字符函数和字符串函数

本章重点

> 重点介绍处理字符和字符串的库函数的使用和注意事项

- 求字符串长度
  - strlen
- 长度不受限制的字符串函数
  - strcpy
  - strcat
  - strcmp
- 长度受限制的字符串函数介绍
  - strncpy
  - strncat
  - strncmp
- 字符串查找
  - strstr
  - strtok
- 错误信息报告
  - strerror
- 字符操作
- 内存操作函数
  - memcpy
  - memmove
  - memset
  - memcmp

前言

> C语言中对字符和字符串的处理很是频繁，但是C语言本身是没有字符串类型的，字符串通常放在`常量字符串`中或者`字符数组`中。`字符串常量`适用于那些对它不做修改的字符串函数.

函数介绍

strlen

```
size_t strlen ( const char * str );
```

- 字符串已经`'\0'`作为结束标志，strlen函数返回的是在字符串中`'\0'`前面出现的字符个数（不包含`'\0'`）。
- 参数指向的字符串必须要以`'\0'`结束。
- 注意函数的返回值为size_t，是无符号的（`易错`）
- 学会strlen函数的模拟实现

注:

```
#include <stdio.h>
int main()
{
    const char*str1 = "abcdef";
    const char*str2 = "bbb";
    if(strlen(str2)-strlen(str1)>0)
    {
        printf("str2>str1\n");
    }
    else
    {
        printf("srt1>str2\n");
    }
    return 0;
}
```

strcpy

```
char* strcpy(char * destination, const char * source );
```

- Copies the C string pointed by source into the array pointed by destination, including the terminating null character (and stopping at that point).
- 源字符串必须以`'\0'`结束。
- 会将源字符串中的`'\0'`拷贝到目标空间。
- 目标空间必须足够大，以确保能存放源字符串。
- 目标空间必须可变。
- 学会模拟实现。

strcat

```
char * strcat ( char * destination, const char * source );
```

- Appends a copy of the source string to the destination string. The terminating null character in destination is overwritten by the first character of source, and a null-character is included at the end of the new string formed by the concatenation of both in destination.
- 源字符串必须以`'\0'`结束。
- 目标空间必须有足够的大，能容纳下源字符串的内容。
- 目标空间必须可修改。
- 字符串自己给自己追加，如何？

strcmp

```
int strcmp ( const char * str1, const char * str2 );
```

- This function starts comparing the first character of each string. If they are equal to each other, it continues with the following pairs until the characters differ or until a terminating null-character is reached.

- 标准规定：
  - 第一个字符串大于第二个字符串，则返回大于0的数字
  - 第一个字符串等于第二个字符串，则返回0
  - 第一个字符串小于第二个字符串，则返回小于0的数字
  - `那么如何判断两个字符串？`

strncpy

```
char * strncpy ( char * destination, const char * source, size_t num );
```

- Copies the first num characters of source to destination. If the end of the source C string (which is signaled by a null-character) is found before num characters have been copied, destination is padded with zeros until a total of num characters have been written to it.
- 拷贝num个字符从源字符串到目标空间。
- 如果源字符串的长度小于num，则拷贝完源字符串之后，在目标的后边追加0，直到num个。

strncat

```
char * strncat ( char * destination, const char * source, size_t num );
```

- Appends the first num characters of source to destination, plus a terminating null-character.
- If the length of the C string in source is less than num, only the content up to the terminating null-character is copied.

```
/* strncat example */
#include <stdio.h>
#include <string.h>

int main ()
{
  char str1[20];
  char str2[20];
  strcpy (str1,"To be ");
  strcpy (str2,"or not to be");
  strncat (str1, str2, 6);
  puts (str1);
  return 0;
}
```

strncmp

```
int strncmp ( const char * str1, const char * str2, size_t num );
```

- 比较到出现另个字符不一样或者一个字符串结束或者num个字符全部比较完。

Return Value

Returns an integral value indicating the relationship between the strings:

return value	indicates
<0	the first character that does not match has a lower value in *str1* than in *str2*
0	the contents of both strings are equal
>0	the first character that does not match has a greater value in *str1* than in *str2*

```
/* strncmp example */
#include <stdio.h>
#include <string.h>

int main ()
{
  char str[][5] = { "R2D2" , "C3PO" , "R2A6" };
  int n;
  puts ("Looking for R2 astromech droids...");
  for (n=0 ; n<3 ; n++)
  if (strncmp (str[n],"R2xx",2) == 0)
  {
    printf ("found %s\n",str[n]);
  }
  return 0;
}
```

[strstr]

```
char * strstr ( const char *, const char * );
```

- Returns a pointer to the first occurrence of str2 in str1, or a null pointer if str2 is not part of str1.

```
/* strstr example */
#include <stdio.h>
#include <string.h>

int main ()
{
  char str[] ="This is a simple string";
  char * pch;
  pch = strstr (str,"simple");
  strncpy (pch,"sample",6);
  puts (str);
  return 0;
}
```

[strtok]

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

- sep参数是个字符串，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。

```
/* strtok example */
#include <stdio.h>
#include <string.h>

int main ()
{
  char str[] ="- This, a sample string.";
  char * pch;
  printf ("Splitting string \"%s\" into tokens:\n",str);
  pch = strtok (str," ,.-");
  while (pch != NULL)
  {
    printf ("%s\n",pch);
    pch = strtok (NULL, " ,.-");
  }
  return 0;
}
```

```
#include <stdio.h>
int main()
{
   char *p = "zhangpengwei@bitedu.tech";
    const char* sep = ".@";
    char arr[30];
    char *str = NULL;
    strcpy(arr, p);//将数据拷贝一份，处理arr数组的内容
    for(str=strtok(arr, sep); str != NULL; str=strtok(NULL, sep))
    {
        printf("%s\n", str);
    }
}
```

strerror

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

> 返回错误码，所对应的错误信息。

```
/* strerror example : error list */
#include <stdio.h>
#include <string.h>
#include <errno.h>//必须包含的头文件
```

```
int main ()
{
  FILE * pFile;
  pFile = fopen ("unexist.ent","r");
  if (pFile == NULL)
    printf ("Error opening file unexist.ent: %s\n",strerror(errno));
    //errno: Last error number
  return 0;
}
Edit & Run
```

字符分类函数：

函数	如果他的参数符合下列条件就返回真
iscntrl	任何控制字符
isspace	空白字符：空格‘ ’，换页‘\f’，换行'\n'，回车‘\r’，制表符'\t'或者垂直制表符'\v'
isdigit	十进制数字 0~9
isxdigit	十六进制数字，包括所有十进制数字，小写字母a~f，大写字母A~F
islower	小写字母a~z
isupper	大写字母A~Z
isalpha	字母a~z或A~Z
isalnum	字母或者数字，a~z,A~Z,0~9
ispunct	标点符号，任何不属于数字或者字母的图形字符（可打印）
isgraph	任何图形字符
isprint	任何可打印字符，包括图形字符和空白字符

字符转换：

```
int tolower ( int c );
int toupper ( int c );
```

```
/* isupper example */
#include <stdio.h>
#include <ctype.h>
int main ()
{
  int i=0;
  char str[]="Test String.\n";
  char c;
  while (str[i])
  {
    c=str[i];
```

```
    if (isupper(c))
        c=tolower(c);
    putchar (c);
    i++;
  }
  return 0;
}
```

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。

```
/* memcpy example */
#include <stdio.h>
#include <string.h>

struct {
  char name[40];
  int age;
} person, person_copy;

int main ()
{
  char myname[] = "Pierre de Fermat";

  /* using memcpy to copy string: */
  memcpy ( person.name, myname, strlen(myname)+1 );
  person.age = 46;

  /* using memcpy to copy structure: */
  memcpy ( &person_copy, &person, sizeof(person) );

  printf ("person_copy: %s, %d \n", person_copy.name, person_copy.age );

  return 0;
}
```

memmove

```
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。

```
/* memmove example */
#include <stdio.h>
#include <string.h>

int main ()
{
  char str[] = "memmove can be very useful.....";
  memmove (str+20,str+15,11);
  puts (str);
  return 0;
}
```

[memcmp]

```
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
- 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 *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)

```
/* 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;
}
```

库函数的模拟实现

模拟实现strlen

三种方式： 方式1：

```
//计数器方式
int my_strlen(const char * str)
{
        int count = 0;
        while(*str)
        {
                count++;
                str++;
        }
        return count;
}
```

方式2：

```
//不能创建临时变量计数器
int my_strlen(const char * str)
{
        if(*str == '\0')
                return 0;
        else
                return 1+my_strlen(str+1);
}
```

方式3：

```
//指针-指针的方式
int my_strlen(char *s)
{
       char *p = s;
       while(*p != '\0' )
              p++;
       return p-s;
}
```

模拟实现strcpy

参考代码：

```
//1.参数顺序
//2.函数的功能，停止条件
//3.assert
//4.const修饰指针
//5.函数返回值
//6.题目出自《高质量C/C++编程》书籍最后的试题部分
char *my_strcpy(char *dest, const char*src)
{
        char *ret = dest;
        assert(dest != NULL);
```

```
        assert(src != NULL);

        while((*dest++ = *src++))
        {
                ;
        }
        return ret;
}
```

模拟实现strcat

参考代码：

```
char *my_strcat(char *dest, const char*src)
{
        char *ret = dest;
        assert(dest != NULL);
        assert(src != NULL);
        while(*dest)
        {
                dest++;
        }
        while((*dest++ = *src++))
        {
                ;
        }
        return ret;
}
```

模拟实现strstr

注：让他们下去自己研究一下KMP算法。

```
char *my_strstr(const char* str1, const char* str2 )
{
        assert(str1);
        assert(str2);

        char *cp = (char*)str1;
        char *substr = (char *)str2;
        char *s1 = NULL;

        if(*str2 == '\0')
                return NULL;

        while(*cp)
        {
                s1 = cp;
                substr = str2;
                while(*s1 && *substr && (*s1 == *substr))
                {
                        s1++;
                        substr++;
```

```
                }
                if(*substr == '\0')
                        return cp;
                cp++;
        }
}
```

模拟实现strcmp

参考代码：

```
int my_strcmp (const char * src, const char * dst)
{
        int ret = 0 ;
                assert(src != NULL);
        assert(dest != NULL);
        while( ! (ret = *(unsigned char *)src - *(unsigned char *)dst) && *dst)
                ++src, ++dst;

        if ( ret < 0 )
                ret = -1 ;
        else if ( ret > 0 )
                ret = 1 ;

        return( ret );
}
```

模拟实现memcpy

参考代码：

```
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);
}
```

模拟实现memmove

参考代码：

```
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);
}
```

本章完