# 定义

串（string）是由零个或多个字符组成的有限序列，又名叫字符串。

# 存储

## 顺序存储

#include<stdio.h>

#include<stdlib.h>

#define MAX\_SIZE 128

typedef char String[MAX\_SIZE];

int StringIndex(String S, String T, int pos){

int i=pos;

int j=1;

while(i<S[0] && j<=TT[0]){

if(S[i]==T[i]){

++i;

++j;

}else{

i=i+1-(j-1);

j=i;

}

}

if(j>T[0])

return i-T[0];

else

return 0;

}

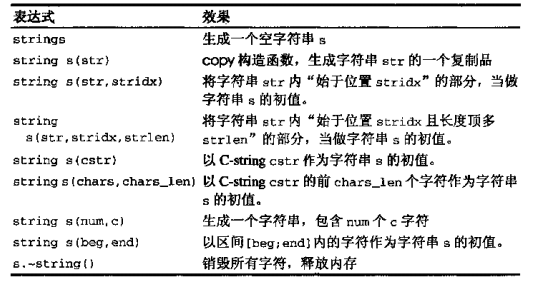
## 链式存储

char \*ch;

访问元素：ch[n]

# 方法

## 构造/析构函数



## 大小和容量

size()/length()：大小

max\_size()：最多包含的字符数

capacity()：重新分配内存之前，string所能包含的最大字符数

## 元素存取

有两种方法可以访问单一字符：下标操作符[]合成员函数at()

注：下标操作符operator[]并不检查索引是否有效，at()则会检查。如果调用at()时指定的索引无效，系统会抛出out\_of\_range异常，如果调用operator[]时指定的索引无效，其行为未定义——可能存取非法内存，因而引起某些讨厌的边缘效应或甚至崩溃。

## 比较

String支持常见的比较操作符，操作数可以是string或C-string，可以使用符号<，<=，>，》=和compare()成员函数比较。

## 更改内容

赋值：operator=、assign()

注：如果需要多个参数描述新值，可以采用成员函数assign()赋值。

交换：swap()

令string为空：s = ""、clear()、erase()

插入字符：operator+=、append()、push\_back()、insert()

删除字符：erase()、replace()

## 子串

提取子字符串：substr()

## 字符串接合

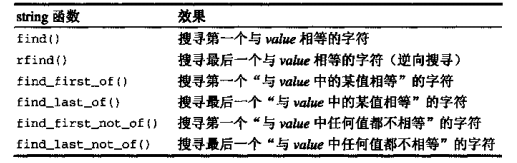
把两个string（或C-string）接合：operator+

## I/O操作

operator>>从input stream读取一个string

operator<<把一个string写到output stream中

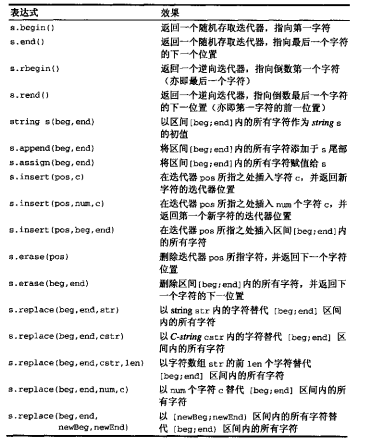
## 查找和搜索



## 数值npos

如果搜寻函数失败，就会返回string::npos。

## 迭代器操作



## 国际化

## 效率

如果你希望速度更快，请确认你使用的string classes采用了类似reference counting（引用计数）概念，这种手法可以加速string的复制和赋值操作，因为在实际操作中不再是对字符串内容进行操作，而仅仅是复制和赋值字符串的reference。

通过reference counting，你甚至不必透过const refrence来传递字符串；不过基于灵活性和可移植性的考虑，一般还是采用const reference来传递参数。

# 算法

## 字符串匹配

### 暴力法

### Hash函数

### KMP算法

## 字典树/Trie

## 后缀数组/SA

## 后缀树

# 应用

## 遍历

### 递归求字符串的长度

思路：

字符串末尾结束表示：\0

编程：Window—DecC++

代码：

#include<stdio.h>

#include<stdlib.h>

int RecursiveStringLength(char \*ch)

{

if(ch[0] == '\0')

return 0;

else

return RecursiveStringLength(ch+1)+1;

}

int main()

{

char ch[] = "L love coding";

printf("char length is :%d\n",RecursiveStringLength(ch));

getchar();

}

**版本2：**

#include <stdio.h>

#include <stdlib.h>

/\*\*

递归的方式求字符串长度

\*/

int RecurLength(char\* str)

{

if(str[0] == '\0')

return 0;

else

return RecurLength(str+1)+1;

}

// 字符串中最后一个单词的长度

int LastWordLen(char\* str,int len)

{

int lastlen = 0;

int index = len-1;

while(index>=0 && str[index--] != ' ')

lastlen++;

return lastlen;

}

int LastWordLen2(char\* str)

{

int lastlen = 0;

int index =0;

while(str[index] !='\0')

{

if(str[index]==' ')

lastlen =0;

else

lastlen++;

index++;

}

return lastlen;

}

/\*

字符串内存的拷贝 实现memmove函数

\*/

char\* my\_memmove(char\* dst,char\* src,int count)

{

char\* ret = dst;

if(dst == NULL || src == NULL)

return NULL;

//如果dst和src区域没有重叠，那么从开始处逐一拷贝

if(dst <= src || dst >= (src+count))

{

while(count--)

{

\*dst = \*src;

dst++;

src++;

}

}

//如果dst和src区域有交叉，那么从尾部开始向起始位置拷贝，这样可

以避免数据重叠

else

{

dst = dst + count-1;

src = src + count-1;

while(count--)

{

\*dst = \*src;

dst--;

src--;

}

}

return ret;

}

int main()

{

char str[]="jike xueyuan";

printf("str len is %d\r\n",RecurLength(str));

// printf("lastword len is %d\r\n",LastWordLen(str,12));

printf("lastword len is %d\r\n",LastWordLen2(str));

return 0;

}

### 字符串最后一个单词长度

思路：

整个字符串包含多个单词

每个单词中间使用空格隔开

字符串中只有单词，没有别的字符

代码：

## 查找

### 字符串查找

代码：

#include <stdio.h>

#include <stdlib.h>

// 查找字符串中各个字符出现的次数

void FindTimes(char\* str)

{

int hash[256]={0};

int index =0;

while(str[index]!='\0')

{

hash[str[index]]++;

index++;

}

for(index=0;index<256;index++)

printf("%d ",hash[index]);

printf("\r\n");

}

// 如何翻转一个句子，将整个句子以单词为单位进行翻转

/\*\*

首先完成整个句子的翻转

然后完成内部单词的二次翻转

\*/

//对字符串中固定位置的区域进行翻转

void Reverse(char\* str,int begin,int end)

{

char tmp;

while(begin < end)

{

tmp = str[begin];

str[begin] = str[end];

str[end] = tmp;

begin++;

end--;

}

}

/\*

字符串内部单词的翻转 从头部开始 每确定一个单词就进行翻转

\*/

void Reverse\_second(char\* str,int len)

{

int begin,i,j;

for(i=0;i<len;)

{

begin =i;

for(j=i;j<len;j++)

if(str[j]==' ')

break;

Reverse(str,begin,j-1);

i = j+1;

}

}

int main()

{

char str[]="jike xue yuan huan ying ni";

// FindTimes(str);

Reverse(str,0,25);

Reverse\_second(str,26);

printf("%s \r\n",str);

return 0;

}

### 第一个只出现一次的字符

注：剑指offer 186

### 找出字符串中第一个匹配项的下标

给你两个字符串 haystack 和 needle ，请你在 haystack 字符串中找出 needle 字符串的第一个匹配项的下标（下标从 0 开始）。如果 needle 不是 haystack 的一部分，则返回 -1 。

注：LeetCode 28

分析：采用双指针实现。

### 字符流中第一个不重复的字符

题目要求：请实现一个函数用来找出字符流中第一个只出现一次的字符。例如，当从字符流中只读出前两个字符“go”时，第一个出现一次的字符是“g”。当从该字符流中读出前六个字符“google”时，第一个只出现一次的字符是“1”。

注：剑指offer P269

## 匹配

### 重复的子字符串

给定一个非空的字符串 s ，检查是否可以通过由它的一个子串重复多次构成。

注：LeetCode 459

分析：

这个问题的核心思路是尝试找到一个长度小于等于原始字符串长度一半的子串，使得原始字符串可以由这个子串重复多次构成。

具体实现步骤如下：

1、枚举所有可能的子串长度len，从1到原始字符串长度的一半。

2、对于每个长度为len的子串，检查是否能够构成原始字符串。如果原始字符串长度不能整除len，则跳过该子串长度。

3、构造一个重复拼接子串的目标字符串，并与原始字符串比较是否相等。如果相等，则返回 true。

4、如果遍历完所有可能的子串长度都没有找到合适的子串，则返回 false。

这样就能判断原始字符串是否可以由一个子串重复多次构成了。

### 旋转字符串

给定两个字符串, s 和 goal。如果在若干次旋转操作之后，s 能变成 goal ，那么返回 true 。

s 的 旋转操作 就是将 s 最左边的字符移动到最右边。

例如, 若 s = 'abcde'，在旋转一次之后结果就是'bcdea' 。

注：LeetCode 796

分析：

通过加倍字符串，搜索子字符串方式实现，与重复子字符串类似。

### 字符串轮转

字符串轮转。给定两个字符串s1和s2，请编写代码检查s2是否为s1旋转而成（比如，waterbottle是erbottlewat旋转后的字符串）。

注：面试题01.09（与前面旋转字符串LeetCode 796一样）

### 最大重复子字符串

给你一个字符串 sequence ，如果字符串 word 连续重复 k 次形成的字符串是 sequence 的一个子字符串，那么单词 word 的 重复值为 k 。单词 word 的 最大重复值 是单词 word 在 sequence 中最大的重复值。如果 word 不是 sequence 的子串，那么重复值 k 为 0 。

给你一个字符串 sequence 和 word ，请你返回 最大重复值 k 。

注：LeetCode 1668

分析：

仍然采用加倍字符串/模拟/字符串匹配的方式。

### 重复叠加字符串匹配

给定两个字符串 a 和 b，寻找重复叠加字符串 a 的最小次数，使得字符串 b 成为叠加后的字符串 a 的子串，如果不存在则返回 -1。

注意：字符串 "abc" 重复叠加 0 次是 ""，重复叠加 1 次是 "abc"，重复叠加 2 次是 "abcabc"。

注：LeetCode 686

分析：

可以直接采用暴力破解方法，循环检查b是否为a叠加的tmlStr的子串，如果不是则继续tmpStr += a，直到检查为子串为止。

class Solution {

public:

int repeatedStringMatch(string a, string b) {

int count = 1;

string repeated = a;

// 不断重复叠加字符串 a，直到字符串 b 成为其子串或者重复叠加次数超过 2 \* a.length() + b.length()

while (repeated.find(b) == string::npos && repeated.length() <= 2 \* a.length() + b.length()) {

// 重复叠加一次

repeated += a;

count++;

}

// 如果超过 2 \* a.length() + b.length() 仍然没有找到子串，则返回 -1

if (repeated.find(b) == string::npos) {

return -1;

}

return count;

}

};

### 数组中的字符串匹配

给你一个字符串数组 words ，数组中的每个字符串都可以看作是一个单词。请你按 任意 顺序返回 words 中是其他单词的子字符串的所有单词。

如果你可以删除 words[j] 最左侧和/或最右侧的若干字符得到 words[i] ，那么字符串 words[i] 就是 words[j] 的一个子字符串。

注：LeetCode 1408

分析：采用暴力破解

### 检查单词是否为句中其他单词的前缀

注：LeetCode 1455

分析：

### 统计包含给定前缀的字符串

给你一个字符串数组 words 和一个字符串 pref 。

返回 words 中以 pref 作为 前缀 的字符串的数目。

字符串 s 的 前缀 就是 s 的任一前导连续字符串。

注：LeetCode 2185

分析：

直接采用模拟方式遍历实现。

### 统计前后缀下标对 I

给你一个下标从 0 开始的字符串数组 words 。

定义一个 布尔 函数 isPrefixAndSuffix ，它接受两个字符串参数 str1 和 str2 ：

当 str1 同时是 str2 的前缀（prefix）和后缀（suffix）时，isPrefixAndSuffix(str1, str2) 返回 true，否则返回 false。

例如，isPrefixAndSuffix("aba", "ababa") 返回 true，因为 "aba" 既是 "ababa" 的前缀，也是 "ababa" 的后缀，但是 isPrefixAndSuffix("abc", "abcd") 返回 false。

以整数形式，返回满足 i < j 且 isPrefixAndSuffix(words[i], words[j]) 为 true 的下标对 (i, j) 的 数量 。

注：LeetCode 3042

分析：

直接采用遍历模拟方式实现。

### 正则表达式匹配

题目要求：请实现一个函数用来匹配包含“.”和“\*”的正则表达式。模式中的字符“.”表示任意一个字符，而“\*”表示它的当前的字符可以出现任意次（含0次）。在本题中，匹配是指字符串的所有字符匹配整个模式。例如，字符串“aaa”与模式“a.a”和“ab\*ac\*a”匹配，但是与“aa.a”及“ab\*a”均不匹配。

注：剑指offer P265

### 驼峰匹配

给你一个字符串数组 queries，和一个表示模式的字符串 pattern，请你返回一个布尔数组 answer 。只有在待查项 queries[i] 与模式串 pattern 匹配时， answer[i] 才为 true，否则为 false。

如果可以将小写字母插入模式串 pattern 得到待查询项 query，那么待查询项与给定模式串匹配。可以在任何位置插入每个字符，也可以不插入字符。

注：LeetCode 1023

分析：

采用双指针模拟实现。

### 最长回文串

给你一个字符串 s，找到 s 中最长的回文子串。

如果字符串的反序与原始字符串相同，则该字符串称为回文字符串。

注：LeetCode 5

说明：

对于这类求最值的问题，并且后面的解与前面有关联得，可以采用动态规划实现。

### 最长公共前缀

### 无重复字符的最长子串

给定一个字符串 s ，请你找出其中不含有重复字符的 最长

子串的长度。

注：LeetCode 3

说明：对于子串问题，一般可以采用双指针的方法。

这里可以采用哈希表。

### 重复的DNA序列

### 最小窗口子串

### 同字符词语分词

### 词语模式

### 字符串循环左移

### 另一棵树的子树

给你两棵二叉树 root 和 subRoot 。检验 root 中是否包含和 subRoot 具有相同结构和节点值的子树。如果存在，返回 true ；否则，返回 false 。

二叉树 tree 的一棵子树包括 tree 的某个节点和这个节点的所有后代节点。tree 也可以看做它自身的一棵子树。

注：LeetCode 572

分析：

这个其实不是字符串的匹配问题，是树的匹配问题。

## 排序

### 字符串排序

题目要求：一个字符串中包含着大小写的英文字符，排序后使得大写字符在前，小写字符在后。

如果需要保持原始小写字符的相对位置该怎么办？

代码：

#include <stdio.h>

#include <stdlib.h>

// 排序字符串 使得字符串中大写字符在前 小写字符在后

void StrSort(char\* str)

{

int index =-1;

int pos =0;

char temp;

for(;str[pos]!='\0';pos++)

{

if(str[pos] >= 'A' && str[pos]<='Z')

{

index++;

temp = str[index];

str[index] = str[pos];

str[pos]= temp;

}

}

}

// 同样的要求 但是使得小写字符的相对顺序不改变

void StrSort2(char\* str)

{

int len =0;

while(str[len]!='\0')

len++;

int index =len;

int pos = len-1;

char temp;

for(;pos>=0;pos--)

{

if(str[pos] >= 'a' && str[pos] <= 'z')

{

index--;

temp = str[index];

str[index]=str[pos];

str[pos] = temp;

}

}

}

int main()

{

char str[] ="bdAFfdDAfdAfaFAfaGSdDFfdsaGd";

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

StrSort2(str);

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

return 0;

}

### 字符串的所有排序

题目要求：输入一个字符串，打印出该字符串中字符的所有排序。例如输入字符串abc，则打印出由字符a、b、c所能排列出来的所有字符串abc、acb、bca、cab和cba。

注：剑指offer P154

## 运算

### 表示数值的字符串

题目要求：请实现一个函数用来判断字符串是否表示数组（包括整数和小数）。例如，字符串“+100”、“5e2”、“-123”、“3.1416”及“-1E-16”都表示数值，但“12e”、“1a3.14”、“1.2.3”、“+-5”、“12e+5.4”都不是。

注：剑指offer P267

### 实现memmove()

### atoi()实现

请你来实现一个 myAtoi(string s) 函数，使其能将字符串转换成一个 32 位有符号整数（类似 C/C++ 中的 atoi 函数）。

函数 myAtoi(string s) 的算法如下：

1、读入字符串并丢弃无用的前导空格

2、检查下一个字符（假设还未到字符末尾）为正还是负号，读取该字符（如果有）。 确定最终结果是负数还是正数。 如果两者都不存在，则假定结果为正。

3、读入下一个字符，直到到达下一个非数字字符或到达输入的结尾。字符串的其余部分将被忽略。

4、将前面步骤读入的这些数字转换为整数（即，"123" -> 123， "0032" -> 32）。如果没有读入数字，则整数为 0 。必要时更改符号（从步骤 2 开始）。

5、如果整数数超过 32 位有符号整数范围 [−231, 231 − 1] ，需要截断这个整数，使其保持在这个范围内。具体来说，小于 −231 的整数应该被固定为 −231 ，大于 231 − 1 的整数应该被固定为 231 − 1 。

6、返回整数作为最终结果。

注意：

本题中的空白字符只包括空格字符 ' ' 。

除前导空格或数字后的其余字符串外，请勿忽略 任何其他字符。

注：

LeetCode 8

代码：

#include <iostream>

#include <vector>

#include <string>

#include <limits>

using namespace std;

/\*

实现atoi函数

需要注意的地方：

1.数值越界

2.正负数

3.字符串结束标志

\*/

int StringToInt(string& str)

{

if(str.length() == 0)

return 0;

int i;

int flag =1;

int result =0;

int digit;

// 清空空格字符

for(i=0;i<str.length();i++)

if(str[i] !=' ')

break;

//查找正负号

if(str[i] == '+')

i++;

if(str[i]== '-')

{

flag =0;

i++;

}

//开始处理字符

for(;i<str.length();i++)

{

if(str[i]<'0' || str[i] > '9')

break;

digit = str[i]-'0';

//判断越界

if(flag && (numeric\_limits<int>::max() - result\*10) <= digit)

return numeric\_limits<int>::max();

else if(!flag && (numeric\_limits<int>::min() + result\*10) >= digit\*-1)

return numeric\_limits<int>::min();

result = result\*10+digit;

}

return flag ==1? result:-result;

}

int main()

{

string str("-4324324878787998797");

cout<<StringToInt(str)<<endl;

return 0;

}

### 字符串相乘

给定两个以字符串形式表示的非负整数 num1 和 num2，返回 num1 和 num2 的乘积，它们的乘积也表示为字符串形式。

注意：不能使用任何内置的 BigInteger 库或直接将输入转换为整数。

注：LeetCode 43

分析：

采用遍历模拟实现。

### 替换空格

### 替换数字

### 字符串去重

题目要求：在一个已经排序好的字符串中，经过处理，得到重复的字符保留一个。

代码：

#include <stdio.h>

#include <stdlib.h>

// 一个已经排序好的字符串 将重复的字符串保留一个

void Remove(char\* str)

{

int index =0;

int pos =1;

for(;str[pos]!='\0';pos++)

{

if(str[pos] != str[index])

{

index++;

str[index]= str[pos];

}

}

str[index+1] ='\0';

}

// 一个已经排序好的字符串 将重复的字符保留两个

void Remove2(char\* str)

{

int index =0;

int pos =1;

int flag =0;

for(;str[pos]!='\0';pos++)

{

//如果当前位置的字符和已经保存的字符的最后一个位置的字符不同

if(str[pos] != str[index])

{

index++;

str[index] = str[pos];

flag=0;

}

//如果当前位置的字符和已经保存的字符的最后一个位置的字符相同

else

{

if(flag == 0)

{

index++;

str[index] = str[pos];

flag =1;

}

}

}

str[index+1]='\0';

}

//重复的字符全部删除

void Remove3(char\* str)

{

int flag =0;

int index =-1;

int pos;

//遍历所有字符

for(pos=0;str[pos+1]!='\0';)

{

//如果当前字符和下一个字符相同

if(str[pos] == str[pos+1])

{

flag =1;

pos++;

}

//如果当前字符和下一个字符不同

else

{

if(flag == 1)

{

flag =0;

pos++;

}

else

{

index++;

str[index] = str[pos];

pos++;

}

}

}

if(str[pos]!= str[pos-1])

str[++index] = str[pos];

str[index+1] = '\0';

}

int main()

{

char str[]="AAABCCCDFGGIIIKKLMNNOOPQQQQ";

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

Remove3(str);

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

return 0;

}

拓展：

如何将重复的字符全部删除？

如何将重复的字符都保留两个？

### 反转字符串II

### 右旋转字符串

### 翻转字符串里的单词