(4条消息)C语言字符串操作总结大全(超详细)_C/C++_jing16337305的博客-CSDN博客

转载自：https://www.cnblogs.com/lidabo/p/5225868.html

1）字符串操作

strcpy(p, p1) 复制字符串
strncpy(p, p1, n) 复制指定长度字符串
strcat(p, p1) 附加字符串
strncat(p, p1, n) 附加指定长度字符串
strlen(p) 取字符串长度
strcmp(p, p1) 比较字符串
strcasecmp忽略大小写比较字符串
strncmp(p, p1, n) 比较指定长度字符串
strchr(p, c) 在字符串中查找指定字符
strrchr(p, c) 在字符串中反向查找
strstr(p, p1) 查找字符串
strpbrk(p, p1) 以目标字符串的所有字符作为集合，在当前字符串查找该集合的任一元素
strspn(p, p1) 以目标字符串的所有字符作为集合，在当前字符串查找不属于该集合的任一元素的偏移
strcspn(p, p1) 以目标字符串的所有字符作为集合，在当前字符串查找属于该集合的任一元素的偏移
* 具有指定长度的字符串处理函数在已处理的字符串之后填补零结尾符

2）字符串到数值类型的转换

strtod(p, ppend) 从字符串 p 中转换 double 类型数值，并将后续的字符串指针存储到 ppend 指向的 char* 类型存储。
strtol(p, ppend, base) 从字符串 p 中转换 long 类型整型数值，base 显式设置转换的整型进制，设置为 0 以根据特定格式判断所用进制，0x, 0X 前缀以解释为十六进制格式整型，0　前缀以解释为八进制格式整型
atoi(p) 字符串转换到 int 整型
atof(p) 字符串转换到 double 符点数
atol(p) 字符串转换到 long 整型

3）字符检查

isalpha() 检查是否为字母字符
isupper() 检查是否为大写字母字符
islower() 检查是否为小写字母字符
isdigit() 检查是否为数字
isxdigit() 检查是否为十六进制数字表示的有效字符
isspace() 检查是否为空格类型字符
iscntrl() 检查是否为控制字符
ispunct() 检查是否为标点符号
isalnum() 检查是否为字母和数字
isprint() 检查是否是可打印字符
isgraph() 检查是否是图形字符，等效于 isalnum() | ispunct()

4）函数原型

原型：strcpy(char destination[], const char source[]);
功能：将字符串source拷贝到字符串destination中
例程：

```
 #include <iostream.h>
#include <string.h>
void main(void)
{
    char str1[10] = { "TsinghuaOK"};
    char str2[10] = { "Computer"};
    cout <<strcpy(str1,str2)<<endl;
}
```

运行结果是:Computer
第二个字符串将覆盖掉第一个字符串的所有内容!
注意：在定义数组时，字符数组1的字符串长度必须大于或等于字符串2的字符串长度。不能用赋值语句将一个字符串常量或字符数组直接赋给一个字符数组。所有字符串处理函数都包含在头文件string.h中。

strncpy(char destination[], const char source[], int numchars);

strncpy：将字符串source中前numchars个字符拷贝到字符串destination中。
strncpy函数应用举例
原型：strncpy(char destination[], const char source[], int numchars);
功能：将字符串source中前numchars个字符拷贝到字符串destination中
例程：

```
#include <iostream.h>
#include <string.h>
void main(void)
{
    char str1[10] = { "Tsinghua "};
    char str2[10] = { "Computer"};
    cout <<strncpy(str1,str2,3)<<endl;
}
```

运行结果：Comnghua
注意：字符串source中前numchars个字符将覆盖掉字符串destination中前numchars个字符！

原型：strcat(char target[], const char source[]);
功能：将字符串source接到字符串target的后面
例程：

```
#include <iostream.h>
#include <string.h>
void main(void)
{
    char str1[] = { "Tsinghua "};
    char str2[] = { "Computer"};
    cout <<strcpy(str1,str2)<<endl;
}
```

运行结果：Tsinghua Computer

注意：在定义字符数组1的长度时应该考虑字符数组2的长度，因为连接后新字符串的长度为两个字符串长度之和。进行字符串连接后，字符串1的结尾符将自动被去掉，在结尾串末尾保留新字符串后面一个结尾符。

原型：strncat(char target[], const char source[], int numchars);
功能：将字符串source的前numchars个字符接到字符串target的后面
例程：

```
#include <iostream.h>
#include <string.h>
void main(void)
{
   char str1[] = { "Tsinghua "};
   char str2[] = { "Computer"};
   cout <<strncat(str1,str2,3)<<endl;
}
```

运行结果：Tsinghua Com

原型：int strcmp(const char firststring[], const char secondstring);
功能：比较两个字符串firststring和secondstring
例程：

```
#include <iostream.h>
#include <string.h>
void main(void)
{
   char buf1[] = "aaa";
   char buf2[] = "bbb";
   char buf3[] = "ccc";
```

```
    int ptr;
    ptr = strcmp(buf2,buf1);
    if(ptr > 0)
      cout <<"Buffer 2 is greater than buffer 1"<<endl;
    else
      cout <<"Buffer 2 is less than buffer 1"<<endl;
    ptr = strcmp(buf2,buf3);
    if(ptr > 0)
      cout <<"Buffer 2 is greater than buffer 3"<<endl;
    else
      cout <<"Buffer 2 is less than buffer 3"<<endl;
}
```

运行结果是:Buffer 2 is less than buffer 1
Buffer 2 is greater than buffer 3

原型：strlen(const char string[]);
功能：统计字符串string中字符的个数
例程：

```
#include <iostream.h>
#include <string.h>
void main(void)
{
  char str[100];
  cout <<"请输入一个字符串:";
  cin >>str;
  cout <<"The length of the string is :"<<strlen(str)<<"个"<<endl;
}
```

运行结果The length of the string is x (x为你输入的字符总数字)

注意：strlen函数的功能是计算字符串的实际长度，不包括'\0'在内。另外，strlen函数也可以直接测试字符串常量的长度，如：strlen("Welcome")。

void *memset(void *dest, int c, size_t count);
将dest前面count个字符置为字符c. 返回dest的值.

void *memmove(void *dest, const void *src, size_t count);
从src复制count字节的字符到dest. 如果src和dest出现重叠, 函数会自动处理. 返回dest的值.

void *memcpy(void *dest, const void *src, size_t count);
从src复制count字节的字符到dest. 与memmove功能一样, 只是不能处理src和dest出现重叠. 返回dest的值.

void *memchr(const void *buf, int c, size_t count);
在buf前面count字节中查找首次出现字符c的位置. 找到了字符c或者已经搜寻了count个字节, 查找即停止. 操作成功则返回buf中首次出现c的位置指针, 否则返回NULL.

void *_memccpy(void *dest, const void *src, int c, size_t count);
从src复制0个或多个字节的字符到dest. 当字符c被复制或者count个字符被复制时, 复制停止.

如果字符c被复制, 函数返回这个字符后面紧挨一个字符位置的指针. 否则返回NULL.

int memcmp(const void *buf1, const void *buf2, size_t count);
比较buf1和buf2前面count个字节大小.
返回值< 0, 表示buf1小于buf2;
返回值为0, 表示buf1等于buf2;
返回值> 0, 表示buf1大于buf2.

int memicmp(const void *buf1, const void *buf2, size_t count);

比较buf1和buf2前面count个字节. 与memcmp不同的是, 它不区分大小写.

返回值同上.

char *strrev(char *string);
将字符串string中的字符顺序颠倒过来. NULL结束符位置不变. 返回调整后的字符串的指针.

char *_strupr(char *string);
将string中所有小写字母替换成相应的大写字母, 其它字符保持不变. 返回调整后的字符串的指针.

char *_strlwr(char *string);
将string中所有大写字母替换成相应的小写字母, 其它字符保持不变. 返回调整后的字符串的指针.

char *strchr(const char *string, int c);
查找字 串string中首次出现的位置, NULL结束符也包含在查找中. 返回一个指针, 指向字符c在字符串string中首次出现的位置, 如果没有找到, 则返回NULL.

char *strrchr(const char *string, int c);
查找字符c在字符串string中最后一次出现的位置, 也就是对string进行反序搜索, 包含NULL结束符.
返回一个指针, 指向字符c在字符串string中最后一次出现的位置, 如果没有找到, 则返回NULL.

char *strstr(const char *string, const char *strSearch);
在字符串string中查找strSearch子串. 返回子串strSearch在string中首次出现位置的指针. 如果没有找到子串strSearch, 则返回NULL. 如果子串strSearch为空串, 函数返回string值.

char *strdup(const char *strSource);
函数运行中会自己调用malloc函数为复制strSource字符串分配存储空间, 然后再将strSource复制到分配到的空间中. 注意要及时释放这个分配的空间.
返回一个指针, 指向为复制字符串分配的空间; 如果分配空间失败, 则返回NULL值.

char *strcat(char *strDestination, const char *strSource);
将源串strSource添加到目标串strDestination后面, 并在得到的新串后面加上NULL结束符. 源串strSource的字符会覆盖目标串strDestination后面的结束符NULL. 在字符串的复制或添加过程中没有溢出检查, 所以要保证目标串空间足够大. 不能处理源串与目标串重叠的情况. 函数返回strDestination值.

char *strncat(char *strDestination, const char *strSource, size_t count);
将源串strSource开始的count个字符添加到目标串strDest后. 源串strSource的字符会覆盖目标串strDestination后面的结束符NULL. 如果count大于源串长度, 则会用源串的长度值替换count值. 得到的新串后面会自动加上NULL结束符. 与strcat函数一样, 本函数不能处理源串与目标串重叠的情况. 函数返回strDestination值.

char *strcpy(char *strDestination, const char *strSource);
复制源串strSource到目标串strDestination所指定的位置, 包含NULL结束符. 不能处理源串与目标串重叠的情况.函数返回strDestination值.

char *strncpy(char *strDestination, const char *strSource, size_t count);
将源串strSource开始的count个字符复制到目标串strDestination所指定的位置. 如果count值小于或等于strSource串的长度, 不会自动添加NULL结束符目标串中, 而count大于strSource串的长度时, 则将strSource用NULL结束符填充补齐count个字符, 复制到目标串中. 不能处理源串与目标串重叠的情况.函数返回strDestination值.

char *strset(char *string, int c);
将string串的所有字符设置为字符c, 遇到NULL结束符停止. 函数返回内容调整后的string指针.

char *strnset(char *string, int c, size_t count);
将string串开始count个字符设置为字符c, 如果count值大于string串的长度, 将用string的长度替换count值. 函数返回内容调整后的string指针.

size_t strspn(const char *string, const char *strCharSet);
查找任何一个不包含在strCharSet串中的字符 (字符串结束符NULL除外) 在string串中首次出现的位置序号. 返回一个整数值, 指定在string中全部由characters中的字符组成的子串的长度. 如果string以一个不包含在strCharSet中的字符开头, 函数将返回0值.

size_t strcspn(const char *string, const char *strCharSet);
查找strCharSet串中任何一个字符在string串中首次出现的位置序号, 包含字符串结束符NULL.
返回一个整数值, 指定在string中全部由非characters中的字符组成的子串的长度. 如果string以一个包含在strCharSet中的字符开头, 函数将返回0值.

char *strspnp(const char *string, const char *strCharSet);
查找任何一个不包含在strCharSet串中的字符 (字符串结束符NULL除外) 在string串中首次出现的位置指针. 返回一个指针, 指向非strCharSet中的字符在string中首次出现的位置.

char *strpbrk(const char *string, const char *strCharSet);
查找strCharSet串中任何一个字符在string串中首次出现的位置, 不包含字符串结束符NULL.
返回一个指针, 指向strCharSet中任一字符在string中首次出现的位置. 如果两个字符串参数不含相同字符, 则返回NULL值.

int strcmp(const char *string1, const char *string2);
比较字符串string1和string2大小.
返回值< 0, 表示string1小于string2;

返回值为0, 表示string1等于string2;
返回值> 0, 表示string1大于string2.

int stricmp(const char *string1, const char *string2);
比较字符串string1和string2大小，和strcmp不同, 比较的是它们的小写字母版本.返回值与strcmp相同.

int strcmpi(const char *string1, const char *string2);
等价于stricmp函数, 只是提供一个向后兼容的版本.

int strncmp(const char *string1, const char *string2, size_t count);
比较字符串string1和string2大小，只比较前面count个字符. 比较过程中, 任何一个字符串的长度小于count, 则count将被较短的字符串的长度取代. 此时如果两串前面的字符都相等, 则较短的串要小.
返回值< 0, 表示string1的子串小于string2的子串;
返回值为0, 表示string1的子串等于string2的子串;
返回值> 0, 表示string1的子串大于string2的子串.

int strnicmp(const char *string1, const char *string2, size_t count);
比较字符串string1和string2大小，只比较前面count个字符. 与strncmp不同的是, 比较的是它们的小写字母版本. 返回值与strncmp相同.

char *strtok(char *strToken, const char *strDelimit);
在strToken 串中查找下一个标记, strDelimit字符集则指定了在当前查找调用中可能遇到的分界符. 返回一个指针, 指向在strToken中找到的下一个标记. 如果找不到标记, 就返回NULL值. 每次调用都会修改strToken内容, 用NULL字符替换遇到的每个分界符.

c++概念字符串操作

一、char_traits 字符特征类
1）意义：包装特定串元素的通用行为界面，以便容器实现时依据特征信息而执行特定行为
2）定义了通用类型名

typedef _Elem char_type;
typedef int int_type;
typedef streampos pos_type;

typedef streamoff off_type;
typedef mbstate_t state_type;

其中 int_type 表示字符元素转换到特定编码时的整型表示，pos_type, off_type 分别作为字符串索引和字符串元素偏移的类型，类似容器迭中的指针，迭代类型和指针，迭代器的偏移类型。最后的 state_type 用于存储流状态，如出错，格式控制等等。

3）定义了字符 / 字符串操作的包装界面，以便通用算法的调用

assign(a, b) 定义将 b 字符赋值给 a 字符的过程，实现 a.operator = 的行为
eq(a, b) 定义 a 字符和 b 字符的相等关系，实现 a.operator == 的行为
lt(a, b) 定义 a 小于 b 的关系，实现 a.operator < 的行为
compare(a_ptr, b_ptr, cnt) 定义两组字符串的比较，返回 int 类型，实现类似 memcmp 的行为
length(ptr) 定义取字符串长度，实现类似 strlen 的行为
copy(a_ptr, b_ptr, cnt) 定义两组字符串的复制，实现类似 memcpy 的行为
move(a_ptr, b_ptr, cnt) 定义两组字符串的不重叠复制，实现类似 memmove 的行为
assign(ptr, cnt, ch) 定义了填充字符串的过程，实现类似 memset 的行为
to_int_type(ch) 定义了 char_type 到 int_type 整型的转换过程
to_char_type(n) 定义了 int_type 到 char_type 字符型的转换过程
eq_int_type(a, b) 定义两个和当前 char_type 类型对应的 int_type 的相等关系
eof() 定义字符串结尾符，使用整型表示
not_eof(n) 定义非字符串结尾符，若输入结尾符，则返回 1，其他输入返回原值，即总是不返回 eof()

4）int_type 类型应是当前字符类型的整型编码

二、std::string 并不是序列容器，没有 front() 和 back() 界面用于取出前端和尾端的元素，使用 std::string::operator [] 并传递 streampos 类型取得特定元素，如 std::string::size() - 1 作为索引取得最后一个字符

三、basic_string 支持的初始化

1）默认初始化
2）分配器
3）复制构造

4）局部复制 [_Roff, _Roff + _Count)
5）局部复制 + 分配器
6）C 字符串 [_Ptr, <null>)
7）C 字符串 + _Count [_Ptr, _Ptr + _Count)
8）C 字符串 + 分配器
9）C 字符串 + _Count + 分配器 [_Ptr, _Ptr + _Count)

10）_Count * _Ch
11）_Count * _Ch + 分配器
12）迭代器 [_ItF, _ItL)
13）迭代器 + 分配器

字符到串不能初始化，但支持 operator = 赋值和 operator += 累加赋值运算。

四、字符串的区间有效性

对串的索引访问在超过字符串的有效区间时，因为串的在实现上对内置的字符缓冲区执行下标访问，所以不会导致异常，但是将得到不可预知的结果，通常是不可用的。
将其他字符串作为右值输入时，对该串取出计数大于串大小时按串大小计算。
std::basic_string::size_type 的实际类型为 size_t，在 Visual C++ 7.1 中实现为 unsigned，std::basic_string::npos 被静态设定为

(basic_string<_Elem, _Traits, _Alloc>::size_type)(-1);

在查找子字符串等操作时，函数返回 npos 的值表示非法索引。

五、比较字符串

允许的比较对象
1）compare(s2) 其他同类型字符串
2）compare(p) C 风格字符串
3）compare(off, cnt, s2) [off, off + cnt) 同 s2 执行比较
4）compare(off, cnt, s2, off2, cnt2) [off, off + cnt) 同 s2 [off2, cnt2) 执行比较

5）compare(off, cnt, p) [off, off + cnt) 同 [p , <null>) 执行比较
6）compare(off, cnt, p, cnt2) [off, off + cnt) 同 [p, p + cnt2) 执行比较

返回 -1, 0, 1 作为小于、等于和大于的比较结果。

六、附加数据

1）使用 operator += 接受其他字符串，C 风格字符串和字符
2）使用 push_back() 在尾部附加字符，并使得通过字符串构造的 back_iterator 可以访问
3）append() 附加
1、append(s) 追加字符串
2、append(s, off, cnt) 追加字符串 s [off, off + cnt)
3、append(p) 追加字符串 [p, <null>)
4、append(p, cnt) 追加字符串 [p, p + cnt)
5、append(n, c) 填充 n * c
6、append(InF, InL) 追加输入流 [InF, InL)

4）insert() 插入
1、insert(off, s2) 插入字符串
2、insert(off, s2, off2, cnt2) 插入字符串 s [off2, off2 + cnt2)
3、insert(off, p) 插入字符串 [p, <null>)
4、insert(off, p, cnt) 插入字符串 [p, p + cnt)

5、insert(off, n, c) 插入 n * c
6、insert(iter) 元素默认值填充
7、insert(iter, c) 插入特定元素
8、insert(iter, n, c) 插入 n*c
9、insert(iter, InF, InL) 插入 [InF, InL)

5）operator +(a, b)
字符串关联运算符重载中支持 operator + 的形式
1、s + s
2、s + p

3、s + c
4、p + s
5、c + s

七、查找、替换和清除

1）find() 查找

1、find(c, off) 在 s [off, npos) 中查找 c
2、find(p, off, n) 在 s [off, npos) 中查找 [p, p + n)
3、find(p, off) 在 s [off, npos) 中查找 [p, <null>)
4、find(s2, off) 在 s [off, npos) 中查找 s2

2）find() 的变种

1、rfind() 具有 find() 的输入形式，反序查找
2、find_first_of() 具有 find() 的输入形式，返回第一个匹配的索引
3、find_last_of() 具有 find() 的输入形式，返回倒数第一个匹配的索引
4、find_first_not_of() 具有 find() 的输入形式，返回第一个不匹配的索引
5、find_last_not_of() 具有 find() 的输入形式，返回倒数第一个不匹配的索引

3）replace() 替换

1、replace(off, cnt, s2) 将 s [off, off + cnt) 替换成 s2
2、replace(off, cnt, s2, off2, cnt2) 将 s [off, off + cnt) 替换成 s2 [off2, off2 + cnt2)
3、replace(off, cnt, p) 将 s [off, off + cnt) 替换成 [p, <null>)
4、replace(off, cnt, p, cnt2) 将 s [off, off + cnt) 替换成 [p, p + cnt2)

5、replace(off, cnt, n, c) 将 s [off, off + cnt) 替换成 c * n

使用迭代器的情况：

6、replace(InF, InL, s2) 将 [InF, InL) 替换成 s2
7、replace(InF, InL, p) 将 [InF, InL) 替换成 [p, <null>)
8、replace(InF, InL, p, cnt) 将 [InF, InL) 替换成 [p, p + cnt)
9、replace(InF, InL, n, c) 将 [InF, InL) 替换成 n * c
10、replace(InF, InL, InF2, InL2) 将 [InF, InL) 替换成 [InF2, InL2)

4）erase() 删除

1、erase(off, cnt) 从字符串 s 中删除 s [off, off + cnt)

2、erase(iter) 从字符串 s 中删除 *iter

3、erase(ItF, ItL) 从字符串 s 中删除 [ItF, ItL)

八、取出字符串

1）取得 C 风格字符串

c_str() 返回常量类型的 C 风格字符串指针，copy(ptr, cnt, off = 0) 则将指定大小的字符串复制到特定指针。data() 在 Visual C++ 7.1 中仅仅调用了 c_str() 实现。

2）取得子字符串

substr(off, cnt) 取得 s [off, off + cnt) 的副本。

3）复制子字符串

copy(p, off, cnt) 将 s [off, off + cnt) 复制到 p。

九、字符串的缓冲区管理

字符串具有类似 std::vector 的缓冲区管理界面。

size() 取得有效元素长度

max_size() 取得当前内存分配器能分配的有效空间

reserve() 为缓冲区预留空间

capacity() 取得缓冲区的容量

resize() 重设串的长度，可以为其指定初始化值

十、定义输入迭代器的尾端

向 istream_iterator 传递输入流对象以创建输入迭代器，输入迭代器持有输入流对象的指针，默认创建和读取流失败的情况下该指针被设置为 0。并且在实现输入迭代器间的 operator == 相等运算时，进行持有的流对象指针的相等比较，这样，默认创建的输入迭代器将被用于匹配输入流的结束。

* 当输入流读取失败，用户执行 if, while 条件判断时，实际上先将判断值转换成 void* 类型，或者根据 operator ! 运算符的返回结果，对输入流重载 operator void* 和 operator ! 运算符，可以定义输入流在布尔表达式中的行为，使得当流读取失败的情况下，输入迭代器可以通过布尔表达式来确认，而不是显式访问 fail() 成员函数.