset把字符串s中的所有字符都设置成字符ch的函数。
char *strset(char *str, int ch)
{
    assert(str != NULL);
    char *s = str;

    while (*s != '\0')
    {
        *s = (char)ch;
        s++;
    }

    return str;
}

//功能：strnset将一个字符串中的前n个字符都设为指定字符ch的函数。
char *strnset(char *str, int ch, int n)
{
    assert(str != NULL);
    char *s = str;

    while (*s != '\0' && s - str < n)
    {
        *s = (char)ch;
        ++s;
    }

    return str;
}


int main()
{
    char dest[25] = "Helloworld";

    strnset(dest, 'a', 5);
    printf("%s\n", dest);
    strset(dest, 'a');
    printf("%s\n", dest);

    return 0;
}
```

17、strtok函数

1. **原型**：`char *strtok(char *s, const char *delim)`
2. **作用**：
3. **源码:**

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

/* Parse S into tokens separated by characters in DELIM.
   If S is NULL, the saved pointer in SAVE_PTR is used as
   the next starting point.  For example:
```

```
        char s[] = "-abc-=-def";
        char *sp;
        x = strtok_r(s, "-", &sp);      // x = "abc", sp = "=-def"
        x = strtok_r(NULL, "-=", &sp);  // x = "def", sp = NULL
        x = strtok_r(NULL, "=", &sp);   // x = NULL
                // s = "abc\0-def\0"
*/
char *my_strtok_r(char *s, const char *delim, char **save_ptr)
{
    char *token;

    if (s == NULL)
    {
        s = *save_ptr;
    }

    /* Scan leading delimiters.  */
    s += strspn(s, delim);
    if (*s == '\0')
    {
        return NULL;
    }

    /* Find the end of the token.  */
    token = s;
    s = strpbrk(token, delim);
    if (s == NULL)
    {
        /* This token finishes the string.  */
        *save_ptr = strchr(token, '\0');
    }
    else
    {
        /* Terminate the token and make *SAVE_PTR point past it.  */
        *s = '\0';
        *save_ptr = s + 1;
    }

    return token;
}

char *my_strtok(char *s, const char *delim)
{
    static char *last;

    return my_strtok_r(s, delim, &last);
}

int main(void)
{
    char str[80] = "I :love :China!";
    const char s[] = ":";
    char *token;

    /* 获取第一个子字符串 */
    token = strtok(str, s);
    /* 继续获取其他的子字符串 */
    while( token != NULL )
```

```
    {
        printf( "%s\n", token);
        token = strtok(NULL, s);
    }

    strcpy(str,  "I :love :China!");

    /* 获取第一个子字符串 */
    token = my_strtok(str, s);
    /* 继续获取其他的子字符串 */
    while( token != NULL )
    {
        printf( "%s\n", token);
        token = my_strtok(NULL, s);
    }

    return 0;
}
```

18、strupr与strlwr函数

1. 原型：`char *strupr(char *str)` `char *strlwr(char *str)`
2. 作用：这两个函数讲一个字符串做大、小写转换

   单个字符大小写转换见下面函数

```
#include <ctype.h>

int toupper(int c);

int tolower(int c);
```

3. 源码：

```
#include <stdio.h>
#include <string.h>
#include <assert.h>
#include <ctype.h>

char *strupr(char *str)
{
    assert(str != NULL);
    char *s = str;

    while (*s != '\0')
    {
#if 0
        if (*s >= 'a' && *s <= 'z')
        {
            *s += 'A' - 'a';
        }
#endif
        *s = toupper(*s);
        s++;
    }
```

```
    return str;
}

char *strlwr(char *str)
{
    assert(str != NULL);
    char *s = str;

    while (*s != '\0')
    {
#if 0
        if (*s >= 'A' && *s <= 'Z')
        {
            *s += 'a' - 'A';
        }
#endif
        *s = tolower(*s);
        s++;
    }

    return str;
}

int main()
{
    char a[] = "HELLOworld!!!";

    printf("%s\n", a);

        strlwr(a);
    printf("%s\n", a);

    strupr(a);
    printf("%s\n", a);

    return 0;
}
```

19、memcpy与memccpy函数

1. 原型：`void *memcpy(void *dest, const void *src, size_t n)`
2. 作用：
3. 源码：

```
#include <stdio.h>
#include <assert.h>

void *my_memcpy(void *dest, const void *src, size_t n)
{
    assert((dest != NULL) && (src != NULL));
    void *o_dest = dest;
    char *dest_c  = (char *)dest;
    const char *src_c   = (const char *)src;

    while (n--)
    {
```

```
        *dest_c++ = *src_c++;
    }

    return o_dest;
}

void print_array(int *a, int n)
{
    int i;
    for (i = 0; i < n; i++)
    {
        printf(" %d ", a[i]);
    }
    printf("\n");
}

int main()
{
    int a[10] = {1, 2, 3, 4, 5, 66, 777, 8888, 99999, 123456789};
    int b[10];

    my_memcpy(b, a, sizeof(a));
    print_array(a, 10);
    print_array(b, 10);

    return 0;
}
```

原型：`void *memccpy(void *dest, const void *src, int c, size_t n)`

头文件：#include <string.h>

1. 功能：由src所指内存区域复制不多于count个字节到dest所指内存区域，如果遇到字符ch则停止复制。
2. 说明：返回指向字符ch后的第一个字符的指针，如果src前n个字节中不存在ch则返回NULL。ch被复制。

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

void *my_memccpy(void *dest, const void *src, int c, size_t n)
{
    assert((dest != NULL) && (src != NULL));

    char *dest_c = (char *)dest;
    const char *src_c = (const char *)src;

    while (n--)
    {
        *dest_c = *src_c;
        if (*src_c == (char)c)
        {
            return dest_c + 1;
        }

        dest_c++;
```

```
        src_c++;
    }

    return NULL;
}

int main()
{
    const char *src = "Taiyuan Unigress";
    char dest[20];
    char *p;

    p = (char*)(my_memccpy(dest, src, 'i', strlen(src)));
    if (p != NULL)
    {
        *p = '\0';
        printf("%s\n", dest);
    }else
    {
        printf("char %c is not found\n", 'i');
    }

    return 0;
}
```

20、memchr函数

1. 原型：`void *memchr(const void *str, int c, size_t n)`
2. 功能：在参数 **str** 所指向的字符串的前 **n** 个字节中搜索第一次出现字符 **c**（一个无符号字符）的位置。

> 参数
>
> - **str** -- 指向要执行搜索的内存块。
> - **c** -- 以 int 形式传递的值，但是函数在每次字节搜索时是使用该值的无符号字符形式。
> - **n** -- 要被分析的字节数。
>
> 返回值
>
> 该函数返回一个指向匹配字节的指针，如果在给定的内存区域未出现字符，则返回 NULL。

使用：

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

int main ()
{
   const char str[] = "http://edu.unigress.com";
   const char ch = '.';
   char *p;

   p = (char*)memchr(str, ch, strlen(str));

   printf("|%c| 之后的字符串是 - |%s|\n", ch, p);
```

```
    return(0);
}
```

`void *memrchr(const void *s, int c, size_t n);`

此函数与memchr()函数的功能类似，但是反向搜索

注意：此函数不能在老版gcc下使用，但是可以在g++下使用。

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

int main ()
{
   const char str[] = "http://edu.unigress.com";
   const char ch = '.';
   char *p;

   p = (char*)memrchr(str, ch, strlen(str));
   printf("|%c| 之后的字符串是 - |%s|\n", ch, p);

   return(0);
}
```

21、memcmp函数

1. 原型 `int memcmp(const void *str1, const void *str2, size_t n)`
2. 把存储区 **str1** 和存储区 **str2** 的前 **n** 个字节进行比较。

   参数

   - **str1** -- 指向内存块的指针。
   - **str2** -- 指向内存块的指针。
   - **n** -- 要被比较的字节数。

     返回值

- 如果返回值 < 0，则表示 str1 小于 str2。
- 如果返回值 > 0，则表示 str2 小于 str1。
- 如果返回值 = 0，则表示 str1 等于 str2。

3. 代码

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

int my_memcmp(const void *s1, const void *s2, size_t n)
{
    assert((s1 != NULL) && (s2 != NULL));
    const char *s1_c = s1;
    const char *s2_c = s2;

    while (n--)
```

```
    {
        if (*s1_c != *s2_c)
        {
            break;
        }
        s1_c++;
        s2_c++;
    }

    return  *s1_c - *s2_c;
}

int main ()
{
   char str1[15];
   char str2[15];
   int ret;

   memcpy(str1, "123456", 6);
   memcpy(str2, "123455", 6);

   ret = memcmp(str1, str2, 5);
   if(ret > 0)
   {
      printf("str2 小于 str1\n");
   }
   else if(ret < 0)
   {
      printf("str1 小于 str2\n");
   }
   else
   {
      printf("str1 等于 str2\n");
   }

   return(0);
}
```

strcmp与memcpy区别

对于memcmp()，如果两个字符串相同而且count大于字符串长度的话，memcmp不会在\0处停下来，会继续比较\0后面的内存单元，直到不相等或者达到count次数。

对于strncmp()比较必定会在最短的字符串的末尾停下来，即使count还未为零。具体的例子：

char a1[]="ABCD";

char a2[]="ABCD";

对于memcmp(a1, a2, 10)，memcmp在两个字符串的\0之后继续比较

对于strncmp(a1, a2, 10），strncmp在两个字符串的末尾停下，不再继续比较。

所以，如果想使用memcmp比较字符串，要保证count不能超过最短字符串的长度，否则结果有可能是错误的。

strncmp("abcd", "abcdef", 6) = 0。比较次数是一样的：

memcmp:在比较到第5个字符也就是'\0'，*su1 - *su2的结果显然不等于0，所以满足条件跳出循环，不会再进行后面的比较。我想在其他情况下也一样。

strncmp:同样的道理再比较到第5个字符时结束循环，其实strncmp中会判断字符串结束，其作用仅在于当两个字符串相同的情形下，防止多余的比较次数。

22、memmove函数

函数作用：用于从src拷贝count个字节到dest，如果目标区域和源区域有重叠的话，memmove能够保证源串在被覆盖之前将重叠区域区域的字节拷贝到目标区域中，但复制后src内容会被更改，但是当目标区域与源区域没有重叠则和memcpy函数功能相同。

memmove可以被分为三种情况：

1.dest < src，源内存的首地址大于目标内存的首地址，进行正向拷贝，此时无需担心dest + n>src，超出的部分只会将src已经拷贝过的部分覆盖，不会造成影响

2.dest >= src + n，此时目标内存的首地址大于源内存的首地址 +n，进行反向拷贝，由于两首地址相距足够大，无需担心被覆盖问题

3.dest < src + n，此时目标内存的首地址小于源内存的首地址+n，此时随着src向dest拷贝，src前段会将末端的覆盖掉，导致数据丢失，此时我们需要手动“扩容”，将两个指针同时增加n - 1

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

void *my_memmove(void *dest, void *src, size_t n)
{
    if (dest == NULL || src == NULL)
    {
        return NULL;
    }

    char *dest_c = (char *)dest;
    char *src_c = (char *)src;

    if (dest_c < src_c || (char *)dest_c >= (char *)src_c + n )
    {
        while (n--)
        {
            *dest_c++ = *src_c++;
        }
    }
    else
    {
        dest_c = dest_c + n - 1;
        src_c = src_c + n - 1;
        while (n--)
        {
            *dest_c-- = *src_c--;
        }
    }

    return dest;
```

```
}

int main()
{
    char src[] = "hello";
    char dest[] = "world";

    my_memmove(dest, src, strlen(src));
    printf("%s\n", dest);

    return 0;
}
```

23、memset函数

原型：`void *memset(void *s, int c, size_t n);`

作用：复制字符 c（一个无符号字符）到参数 str 所指向的字符串的前 n 个字符。

源码：

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

void *my_memset(void *s, int c, size_t n)
{
    assert(s != NULL);
    char  *s_c = (char *)s;

    while (n--)
    {
        *s_c++ = (char)c;
    }

    return s;
}

int main ()
{
   char str[50];

   strcpy(str,"This is string.h library function");
   puts(str);

   my_memset(str, 'A', 5);
   puts(str);

   return(0);
}
```

另外：

```
#include <strings.h>

void bzero(void *s,  size_t n);
```

bzero的实现：

```
memeset(s, 0, n);
```