中国科学技术大学
University of Science and Technology of China

程序设计II

第4讲 字符串处理

计算机学院 黄章进

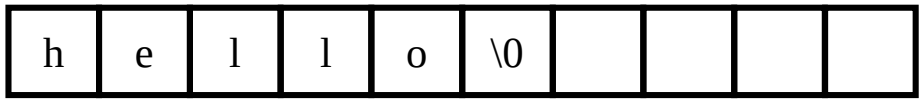
zhuang@ustc.edu.cn

内容

- 字符串处理函数
- 例题：Caesar密码 2767
- 例题：单词排序
- 例题：子串 2744
- 例题：All in All 2976

字符串

- 每个字符串是一个特殊的数组，满足两个条件
  - 元素的类型为char
  - 最后一个元素的值为'\0'，Ascii码就是 0
- 以字符型数组存储
  - 从0号元素开始存储
  - 最大可以存储长度为N-1的字符串，N是数组的大小。
- 字符串"hello"在长度为10的字符串数组中的存储



字符串的读与写

- Writing a string is easy using either `printf` or `puts`.
- Reading a string is a bit harder, because the input may be longer than the string variable into which it's being stored.
- To read a string in a single step, we can use either `scanf` or `gets`.
- As an alternative, we can read strings one character at a time.

用printf和puts写字符串

字符串的读与写

- The `%s` conversion specification allows `printf` to write a string:

```
char str[] = "Are we having fun yet?";

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

The output will be

```
Are we having fun yet?
```

- `printf` writes the characters in a string one by one until it encounters a null character.

用printf和puts写字符串

字符串的读与写

- To print part of a string, use the conversion specification `%.ps`.
- *p* is the number of characters to be displayed.
- The statement

```
printf("%.6s\n", str);
```

will print

```
Are we
```

用printf和puts写字符串

字符串的读与写

- The `%m.s` conversion will display a string in a field of size *m*.
- If the string has fewer than *m* characters, it will be right-justified within the field.
- To force left justification instead, we can put a minus sign in front of *m*.
- The *m* and *p* values can be used in combination.
- A conversion specification of the form `%m.ps` causes the first *p* characters of a string to be displayed in a field of size *m*.

用printf和puts写字符串

- `printf` isn't the only function that can write strings.
- The C library also provides `puts`:

```
puts(str);
```

- After writing a string, `puts` always writes an additional new-line character.

用scanf和gets读字符串

字符串的读与写

- The `%s` conversion specification allows `scanf` to read a string into a character array:

```
scanf("%s", str);
```

- `str` is treated as a pointer, so there's no need to put the `&` operator in front of `str`.
- When `scanf` is called, it skips white space, then reads characters and stores them in `str` until it encounters a white-space character.
- `scanf` always stores a null character at the end of the string.

用scanf和gets读字符串

字符串的读与写

- `scanf` won't usually read a full line of input.
- A new-line character will cause `scanf` to stop reading, but so will a space or tab character.
- To read an entire line of input, we can use `gets`.
- Properties of `gets`:
  - Doesn't skip white space before starting to read input.
  - Reads until it finds a new-line character.
  - Discards the new-line character instead of storing it; the null character takes its place.

用scanf和gets读字符串

- Consider the following program fragment:

```
char sentence[SENT_LEN+1];

printf("Enter a sentence:\n");
scanf("%s", sentence);
```

- Suppose that after the prompt

```
Enter a sentence:
```

the user enters the line

```
To C, or not to C: that is the question.
```

- `scanf` will store the string `"To"` in `sentence`.

用scanf和gets读字符串

- Suppose that we replace `scanf` by `gets`:

```
gets(sentence);
```

- When the user enters the same input as before, `gets` will store the string

```
" To C, or not to C: that is the question."
```

in `sentence`.

用scanf和gets读字符串

字符串的读与写

- As they read characters into an array, `scanf` and `gets` have no way to detect when it's full.
- Consequently, they may store characters past the end of the array, causing undefined behavior.
- `scanf` can be made safer by using the conversion specification `%`*n*`s` instead of `%s`.
- *n* is an integer indicating the maximum number of characters to be stored.
- `gets` is inherently unsafe; `fgets` is a much better alternative.

字符串处理函数

- #include <string.h>
- 将格式化数据写入字符串： sprintf
- 字符串长度查询函数： strlen
- 字符串复制函数：strcpy、strncpy
- 字符串连接函数： strcat
- 字符串比较函数： strcmp、strncmp、_stricmp、_strnicmp
- 字符串搜索函数： strcspn、strspn、strstr、strtok、strchr
- 字符串大小写转换函数： strlwr、strupr

字符串拷贝和求字符串长度

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

int strlen(const char *s);

```
#include <stdio.h>
#include <string.h>
int main()
{
        char str1[10]="hello ", str2[12];
        strcpy(str2, "hello world");
        printf("length: %d(str1); %d(str2)\n", strlen(str1), strlen(str2));
        strcpy(str1, str2);
        printf("length: %d(str1); %d(str2)\n", strlen(str1), strlen(str2));
        printf("%s\n", str1);
        return 0;
}
```

把"hello world"复制到str2

查询str1中字符串的长度

把str2复制到str1

strcpy

- 输出结果：

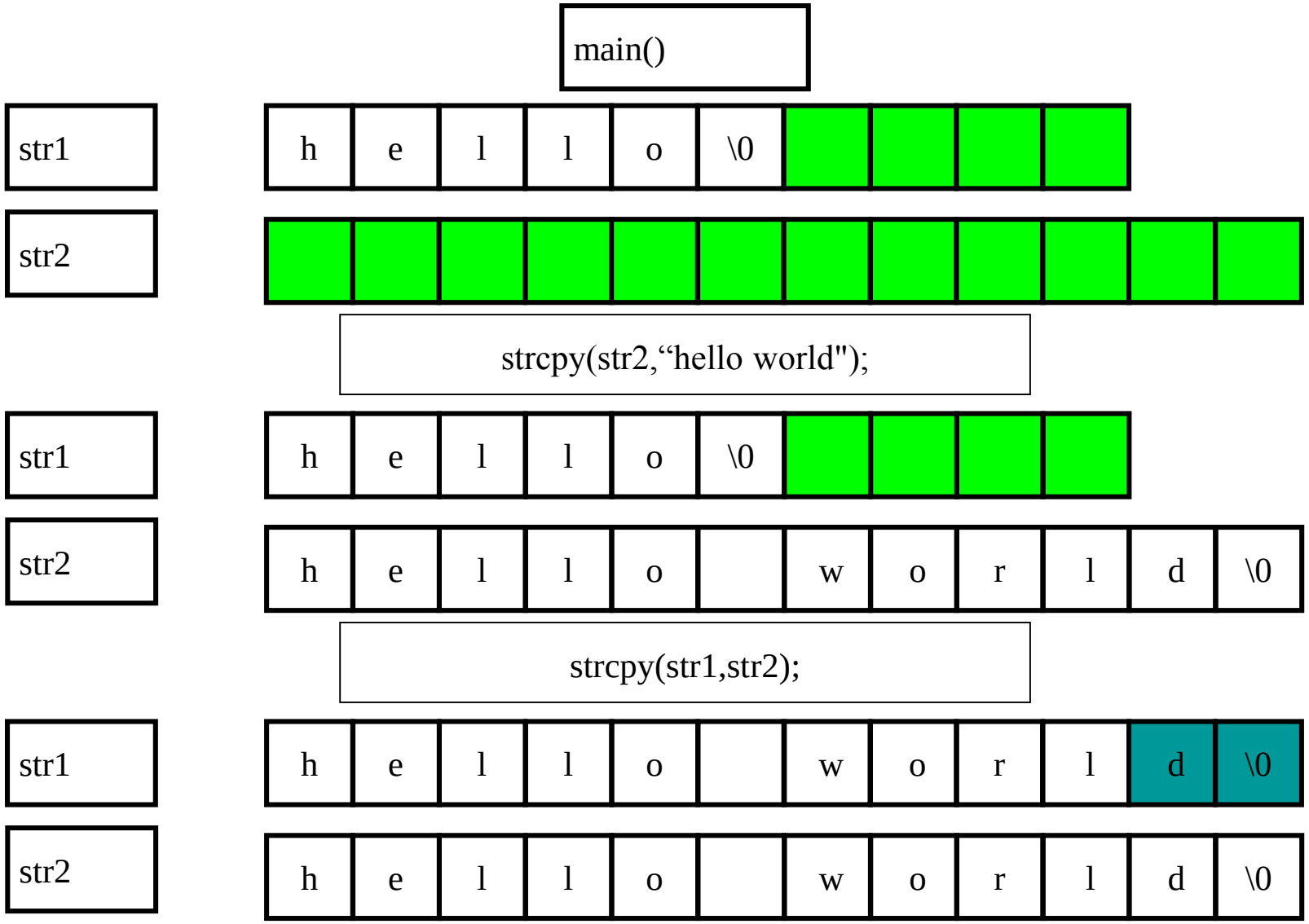
length: 5(str1); 11(str2)

length: 11(str1); 11(str2)

hello world

- str1存储了11个非'\0'字符？
  - strcpy在复制字符串str2到str1时，不检查str2是否超出了str1的存储容量，而是直接将str2中存储的字符串复制到从str1开始的一段连续区域
  - 在程序中要特别注意这种情况所引发的程序运行不确定性

strcpy


main()
str1
h e l l o \0
str2
strcpy(str2,“hello world");
str1
h e l l o \0
str2
h e l l o w o r l d \0
strcpy(str1,str2);
str1
h e l l o w o r l d \0
str2
h e l l o w o r l d \0

用strlen时常犯的错误


```
int MyStrchr(char *s,  char c)   //看s中是否包含 c
{
    int i;
    for ( i = 0; i < strlen(s) ; i ++ )
        if ( s[i] ==  c)
            return 1;
    return 0;
}
```

哪里不好？这个函数执行时间和 s 的长度是什么关系？

strlen 是一个O(N)的函数，每次判断 i < strlen(s) 都要执行，太浪费时间了

字符串添加strcat

char *strcat(char *dest, const char *src);

- 把src内容加到dest后面，同样不会考虑 dest是否够长

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str1[100]="hello", str2[10]="^_^";
    strcat(str1, " world ");
    printf("%s\n", str1);
    strcat(str1, str2);
    printf("%s\n", str1);
    return 0;
}
```

把" world "添加到str1中原字符串的末尾

把str2中的字符串添加到str1中原字符串的末尾

输出：

```
hello world
hello world ^_^
```

字符串比较函数

int strcmp(const char *s1, const char *s2);

- 区分大小写

If s1 is...	return value is...
less than s2	< 0
the same as s2	== 0
greater than s2	> 0

int _stricmp(const char *s1, const char *s2);

- 不分大小写，不是标准C库函数

字符串比较函数


```
#include <string.h>
#include <stdio.h>
char string1[] = "The quick brown dog jumps over the lazy fox";
char string2[] = "The QUICK brown dog jumps over the lazy fox";
int main( void )
{
   int result;
   printf( "Compare strings:\n\t%s\n\t%s\n\n", string1, string2 );
   result = strcmp( string1, string2 );
   printf( "strcmp: result=%d\n", result);
   result = _stricmp( string1, string2 );
   printf( "stricmp: result=%d\n", result);
   return 0;
}
```

字符串比较函数

- 输出：

Compare strings:

The quick brown dog jumps over the lazy fox

The QUICK brown dog jumps over the lazy fox

strcmp: result=1

stricmp: result=0

查找子串strstr

char *strstr(char *s1, char *s2);

- 查找给定字符串在字符串中第一次出现的位置，返回位置指针
  - 如果找到，返回指针，指向s1中第一次出现s2的位置
  - 如果找不到，返回 NULL

查找子串strstr


```
#include <string.h>
#include <stdio.h>
char str[] =    "lazy";
char string[] = "The quick brown dog jumps over the lazy fox";
int main( void )
{
   char *pdest;
   int  result;
   pdest = strstr( string, str );
   result = pdest - string + 1;
   if ( pdest != NULL )
      printf( "%s found at position %d\n\n", str, result );
   else
      printf( "%s not found\n", str );
   return 0;
}
```

输出：

lazy found at position 36

查找字符strchr

char *strchr(char *s, int c);

- 查找给定字符在字符串中第一次出现的位置，返回位置指针
  - 如果找到，返回指针，指向c在s中第一次出现的位置
  - 如果找不到，返回 NULL

查找字符strchr


```
#include <string.h>
#include <stdio.h>
int  ch = 'r';
char string[] = "The quick brown dog jumps over the lazy fox";
int main( void )
{
   char *pdest;
   int result;
   pdest = strchr( string, ch );
   result = pdest - string + 1;
   if( pdest != NULL )
      printf( "Result:\tfirst %c found at position %d\n\n",  ch, result );
   else
      printf( "Result:\t%c not found\n" );
   return 0;
}
```

输出：

```
Result:   first r found at position 12
```

字符串部分拷贝strncpy

char *strncpy(char *dest, char *src, int maxlen);

- 将前 maxlen 个字符从src拷贝到dest
  - 如果src中字符不足 maxlen 个，则连'\0'一起拷贝，'\0'后面的不拷贝
  - 如果src中字符大于等于maxlen个，则拷贝maxlen个字符
  - 返回值：dest的地址

字符串部分拷贝strncpy


```
#include <string.h>
#include <stdio.h>
int main(void)
{
        char s1[20] = "1234567890";
        char s2[] =  "abcd" ;
        strncpy( s1, s2, 5);
        printf("%s\n", s1);
        strcpy( s1, "1234567890");
        strncpy( s1, s2, 4);
        printf("%s\n", s1);
        return 0;
}
```

输出：

abcd

abcd567890

数组作为函数参数


```
#include <stdio.h>
char str1[200] = "Hello, World";
char str2[100] = "Computer";
void swap( char s1[ ], char *s2)  //交换两个字符串的内容
{
    char c;
    int i;
    for ( i = 0; s1[i] || s2[i]; i++ ){  //  '\0'的Ascii 码就是 0
        c = s2[i];
        s2[i] = s1[i];
        s1[i] = c;
    }
    s1[i+1] = s2[i+1] = 0;
}
int main()
{
    swap(str1, str2);
    printf("%s\n%s\n", str1, str2);
    return 0;
}
```

输出：

Computer

Hello, World

例题：Caesar密码

- 问题描述
  - Julius Caesar 生活在充满危险和阴谋的年代。为了生存，他首次发明了密码，用于军队的消息传递
  - 假设你是Caesar 军团中的一名军官，需要把Caesar 发送的消息破译出来
  - 消息加密的办法：对消息原文中的每个字母，分别用该字母之后的第5个字母替换（例如：消息原文中的每个字母A都分别替换成字母F，V替换成A, W替换成B…），其他字符不变，并且消息原文的所有字母都是大写的

Caesar密码

密码字母：A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
原文字母：V W X Y Z A B C D E F G H I J K L M N O P Q R S T U

- 输入
  - 最多不超过100个数据集组成。每个数据集由3部分组成
    - 起始行：START
    - 密码消息：由1到200个字符组成一行，表示Caesar发出的一条消息
    - 结束行：END
  - 在最后一个数据集后，是另一行：ENDOFINPUT
- 输出
  - 每个数据集对应一行，是Caesar 的原始消息

Caesar密码

- 输入样例

START

NS BFW, JAJSYX TK NRUTWYFSHJ FWJ YMJ WJXZQY TK YWNANFQ HFZXJX

END

START

N BTZQI WFYMJW GJ KNWXY NS F QNYYQJ NGJWNFS ANQQFLJ YMFS XJHTSI NS WTRJ

END

START

IFSLJW PSTBX KZQQ BJQQ YMFY HFJXFW NX RTWJ IFSLJWTZX YMFS MJ

END

ENDOFINPUT

- 输出样例

IN WAR, EVENTS OF IMPORTANCE ARE THE RESULT OF TRIVIAL CAUSES

I WOULD RATHER BE FIRST IN A LITTLE IBERIAN VILLAGE THAN SECOND IN ROME

DANGER KNOWS FULL WELL THAT CAESAR IS MORE DANGEROUS THAN HE

Caesar密码

问题分析

- 关键是识别输入数据中的消息行、读入消息行的数据
- 输入数据中，每个消息行包括多个单词、以及若干个标点符号
  - 识别非字母符号
- 采用哪种输入函数，可以用scanf吗？

2767 Caesar密码


```
#include<stdio.h>
#include<string.h>
int main(void)
{
    char s[201];
    char map[] = "VWXYZABCDEFGHIJKLMNOPQRSTU";
    while(1) {
        gets(s);
        if (strcmp(s, "ENDOFINPUT")==0)       break;
        if (strcmp(s, "START")==0 || strcmp(s, "END")==0)         continue;
        int len = strlen(s);
        for (int i = 0; i < len; i++) {
            if (s[i] >= ‘A’ && s[i] <= ‘Z’)     // 请写出下一行代码

            else
                printf("%c", s[i]);
        }
        printf("\n");
    }
    return 0;
}
```

例题：单词排序

- 输入若干行单词（不含空格），请按字典序排序输出。大小写有区别。单词一共不超过100行，每个单词不超过20字符。

- 输入样例

What
man
Tell
About
back

- 输出样例

About
Tell
What
back
man

单词排序

- 用什么保存多个单词？
  - 采用二维字符数组

  char Word[100][21];

  则表达式 Word[i] 的类型就是 char[21]

  Word[i] 就是数组中的一行，就是一个字符串

  Word[i][0] 就是Word[i] 这个字符串的头一个字符
- 如何排序？
  - qsort

qsort函数

- qsort函数是ANSI C标准库中提供的可给任意数组排序的通用函数，声明在stdlib.h文件中
- 因为数组元素可能是任何类型的，必须告诉qsort如何确定两个数组元素哪一个“更小”
- 通过传递给qsort一个比较函数指针

qsort函数


```
void qsort( void *base, size_t nelem, size_t size,
            int (*comp)(const void *,const void *) );
```

- 对数组按升序排序
  - base 指向待排序数组的第一个元素
  - nelem 待排序元素的个数
  - size 数组元素的大小（字节数）
  - comp 指向比较函数的指针
    - int comp(const void *a, const void *b);
    - *a “小于” *b，返回负整数值
    - *a “等于” *b，返回0
    - *a “大于” *b，返回正整数值

qsort函数


```
#include <stdio.h>
#include <stdlib.h>
int compare_ints(const void* a, const void* b)
{
    int arg1 = *(const int*)a;
    int arg2 = *(const int*)b;
    return (arg1 - arg2);
}
int main(void)
{
    int i, ints[] = { -2, 99, 0, -743, 2, 3, 4 };
    int size = sizeof ints / sizeof *ints;
    qsort(ints, size, sizeof(int), compare_ints);
    for (i = 0; i < size; i++) {
        printf("%d ", ints[i]);
    }
     printf("\n");
  return EXIT_SUCCESS;
}
```

输出：

-743 -2 0 2 3 4 99

单词排序


```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int MyCompare( const void *e1, const void *e2 )
{  // 请写出下一行代码

}
int main()
{
    int n = 0; //单词个数
    char Words[100][21];
    while (scanf("%s", Words[n]) != EOF && Words[n][0])
        n++;
    qsort(Words, n, sizeof(Words[0]), MyCompare);
    for ( int i = 0; i < n; i++ )
        printf("%s\n", Words[i]);
    return 0;
}
```

为了处理有可能最后一行读入的是空行

例题：子串

- 问题描述
  - 现有一些由英文字符组成的大小写敏感的字符串的集合s，请找到一个最长的字符串x，使得对于s中任意字符串y，x或者是y的子串，或者x中的字符反序之后得到的新字符串是y的子串。
- 输入
  - 第一行是一个整数t (1 <= t <= 10)，t表示测试数据组的数目
  - 对于每一组测试数据，第一行是一个整数n (1 <= n <= 100)，表示给出n个字符串。接下来n行，每行给出一个长度在1和100之间的字符串。
- 输出
  - 对于每一组测试数据，输出一行，给出题目中要求的字符串x的长度。

子串

- 输入样例

2
3
ABCD
BCDFF
BRCD
2
rose
orchid

- 输出样例

2
2

子串

- 思路：
  - 随便拿出输入数据中的一个字符串，从长到短找出它的所有子串，直到找到否符合题目要求的子串。
- 改进：
  - 不要随便拿，要拿输入数据中最短的那个。从长到短找出它的所有子串，直到找到否符合题目要求的子串。

2744 子串


```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
int searchMaxSubString(char* source);
int n;  char str[100][101];
int main(){
    int i, t, minStrLen, subStrLen;
    char minStr[101];  // 最短字符串
    scanf("%d", &t);
    while(t--) {
        scanf("%d", &n);
        minStrLen = 100; //记录输入数据中最短字符串的长度
        for (i = 0; i < n; i++) { //输入一组字符串
            scanf("%s", str[i]);
            if ( strlen(str[i]) < minStrLen ) { //找其中最短字符串
                strcpy(minStr, str[i]);
                minStrLen = strlen(minStr);
            }
        }
        subStrLen = searchMaxSubString(minStr); //找答案
        printf("%d\n", subStrLen);
    }
    return 0; }
```

子串


```
int searchMaxSubString(char* source){
   int subStrLen = strlen(source), sourceStrLen = strlen(source);
   int i, j;
   char subStr[101], revSubStr[101];
   while ( subStrLen > 0 ) { //搜索不同长度的子串，从最长的子串开始搜索
      for (i = 0; i <= sourceStrLen - subStrLen; i++) { //搜索长度为subStrLen的全部子串
         strncpy(subStr, source+i, subStrLen);
         strncpy(revSubStr, source+i, subStrLen);
         subStr[subStrLen] = revSubStr[subStrLen] = '\0';
         strrev1(revSubStr);    // 将字符串反序，也可调用非标准C库函数 _strrev
         bool foundMaxSubStr = true;
         for (j = 0; j < n; j++)  //遍历所有输入的字符串
            if (strstr(str[j], subStr) == NULL && strstr(str[j], revSubStr) == NULL ) {
               foundMaxSubStr = false;          break;
            }
         if (foundMaxSubStr)  return subStrLen;
      }
      subStrLen--;
   }
   return 0;  }
```

子串


```
char *strrev1(char *source)
{
    char temp[200];
    int i, len = strlen(source);

    strcpy(temp, source);
    for(i = 0; i < len; i++)
        source[i] = temp[len-1-i];

    return source;
}
```

例题：All in All

- 问题描述
  - 给定两个字符串s和t，请判断s是否是t的子序列。即从t中删除一些字符，将剩余的字符连接起来，即可获得s。
- 输入
  - 包括若干个测试数据。每个测试数据由两个ASCII码的数字和字母串s和t组成， s和t的长度不超过100000。
- 输出
  - 对每个测试数据，如果s是t的子序列则输出"Yes"，否则输出"No"

All in All

- 输入样例：s t

sequence subsequence

person compression

VERDI vivaVittorioEmanueleReDiItalia

caseDoesMatter CaseDoesMatter

- 输出样例

Yes

No

Yes

No

All in All

- 思路
  - 关键是看t中是否可以找到s的所有字符，而且顺序与s一致。
- 设两个指针，分别指向s和t的开头。
  - t的指针不停向前每次移动一个字符，如果t的指针指向的字符和s的指针指向的字符相同，则将s的指针向前移动一个字符。
  - 直到s的指针指向末尾的'\0'，说明Yes；
  - 或 t 的指针指向末尾的'\0'时，s的指针还没有指到末尾，说明No

2976 All in All


```
#include <stdio.h>
int main()
{
    int i, j;  // i指向s, j指向t
    char s[100001], t[100001];
    while (scanf("%s%s", s, t) > 0 ) {  // 请写出循环体代码




    }
    return 0;
}
```

bsearch函数

- C语言中可以用bsearch实现二分查找。
- 同qsort一样，bsearch也包含在<stdlib.h>库中，且同样要自定义比较函数
- bsearch函数在有序数组中搜索一个特定的值（键）
  - qsort函数可以对任何数组进行排序
  - 可以在bsearch函数搜索数组之前先用qsort函数对其进行排序

bsearch函数


```
void* bsearch( const void *key, const void *base, size_t nelem, size_t size,
        int (*comp)(const void*, const void*) );
```

- 在有序数组中查找值等于*key的元素
  - key 指向键值
  - base 指向待查找数组的第一个元素
  - nelem 待查找元素的个数
  - size 数组元素的大小（字节数）
  - comp 指向比较函数的指针
    - 待查找数组必须是已按照comp规则“升序”排序
  - 返回值：一个指向与键匹配的元素的指针；找不到，则返回空指针
    - 若有多个元素与键值相等，返回哪个元素的指针是未定的

bsearch函数


```
#include <stdio.h>
#include <stdlib.h>
int compare(const void *p, const void *q)
{
    return (*(int *)p - *(int *)q);
}
int main(void)
{
    int array[8] = {9, 2, 7, 11, 3, 87, 34, 6};
    int key = 3;
    int *p;
    qsort(array, 8, sizeof(int), compare);
    p = (int *) bsearch(&key, array, 8, sizeof(int), compare);
    (p == NULL) ? puts("not found") : puts("found");
    return 0;
}
```

输出：
found

bsearch函数


```
#include <stdlib.h>
#include <stdio.h>
struct data {
    int nr;
    char const *value;
} dat[] = {
    {1, "Foo"}, {2, "Bar"}, {3, "Hello"}, {4, "World"}
};
int data_cmp(void const *lhs, void const *rhs)
{
    struct data const *const l = lhs;
    struct data const *const r = rhs;
    return (l->nr > r->nr) - (l->nr < r->nr);  // 返回1, 0 or -1
}
```

bsearch函数


```
int main(void)
{
    struct data key = { .nr = 3 };
    struct data const *res = bsearch(&key, dat, sizeof(dat)/sizeof(dat[0]),
                                     sizeof(dat[0]), data_cmp);
    if (!res) {
        printf("No %d not found\n", key.nr);
    } else {
        printf("No %d: %s\n", res->nr, res->value);
    }
}
```

输出：

No 3: Hello

词典作业提示

- 采用bsearch查找函数
- 怎么判断读到空行？

char s[100];

可以用 gets(s)一次读取一行，然后再用sscanf从s中分出英文和外文

gets(s)读到空行时，s的长度为 0，即 s[0] = 0

读到文件尾时，gets返回 NULL

scanf 返回 EOF也可以说明到了文件尾巴

注意：文件尾可能有空行