# 关键字检索系统

## 一.项目背景：

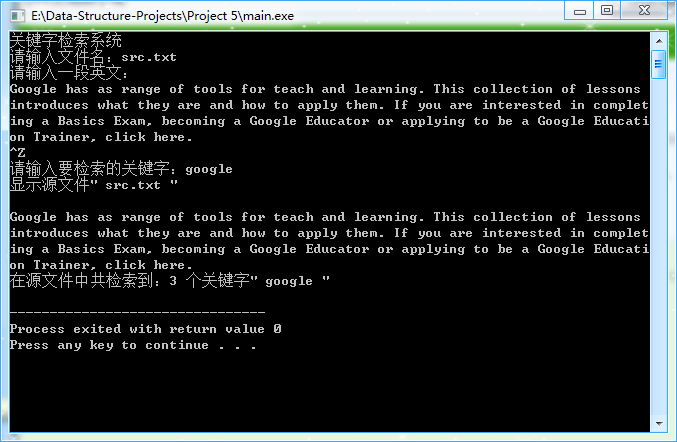
建立一个文本文件，文件名由用户用键盘输入，输入一个不含空格的关键字，统计输出关键字在文本中的出现次数。

## 二.项目功能

本项目的设计要求可以分成两个部分实现：首先建立一个文本文件，文件名由用户用键盘输入；然后输入一个不含空格的关键字，统计输出该单词在文本中的出现次数。

**PS：本程序忽略了大小写，输入数据以ctrl+z结尾，具体示例见下图**

项目示例：



## 三.程序描述

本题十分水，本来可以只使用一个内置函数：strstr 就可以完成本程序。但是为了锻炼我的算法能力，我选择了稍微难一点的KMP算法，时间复杂度为O(n)，具体思路如下：

## KMP算法

   KMP 匹配算法是由 "Knuth  Morris  Pratt"  提出的一种快速的模式匹配算法。    
   hint：不为自身的最大首尾重复子串长度

   1.待解决的问题：假设P为给定的子串，T是待查找的字符串，要求从T中找出与P相同的所有子串，这称为模式匹配问题。 (可以给出子串在T中的位置) (下文中提到的P和T分别为子串和目标串)

   让我们先来看个例题：

   T:   t0      t1     t2      t3 .... tm-1 ... tn-1

   P:   p0      p1     p2      p3 .....pm-1

   从T的最左边开始比较，使得 TK = PK，则匹配成功。

   2.解决模式匹配问题的方案：

   A：朴素的模式匹配算法(思路简单,但不够简便，时间长，有回溯)：最简单和最直接的做法，用P中的字符依次与T中的字符进行比较，遇到不相等的字符，则可将P右移一个字符，重新进行比较，直到某次匹配成功或者到达P的最右字符移出T为止。

   如：若P="aaaba", T="aaabbaaaba", 则匹配过程如下图

    T:     a   a   a   b   b   a   a   a   b  a

    P:     a   a   a   b   a

               a   a   a   b   a

                                   .....

                               a   a   a   b  a

   从上不难分析，最坏的情况是“每次比较都在最后一个字符出现不等，每趟最多比较M次，最多比较N-M+1趟，总的比较次数最多为M\*(N-M+1)” ，时间复杂性为0(M\*N)。 在P右移一位时，不管上一趟比较的中间结果是什么，因此回溯是不可避免的(如：前3个aaa 不需要一位一位的移 ) 。下面我来介绍无回溯的KMP算法。

   3.KMP算法解决匹配中哪些主要问题：

   A.当字符串比较出现不等时，确定下一趟比较前，应该将P右移多少个字符；

   B. P右移后，应该从哪个字符开始和T中刚才比较时不等的那个字符继续开始比较。

    我们通过朴素模式匹配的例子来引出问题。在第一次比较过程中失败的是P的第4个字符b，这表明P的前4个字符是成功的。模式P的第3个字符b在它的前3个字符(aaa)中并未出现。因此，在下一次比较时候，至少要将P向后移4个字符；再看P的第一个字符与最后一个字符是相同的，因此将P右移4个字符后，再从第一个字符比较，肯定也是不等的。综上所诉：应该将P右移5个字符，再从P的第0个字符和T的第5个字符开始比较！

   KMP算法核心：KMP算法借助于一个辅助数组next来确定当匹配过程中出现不等时，模式P右移的位置和开始比较的位置。next[i]的取值只与模式P本身的前i+1项有关，而与目标T无关。匹配过程中遇到Pi不等于Tj时，若next[i]>=0，则应将P右移i-next[i]位个字符，用P中的第next[i]个字符与Tj 进行比较；若：next[i]= -1，P中的任何字符都不必再与Tj比较，而应将P右移i+1个字符，从P0和Tj+1从新开始下一轮比较(可能不太好理解，自己找个例子，对着话一句一句试试看)

   因此只要计算出与模式P相关的next数组，按上面的含义，就可以很容易地给出串的匹配算法。(问题就这样转化了)

    C.next的计算：以P = " 01001010100001"为例。

     i   :            0   1   2   3   4   5   6    .....

     P   :            0   1   0   0   1   0   1    .....

    j(next[i]) :     -1   0   0   1   1   2   3    .....

   如1：我们要算next[2]的值,有关的为P本身的前2个字符0,1。在字符串01中，寻找出“左右相同的最大字符串，此字符串所含字符的个数就为next[i]的值”而0不等于1，相同字符串不存在，所以next[i] = 0；

   如2：我们要算next[6]的值，有关的为P本身前6个字符010010 。此字符串中010 = 010左右相同的最大字符串为010，个数为3。所以next[i]=3；

   如3：我们要算next[5]的值，有关的为P本身前5个字符01001。此字符串中 01=01 左右相同的最大字符串为01，个数为2。所以next[i]=2；

在本程序中，相关变量含义如下：

/\*\*

\*\*a是要找的文本

\*\*b是关键字

\*\*text是保存在文本中的文本，它的值等于a

\*\*temp是用于读取文本时的缓冲区

\*\*p是kmp算法的初始化数组

\*/

char b[100],temp[10000],text[10000],a[10000];

int p[100];

以下函数计算next数组：

void initial()

{

p[0]=-1;

j=-1;

for(i=1;i<m;i++)

{

while (j>=0&&b[j+1]!=b[i])

j=p[j];

if (b[j+1]==b[i])

j++;

p[i]=j;

}

}

以下函数进行字符串匹配

int find()

{

int i,j,sum=0;

j=-1;

for (i=0;i<n;i++)

{

while(j>=0&&b[j+1]!=a[i])

j=p[j];

if (b[j+1]==a[i])

j++;

if (j==m-1)

{

// return (i-m+1);

sum=sum+1;

j=p[j];

}

}

return sum;

}