# (07)－翻转句子中单词的顺序

题目：输入一个英文句子，翻转句子中单词的顺序，但单词内字符的顺序不变。句子中单词以空格符隔开。为简单起见，标点符号和普通字母一样处理。

例如输入“I am a student.” 🡪 “.tneduts a ma I” 🡪 “student. a am I”。

1. Reverse each word
2. Reverse entire sentence

由于本题需要翻转句子，我们先颠倒句子中的所有字符。这时，不但翻转了句子中单词的顺序，而且单词内字符也被翻转了。我们再颠倒每个单词内的字符。由于单词内的字符被翻转两次，因此顺序仍然和输入时的顺序保持一致。

char\* ReverseWordOrder(char\* str)

{

if (!str) return str;

int len = strlen(str); if (len < 2) return str;

ReverseString(str, str+len-1); // char-level reverse

char \*beg = str, \*end = NULL;

while (true)

{

while (\*beg != 0 && \*beg == ' ') ++beg; // find beginning of next word

if (\*beg == 0) break;

end = beg+1;

while (\*end != 0 && \*end != ' ') ++end; // find ending of the word

ReverseString(beg, end-1); // reverse the found word

if (\*end == 0) break;

else beg = end+1;

}

return str;

}

# (13)－第一个只出现一次的字符

题目：在一个字符串中找到第一个只出现一次的字符。如输入abaccdeff，则输出b。 分析：这道题是2006年google的一道笔试题。

由于题目与字符出现的次数相关，我们是不是可以统计每个字符在该字符串中出现的次数？我们只需要一个非常简单的哈希表就能满足要求。由于字符（char）是一个长度为8的数据类型，因此总共有可能256 种可能。于是我们创建一个长度为256的数组，每个字母根据其ASCII码值作为数组的下标对应数组的对应项，而数组中存储的是每个字符对应的次数。这样我们就创建了一个大小为256，以字符ASCII码为键值的哈希表。我们第一遍扫描这个数组时，每碰到一个字符，在哈希表中找到对应的项并把出现的次数增加一次。这样在进行第二次扫描时，就能直接从哈希表中得到每个字符出现的次数了。

# (17)－把字符串转换成整数

题目：输入一个表示整数的字符串，把该字符串转换成整数并输出。例如输入字符串"345"，则输出整数345。 分析：这道题尽管不是很难，学过C/C++语言一般都能实现基本功能，但不同程序员就这道题写出的代码有很大区别，可以说这道题能够很好地反应出程序员的思维和编程习惯，因此已经被包括微软在内的多家公司用作面试题。建议读者在往下看之前自己先编写代码，再比较自己写的代码和下面的参考代码有哪些不同。

* 空指针
* 正负号
* 非法字符
* 整型溢出

bool WillOverflow(int currVal, int multiplier, int addition)

{

int maxVal = std::numeric\_limits<int>::max();

if (currVal <= maxVal / multiplier)

{

if (addition <= maxVal - currVal \* multiplier)

return false;

}

return true;

}

int Str2Int(const char\* str, bool& bSuccess)

{

bSuccess = false;

if (!str || !(\*str)) { return 0; }

while (\*str && \*str == ' ') ++str; // ignore beginning whitespace(s)

if (!(\*str)) return 0;

// parse sign

bool bNegative = false;

if (\*str == '+') { ++str; }

else if (\*str == '-') { ++str; bNegative = true; }

// parse absolute value

int val = 0;

int numParsedDigits = 0;

while (\*str >= '0' && \*str <= '9')

{

int digit = (int)(\*str - '0');

if (WillOverflow(val, 10, digit)) return 0; // overflow

val = val \* 10 + digit;

++numParsedDigits;

++str;

}

if (numParsedDigits == 0) return 0;

if (\*str != 0) return 0; // invalid ending char

bSuccess = true;

return bNegative ? -val : val;

}

# (21)－左旋转字符串

题目：定义字符串的左旋转操作：把字符串前面的若干个字符移动到字符串的尾部。如把字符串abcdef左旋转2位得到字符串cdefab。请实现字符串左旋转的函数。要求时间对长度为n的字符串操作的复杂度为O(n)，辅助内存为O(1)。

void LoopMoveStringByReverse(char\* str, int step)

{

if (!str) return;

int len = strlen(str);

if (len < 2) return;

char\* end = str + len - 1;

while (step < 0) step += len;

step %= len;

if (step == 0) return;

ReverseString(str, str + step - 1);

ReverseString(str + step, end);

ReverseString(str, end);

}

# (36) －在字符串中删除特定的字符

题目：输入两个字符串，从第一字符串中删除第二个字符串中所有的字符。例如，输入”They are students.” 和”aeiou” ，则删除之后的第一个字符串变成”Thy r stdnts.” 。

* 用char[256]作为哈希表存储待删除的字符集
* O(n)复杂度的字符删除算法（从左至右）

char\* src = str; // move fast

char\* dst = str; // move slow

while (\*src)

{

if (\*src != ch) // copy if do not delete

\*dst++ = \*src++;

else ++src;

}

\*dst = '\0';

# (46)－对称子字符串的最大长度

题目：输入一个字符串，输出该字符串中对称的子字符串的最大长度。比如输入字符串“google”，由于该字符串里最长的对称子字符串是“goog”，因此输出4。

我们从里向外来判断。也就是我们先判断子字符串A是不是对称的。如果A不是对称的，那么向该子字符串两端各延长一个字符得到的字符串肯定不是对称的。如果A对称，那么我们只需要判断A两端延长的一个字符是不是相等的，如果相等，则延长后的字符串是对称的

int GetLongestSymmetricalLength(const char\* str)

{

if (!str) return -1;

const char\* end = str + strlen(str); // 超尾指针

int maxLen = -1;

const char\* pChar = str;

while (\*pChar)

{

// Substrings with odd length

const char\* pFirst = pChar - 1;

const char\* pLast = pChar + 1;

while (pFirst >= str && pLast < end && \*pFirst == \*pLast)

{ --pFirst; ++pLast; }

int len = pLast - pFirst - 1;

if (len > maxLen) maxLen = len;

// Substrings with even length

pFirst = pChar;

pLast = pChar + 1;

while (pFirst >= str && pLast < end && \*pFirst == \*pLast)

{ --pFirst; ++pLast; }

len = pLast - pFirst - 1;

if (len > maxLen) maxLen = len;

++pChar;

}

return maxLen;

}

# (28)－字符串的排列

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

void Permutation(char\* str, char\* beg)

{

if (\*beg == '\0') { printf("%s\n", str); return; }

Permutation(str, beg + 1);

for (char\* p=beg+1; \*p; ++p)

{

char tmp = \*beg; \*beg = \*p; \*p = tmp; // swap [beg] and [p]

Permutation(str, beg + 1);

tmp = \*beg; \*beg = \*p; \*p = tmp; // swap [beg] and [p] again to restore

}

}

# (59)－字符串的组合

题目：输入一个字符串，输出该字符串中字符的所有组合。举个例子，如果输入abc，它的组合有a、b、c、ab、ac、bc、abc。

void Combination(char\* str, int numCharsToSelect, vector<char>& result)

{

if (numCharsToSelect == 0)

{

vector<char>::iterator iter = result.begin();

for(; iter < result.end(); ++ iter)

printf("%c", \*iter);

printf("\n");

return;

}

if (\*str == '\0') return;

result.push\_back(\*str); // 分支1：选择当前字符

Combination(str + 1, numCharsToSelect - 1, result);

result.pop\_back(); // 回溯到未选择当前字符

Combination(str + 1, numCharsToSelect, result); //分支2：不选择当前字符

}

# (基础) 字符串拷贝

int StrCpy(char\* dst, const char\* src)

{

if (!dst || !src) return -1;

size\_t num = strlen(src) + 1; // number of chars to copy

// check memory overalp:

// should not overwrite to [src, src+num-1]

// must hold (dst+num-1 < src || dst > src+num-1)

if (dst+num-1 >= src && dst <= src+num-1)

return -2;

while (num--)

\*dst++ = \*src++;

return 0;

}

# (基础) 字符串比较

int StrCmp(const char\* str1, const char\* str2)

{

assert(str1 && str2);

while (true)

{

if (!(\*str1)) {

if (!(\*str2)) return 0;

else return -1;

}

else if (!(\*str2))

return 1;

if (\*str1 == \*str2) {

++str1; ++str2;

}

else return (\*str1 < \*str2) ? -1 : 1;

}

}

# (基础) 字符串查找

const char \* strstr ( const char \* str1, const char \* str2 );

KMP

逆波兰式

后缀数组

**单个字符串中（可重叠的）最长重复子串**

* 建立后缀数组并按字母表顺序排序
* 比较相邻后缀的公共前缀，找到最长公共前缀即为所求

单个字符串中最长回文子串

* 将整个字符串反向接到原字符串后面，中间用一个特殊字符隔开
* 从新字符串中寻找最长重复子串（基于后缀数组）

两个字符串中的最长公共子串

* 将字符串2接到字符串1后面，中间用一个特殊字符隔开
* 从新字符串中寻找最长重复子串（只考虑来自不同字符串的后缀）