C语言中常用的字符串操作（子串分割、替换、去前后空格、递归实现字符串反转）

在C语言中，并没有像java中对字符串操作的封装好的函数，在C语言中，都需要自己根据C语言函数库来实现常用的字符串操作

一、字符串的分割，根据子串进行分割

```
#include<stdio.h>
#include<string.h>
#include<iostream>
using namespace std;

//字符串的分割，根据子串分割字符串
//参数：
//str被分割的字符串
//sub子串
//sub_before为str中sub所在位置的之前部分
//sub_after为str中sub所在位置的之后部分，包括sub
void str_split(const char* str/*in*/,char* sub/*in*/,char** sub_before/*out*/,char** sub_after/*out*/){
  if ( str == NULL || *str=='\0' || sub==NULL){
        printf("function str_split error:( str == NULL || *str=='\0' || sub==NULL)");
        return;
  }
  if (*sub_before == NULL){
        *sub_before = (char*)malloc(strlen(str));
  }
  if (*sub_after == NULL){
        *sub_after = (char*)malloc(strlen(str));
  }
  //分割字符串的时候使用strstr函数，返回是第一个sub地址,如果没有匹配的则返回NULL
  char* temp = (char*)malloc(strlen(str));
```

```
    strcpy(temp,str);
    *sub_after = strstr(temp, sub);
    //没有匹配的子串
    if (*sub_after == NULL || **sub_after == '\0'){
            printf("the str without sub");
            return;
    }
    //添加结束标志符
    *(*sub_after + strlen(*sub_after)) = '\0';
    //将字符串中sub之前的字符赋值给sub_before
    int i = 0;
    while (i < (strlen(temp)-strlen(*sub_after))){
            *(*sub_before + i) = *(temp+i);
            i++;
    }
    //添加结束标志符,添加结束标志的目的是为了防止乱码
    *(*sub_before + i) = '\0';
}
```

二、去字符串的前后空格

```
//传入一个字符串，去除字符串前后的空格
//参数：
//str为需要去除空格的字符串
//newchar为去除空格之后的字符串
void trim_space(const char* str/*in*/,char* newchar/*out*/){
    if (str == NULL || *str == '\0' || newchar == NULL){
            printf("function trim_space error:(str==NULL || *str=='\0' || newchar == NULL)");
            return;
    }
    //利用左右开弓来去除前后的空格
    int i = 0, j = strlen(str);
    //isspace函数是用来判断字符是否为空格
```

```
    //求出,在str字符串前面有多少个空格
    while (isspace(str[i]) || str[i]=='\0')
    {
        i++;
    }
    //求出,在str字符串后面有多少个空格
    while (isspace(str[j]) || str[j]=='\0'){
        j++;
    }
    //从str+i个位置开始将(j-i+1)个字符拷贝到newchar中
    strncpy(newchar,str+i,j-i+1);
}
```

三、递归实现字符串的反转

```
//通过函数的递归调用来实现字符串的反转
//参数:
//str:需要反转的字符串
//inverse_str:反转之后的字符串
void str_inverse(const char* str/*in*/,char* inverse_str/*out*/){
    //遇到异常情况，直接跳出方法
    if (str == NULL || inverse_str==NULL){
        printf("function str_inverse error:(str == NULL || inverse_str==NULL)");
        return;
    }
    //递归的结束，当到str字符串结束的时候
    if (*str == '\0'){
        return;
    }
    str_inverse(str+1,inverse_str);
    strncat(inverse_str,str,1);
}
```

```
void main(){//调用字符串反转
    char* s = "abcdefg";
```

```
    //char result[10] = {0};//这种方式可以调用成功
    //char* result = (char*)malloc(strlen(s));//这种会宕机（出现异常）,原因是因为malloc函数不会初始化导致,因为开辟空间的时候只开辟了
//那么多，而后面使用strcat函数进行拼接，是直接在以前的基础上进行拼接，会出现，前面乱码，但是结果还是正确的
    char* result = (char*)calloc(strlen(s),1);
    str_inverse(s,result);
    cout << result << endl;
}
```

四、字符串的替换（下面的函数使用到了上面的字符串分割函数str_split）

```
//字符串的替换
//参数：
//str:原字符串
//replace_str:要被替换的字符串
//replace_become：要替换成什么字符串
//replace_after:替换之后的字符串
void str_replace(const char* str/*in*/,char* replace_str/*in*/,char* replace_become/*in*/,char** replace_after/*out*/){
    //遇到异常情况直接退出
    if (str == NULL || *str == '\0' || replace_str == NULL || replace_str == '\0' || replace_become == NULL || *replace_become=='\0'){
        printf("function str_replace error:(str==NULL || *str =='\0' || replace_str==NULL || replace_str=='\0')");
        return;
    }
    const char* flag = strstr(str,replace_str);
    //字符串str中没有子串
    if (flag==NULL){
        printf("the str without replace_str");
        return;
    }
```

```
    //为结果开辟空间
    *replace_after = (char*)calloc(strlen(str)+strlen(replace_become)-strlen(replace_str),1);
    //通过字符串的分割之后进行拼接
    char *after = NULL, *before=NULL;
    while (flag){
        if (after == NULL){
            str_split(str, replace_str, &before, &after);
        }
        else{
            str_split(after, replace_str, &before, &after);
        }
        strcat(*replace_after,before);
        strcat(*replace_after,replace_become);
        after = after + strlen(replace_str);
        flag = strstr(after,replace_str);
    }
    if (after != '\0'){
        strcat(*replace_after,after);
    }
}
```

要想掌握好c语言中的指针，多做一下c语言中的字符串操作是最好的办法。

C语言模拟实现字符串操作函数

在编写程序过程中，我们经常使用到一些字符串函数，例如求字符串长度，拷贝字符串……，这些函数都在C标准库中存在，我们可以直接使用。但我们还需要掌握这些函数的实现方法，今天来看看一些常用的字符串操作函数的实现方法

1. strlen

strlen是用来求字符串长度的函数，字符串长度就是字符串中包含的字符的个数，但是不包含字符串结尾的 ‘\0’

实现strlen有三种方法：

（1）定义一个计数器

```
size_t mystrlen(const char* str)
{
    size_t count = 0;
    while (*str != '\0')
    {
        count++;
        str++;
    }
    return count;
}
```

（2）递归版本

```
size_t my_strlen(const char *str)
{
   if (*str == '\0')
      return 0;
   else
      return my_strlen(str + 1) + 1;
}
```

（3）利用指针 - 指针

```
size_t mystrlen(const char* str)
{
    const char* end = str;
    while (*end++)
        ;
    return end - str - 1;
}
```

用于复制字符串的函数是strcpy，它的原型如下：

```
char* strcpy ( char* dest, const char* src );
```

使用这个函数时，要注意几点

（1）目标字符数组的空间必须足够大，足以容纳需要复制的字符串

（2）目标字符数组要可以被修改

（3）被复制的字符串要可以找到’ \ 0’

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

```
    char *res = dest;
    assert(dest);
    assert(src);
    while (*dest++ = *src++) //注意这里是一个等号
        ;
    return res;
}
```

3.strcat

strcat函数是可以把一个字符串添加(连接)到另一个字符串的后面。strcat函数要求dest参数原先已经包含了一个字符串(可以是空字符串)。它找到这个字符串的末尾，并把src字符串的一份拷贝添加到这个位置。

```
char *mystrcat(char *dest, const char *src)
{
    char *res = dest;
    assert(dest);
    assert(src);
    while (*dest != '\0')
        dest++;
    while (*dest++ = *src) //这里同样也是一个等号
        ;
    return res;
}
```

4.strcmp

strcmp用于比较两个字符串，及对两个字符串对应的字符逐个进行比较，直到发现不匹配。那个最先不匹配的字符中较“小”的那个字符所在的字符串被认为“小于”另外一个字符串。如果其中一个字符串是另外一个字符串的前面一部分，那么它也被认为“小于”另外一个字符串，因为它的’\0’出现的更早。

```
int my_strcmp(const char* src1, const char* src2)
{
  while (*src1 == *src2)
  {
    if (*src1 == '\0')
      return 0;
    src1 ++;
    src2 ++;
  }
```

```
    return *src1 - *src2;
}
```

5.strstr

为了在一个字符串中查找一个子串，可以使用strstr函数，该函数是在s1中查找整个s2第1次出现的起始位置，并返回一个指向该位置的指针。如果s2并没有出现在s1的任何地方，函数将返回一个NULL指针。如果第二个函数是一个空字符串，函数就返回s1。

```
char* my_strstr(char* s1, const char* s2)
{
    char* p = s1;
    const char* q = s2;
    char* cur = NULL;

    assert(s1);
    assert(s2);
    if (*s2 == '\0')
        return s1;
    while (*p != '\0')
    {
        cur = p;
        while ((*p != '\0') && (*q != '\0') && (*p == *q))
        {
            p++;
            q++;
        }
        if (*q == '\0')
            return cur;
        p = cur + 1;
        q = s2;
    }
    return NULL;
}
```

6.strchr

strchr是用来查找一个特定的字符，在字符串str中查找字符ch第一次出现的位置，找到后函数返回一个指向该位置的指针。如果该字符并不存在于字符串中，函数就返回一个NULL指针

```
char* my_strchr(const char* str, char ch)
{
    const char* tmp = str;
    while (*tmp)
    {
        if (*tmp == ch)
            return tmp;
        tmp++;
    }
    return NULL;
```

7.strrchr

与strchr类似的查找函数还有一个是strrchr，它和strchr的不同之处在于，该函数返回的是一个指向字符串中该字符最后一次出现的位置

```
char* my_strrchr(const char* str, int ch)
{
    char* pos = 0;
    assert(str);
    while (*str)
    {
        if (*str == ch)
        {
            pos = str;
        }
        str++;
    }
    if (pos != 0)
    {
        return pos;
    }
    else
        return NULL;
}
```

长度受限制的字符串函数

标准库中还包含一些函数，它们以一种不同的方式去处理字符串。这些函数接受一个显示的长度参数，用于限定进行复制或比较的字符数。

1.strncpy

和strcpy一样，strncpy（）函数把源字符串的字符复制到目标空间，但是，它总是正好向dest中拷贝len个字符，如果strlen的（src）的值小于len，dest数组就用额外的’\0’填充到len字节长度。如果strlen的（src）的值大于或等于len，那么只有len个字符被复制到目标寄存器中。

```
char* my_strncpy(char* dest, const char* src, size_t len)
{
    char* res = dest;
    assert(dest);
    assert(src);

    while (len--)
    {
        *res++ = *src++;
    }
    if (*(res) != '\0')
        *res = '\0';
    return dest;
}
```

2.strncat

strncat函数，它从src中最多复制的len个字符到目标数组的后面。

```
char* my_strncat(char* dest, const char* src, size_t len)
{
    char* res = dest;
    assert(dest);
    assert(src);
    while (*dest != '\0')
        dest++;
    while (len--)
    {
        *dest = *src;
        dest++;
        src++;
    }
    return res;
}
```

3.strncmp

strncmp也用于比较两个字符串，但它最多比较len个字节。如果两个字符串在第len个字符之前存在不相等的字符，这个函数就像的strcmp一样停止比较，返回结果。如果两个字符串的前len个字符相等，函数就返回零。

```
int my_strncmp(const char* s1, const char* s2, size_t len)
{
    assert(s1);
    assert(s2);
    while (len--)
    {
        if (*s1 == *s2)
        {
            s1++;
            s2++;
        }
        else
            return *s1 - *s2;
    }
    return 0;
}
```

标准库里的字符串函数还有很多，今天就先介绍到这里。