# 转载：http://blog.csdn.net/pipisorry/article/details/36898405

[http://blog.csdn.net/pipisorry/article/details/36898405](http://blog.csdn.net/pipisorry/article/details/36898405" \t "_blank)

使用C语言字符串处理的库函数，务必#include <string.h>

**1、 比较字符串大小函数**

 1） 忽略大小写---strcasecmp

函数原型： int strcasecmp (const char \*s1, const char \*s2);

函数说明： 用来比较参数s1和s2字符串，比较时会自动忽略大小写的差异

2）忽略大小写—stricmp

函数原型：int stricmp(char \*str1, char \*str2);

函数说明：以大小写不敏感方式比较两个串

3） 不忽略大小写—strcmp

函数原型：int strcmp(char\*str1,char\*str2);

函数说明：通过比较字串中各个字符的ASCII码，来比较参数Str1和Str2字符串，比较时考虑字符的大小写。

当str1<str2时，返回值<0 当str1=str2时，返回值=0

4） 比较一部分—strncmpi

函数原型：int strncmpi(char \*str1, char \*str2, unsigned maxlen);

函数说明：比较字符串str1和str2的前maxlen个字符

5）内存区域比较---memcmp

函数原型：int memcmp(void\*buf1,void \*buf2,unsigned int count)

函数说明：比较内存区域buf1和buf2的前count个字节。Void\*是指任何类型的指针。

6）内存区域部分比较-- memicmp Void\*是指任何类型的指针。

函数原型：int memicmp(void\*buf1,void \*buf2,unsigned int count)

函数说明：比较内存区域buf1和buf2的前count个字节，但不区分大小写。

以上比较函数的返回值： 若参数1中字符串和参数中2字符串相同则返回0；

            若参数1中字符串长度大于参数2中字符串长度则返回大于0 的值；

            若参数1中字符串 长度小于参数2中字符串 长度则返回小于0的值。

[皮皮blog](http://blog.csdn.net/pipisorry)

**2、 从字符串中提取子串**

**1） 提取子串--strstr**

函数原型：char\* strstr(char\*src,char\*find)

函数说明：从字符串src中寻找find第一次出现的位置（不比较结束符NULL）

返回值：返回指向第一次出现find位置的指针，如果没有找到则返回NULL

**2） 字符串分割，提取分隔符间字串—strtok**

一、用strtok函数进行字符串分割

函数原型：char \*strtok(char\*src， char\*delim);

函数说明：分解字符串诶一组标记串，src为要分解的字符串，delim为分隔符字符串。

            首次调用时，src必须指向要分解的字符串，随后调用要把s设成NULL；

              strtok中src中查找包含在delim中的字符，并用NULL（’\0’）来替换直到找遍整个字符串。

返回值：从s开头开始的一个个被分割的串。当没有被分割的串时则返回NULL。

             所有delim中包含的字符都会被滤掉，并将被滤掉的地方设为一处分割的节点。

举例：

**[cpp]** [view plain](http://blog.csdn.net/pipisorry/article/details/36898405) [copy](http://blog.csdn.net/pipisorry/article/details/36898405)

1. #include<string.h>
2. #include<stdio.h>
3. **int** main(){
4. **char**\*s="Golden Global View";
5. **char**\*d=" ";
6. **char** \*p;
7. p=strtok(s,d);
8. **while**(p){
9. printf("%s\n",p);
10. strtok(NULL,d);
11. }
12. **return** 0;
13. }

输出：Golden

           Global

           View

二、用STL进行字符串的分割

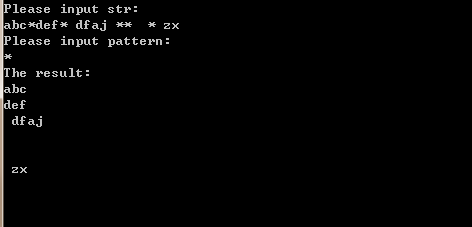
涉及到string类的两个函数find和substr：  
1、find函数  
原型：size\_t find ( const string& str, size\_t pos = 0 ) const;  
功能：查找子字符串第一次出现的位置。  
参数说明：str为子字符串，pos为初始查找位置。  
返回值：找到的话返回第一次出现的位置，否则返回string::npos

2、substr函数  
原型：string substr ( size\_t pos = 0, size\_t n = npos ) const;  
功能：获得子字符串。  
参数说明：pos为起始位置（默认为0），n为结束位置（默认为npos）  
返回值：子字符串

实现如下：

复制代码

1 //字符串分割函数  
 2 std::vector<std::string> split(std::string str,std::string pattern)  
 3 {  
 4 std::string::size\_type pos;  
 5 std::vector<std::string> result;  
 6 str+=pattern;//扩展字符串以方便操作  
 7 int size=str.size();  
 8   
 9 for(int i=0; i<size; i++)  
10 {  
11 pos=str.find(pattern,i);  
12 if(pos<size)  
13 {  
14 std::string s=str.substr(i,pos-i);  
15 result.push\_back(s);  
16 i=pos+pattern.size()-1;  
17 }  
18 }  
19 return result;  
20 }

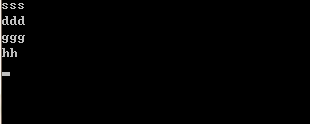


三、用Boost进行字符串的分割

用boost库的正则表达式实现字符串分割  
实现如下：

复制代码

1 std::vector<std::string> split(std::string str,std::string s)  
 2 {  
 3 boost::regex reg(s.c\_str());  
 4 std::vector<std::string> vec;  
 5 boost::sregex\_token\_iterator it(str.begin(),str.end(),reg,-1);  
 6 boost::sregex\_token\_iterator end;  
 7 while(it!=end)  
 8 {  
 9 vec.push\_back(\*it++);  
10 }  
11 return vec;  
12 }



boost里面有自带的split的函数，如果用boost的话，还是直接用split的好，这里就不多说了，代码如下：

复制代码

#include <iostream>

#include <string>

#include <vector>

#include <boost/algorithm/string/classification.hpp>

#include <boost/algorithm/string/split.hpp>

using namespace std;

int main()

{

string s = "sss/ddd,ggg";

vector<string> vStr;

boost::split( vStr, s, boost::is\_any\_of( ",/" ), boost::token\_compress\_on );

for( vector<string>::iterator it = vStr.begin(); it != vStr.end(); ++ it )

cout << \*it << endl;

return 0;

}

[[字符串分割(C++)](http://www.cnblogs.com/MikeZhang/archive/2012/03/24/MySplitFunCPP.html)]

此外还有：strtok函数、strtok\_s函数、strstr函数

[[C/C++ 分割字符串的多种方式](http://blog.csdn.net/mycwq/article/details/14648011)]

[皮皮blog](http://blog.csdn.net/pipisorry)

**3、 字符串复制**

1) 字串复制--strcpy

函数原型：char\*strcpy(char\*dest,char\*src)

函数说明：把src所指由NULL结束的字符串复制到dest所指的数组中。

                  其中，src和dest所致内存区域不可重叠且dest必须有足够的空间来容纳src的字符串。

返回值：返回指向dest的指针。

strcpy 是依据 /0 作为结束判断的，如果 to 的空间不够，则会引起 buffer overflow。strcpy 常规的实现代码如下（来自 OpenBSD 3.9）：

char \* strcpy(char \*to, const char \*from){

       char \*save = to;

       for (; (\*to = \*from) != '/0'; ++from, ++to);

       return(save);

}

但通常，我们的 from 都来源于用户的输入，很可能是非常大的一个字符串，因此 strcpy 不够安全。

利用strncpy替代strcpy来防止缓冲区越界,但是如果还要考虑运行效率的话，也许strlcpy是一个更好的方式。

2. strncpy

原型：extern  char \*strncpy(char   \*dest,   char   \*src,   int   n);

功能：把src所指由NULL结束的字符串的前n个字节复制到dest所指的数组中。

说明：

如果src的前n个字节不含NULL字符，则结果不会以NULL字符结束。

如果src的长度小于n个字节，则以NULL填充dest直到复制完n个字节。src和dest所指内存区域不可以重叠且dest必须有足够的空间来容纳src的字符串。

返回指向dest的指针。

在 ANSI C 中，strcpy 的安全版本是 strncpy,但 strncpy 其行为是很诡异的（不符合我们的通常习惯）。

标准规定 n 并不是 sizeof(s1)，而是要复制的char 的个数。一个最常见的问题，就是 strncpy 并不帮你保证 /0结束。

char buf[8];

strncpy( buf, "abcdefgh", 8 );

看这个程序，buf 将会被 "abcdefgh" 填满，但却没有 /0 结束符了。

另外，如果 s2 的内容比较少，而 n 又比较大的话，strncpy 将会把之间的空间都用 /0 填充。这又出现了一个效率上的问题，如下：

char buf[80];

strncpy( buf, "abcdefgh", 79 );

上面的 strncpy 会填写 79 个 char，而不仅仅是 "abcdefgh" 本身。

strncpy 的标准用法为：（手工写上 /0）

strncpy(path, src, sizeof(path) - 1);

path[sizeof(path) - 1] = '/0';

len = strlen(path);

3. strlcpy

// Copy src to string dst of size siz. At most siz-1 characters will be copied. Always NUL terminates (unless siz == 0).

// Returns strlen(src); if retval >= siz, truncation occurred.

size\_t   strlcpy(char \*dst, const char \*src, size\_t siz);

而使用 strlcpy，就不需要我们去手动负责 /0 了，仅需要把 sizeof(dst) 告之 strlcpy 即可：

strlcpy(path, src, sizeof(path));

len = strlen(path);

if ( len >= sizeof(path) )

       printf("src is truncated.");

并且 strlcpy 传回的是 strlen(str)，因此我们也很方便的可以判断数据是否被截断。

[\* 一点点历史 \*]

strlcpy 并不属于 ANSI C，至今也还不是标准。

strlcpy 来源于 OpenBSD 2.4，之后很多 unix-like 系统的 libc 中都加入了 strlcpy 函数，我个人在FreeBSD、Linux 里面都找到了 strlcpy。（Linux使用的是 glibc，

glibc里面有 strlcpy，则所有的 Linux 版本也都应该有 strlcpy）

但 Windows 下是没有 strlcpy 的，对应的是strncpy和memset函数

memset函数

void \*memset(void \*s,int c,size\_t n)  
作用：将已开辟内存空间 s 的首 n 个字节的值设为值 c。

[[memset函数详细说明]](http://blog.csdn.net/yangsen2016/article/details/1638503)

2） 字串复制--strdup

函数原型：char\* strdup(char\*src)

函数说明：复制字符串src

返回值：返回指向被复制字符串的指针，所需空间有malloc（）分配且可以有free（）释放。

3） 内存空间复制--memcpy

函数原型：void \*memcpy(void \*dest,void \*src,unsigned int count);

函数说明：src和dest 所指内存区域不能重叠；由src所致内存区域复制count个字节到dest所指内存区域中。

返回值：返回指向dest的指针。

[http://blog.csdn.net/tigerjibo/article/details/6412759]

[皮皮blog](http://blog.csdn.net/pipisorry)

**4、字符串连接**

1）接尾连接--strcat

函数原型：char\* strcat(char\*dest,char\*src)

函数说明：把src所指字符串添加到dest结尾处(覆盖dest结尾处的'\0')并添加'\0'

2)部分连接--strncat

函数原型:char\* strncat(char\*dest,char\*src,int n);

函数说明：把src所指字符串的前n个字符添加到dest结尾处（覆盖dest结尾处的’\0’）并添加’’\0’.

返回值：返回指向dest的指针。

[皮皮blog](http://blog.csdn.net/pipisorry)

**5、从字符串中查找字符**

**1）内存区域找字符--memchr**

函数原型：void \*memchr(void\*buf,char ch,usigned count)

函数说明：从buf所指内存区域的前count个字节查找字符ch，当第一次遇到字符ch时停止查找。

返回值：如果找到了，返回指向字符ch的指针；否则返回NULL

**2）字串中找字符--strchr**

函数原型:char\*strchr(char\*src,char ch)

函数说明：查找字符串s中首次出现字符ch的位置

返回值：返回首次出现c的位置的指针，如果s中不存在c则返回NULL

**3）搜所现字符--strcspn**

函数原型：int strcspn（char\*src,char\*find）

函数说明：在字符串src中搜寻find中所出现的字符

返回值：返回第一个出现的字符在src中的下标值，即src中出现而不在find中出现的字串的长度。

举例：

**[cpp]** [view plain](http://blog.csdn.net/pipisorry/article/details/36898405) [copy](http://blog.csdn.net/pipisorry/article/details/36898405)

1. #include<string.h>
2. #include<stdio.h>
3. **int** main(){
4. **char**\*s="Golden Global View";
5. **char**\*r="new";
6. **int** n;
7. n=strcspn(s,r);
8. printf("The first char both in s1 and s2 is: %c",s[n]);
9. **return** 0;
10. }

输出：The first char both in s1 and s2 is :e

**4)匹配任一字符--strpbrk**

函数原型：char\*strpbrk(char\*s1,char\*s2)

函数说明：在字符串S1中寻找字符串S2中任何一个字符相匹配的第一个字符的位置，空字符不包括在内。

返回值：返回指向S1中第一个相匹配的字符的指针，如果没有匹配字符则返回空指针。

举例：

**[cpp]** [view plain](http://blog.csdn.net/pipisorry/article/details/36898405) [copy](http://blog.csdn.net/pipisorry/article/details/36898405)

1. #include<stdio.h>
2. #include<string.h>
3. **int** main(){
4. **char** \*s1="WelcomeTo Beijing";
5. **char**\*s2="BIT";
6. **char** \*p;
7. p=strpbrk(s1,s2);
8. **if**(p)
9. printf("%s\n",p);
10. elseprintf("Not Found!\n");
11. **return** 0;
12. }

输出：To Beijing

[皮皮blog](http://blog.csdn.net/pipisorry)

**6、其他函数**

**1）全部转成大写---strupr**

函数原型：char\*strupr(char\*src)

函数说明：将字符串src转换成大写形式，只转换src中出现的小写字母，不改变其他字符

返回值：返回指向src的指针。

**2）全部转成小写---strlwr**

函数原型：char\*strlwr(char\*src)

函数说明：将字符串src转换成小写形式，只转换src中出现的大写字母，不改变其他字符

返回值：返回指向src的指针。

**3）将字串逆向--strrev**

函数原型：char\*strrev（char\*src）

函数说明：把字符串src的所有字符的顺序颠倒过来（不包括NULL， 原地颠倒）,会改变原来的str,返回的也是改变之后的str

不是标准string库里包含的，是微软自己发明的，类似还有atoi之类(在标准的g++中编译可能不会通过)

只能自己实现：

**[cpp]** [view plain](http://blog.csdn.net/pipisorry/article/details/36898405) [copy](http://blog.csdn.net/pipisorry/article/details/36898405)

1. **char** \* strrev(**char** \*str){                   //字符串反转，不包括'\0'
2. **char** \* head = str,\* tail = str;
3. **while**(\*tail++);
4. tail -= 2;//tail指向最后一个非'\0'字符
6. **char** tmp;
7. **while**(head < tail){
8. tmp= \*head;
9. \*head++ = \*tail;
10. \*tail-- = tmp;
11. }
12. **return** str;
13. }

返回值：返回指向颠倒顺序后的字符串指针

**4)计算字符串长度--strlen**

原型：extern int strlen(char \*s);

用法：#include <string.h>

功能：计算字符串s的(unsigned int型）长度

说明：返回s的长度，**不包括结束符NULL。**

from:<http://blog.csdn.net/pipisorry/article/details/36898405>

ref:http://blog.csdn.net/21aspnet/article/details/1539970