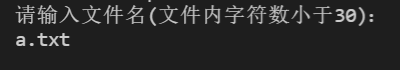
测试用例

ghjkl ghjkl

qwertt ertyui

rtuiok dfghjkl

fgh fgh



数据结构设计

功能由两个函数实现，statistics\_file\_word 函数统计文本中单词种类、出现次数、出现位置、频度；sorting 函数对统计到的单词进行字典排序

建立结构体，统计一个单词的信息，打开文件，建立结构体数组，调用以上两个函数，关闭文件。

statistics\_file\_word 函数

利用一个主串判断行数，子串判断单词是否重复和是本行第几个单词。

将文本中的字母依次赋值给子串并，若为空格，说明已输入一个单词，下一次输入前，单词数加一，若为换行符，说明一行已经输入完毕，下一次输入前，行数加一。

将子串和结构体数组中每个单词比较，若相同，该单词出现次数加一，不同，加入结构体数组中，记录单词，行数等信息。

重复以上过程，指导读取到文件末尾，跳出循环。

依次根据总次数和单词出现次数，计算每个单词的频度。

返回单词种类数目。

sorting 函数

对结构体数组进行选择排序。

从第一个单词的第一个字母开始比较，字母顺序在前的，和第一个单词交换，直到比较完所有单词。

再从第二个单词依次比较，直到末尾。

重复，直到第n – 1个单词也同第n个单词比较过。

//统计文本文件的所有单词（每个单词不包括空格及跨行，单词由字符序列构成且不区分大小写），字典顺序排列

//统计每个单词在文本文件中出现的总次数、频度（总次数/所有单词出现次数之和）

//检索输出某单词在文本文件中首次出现的行号和位置

#include <errno.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define FILE\_NAME\_LEN 30

#define TEXT\_CONTENT 50

#define TEXT\_WORD 50

#define WORD\_NUM 20

typedef struct list\_word {

  char word[WORD\_NUM]; //记录这个单词

  float frequency;       //该单词的频度

  int tag; // tag == 1(这个结构体有记录信息) tag == 0(无信息)

  int total;  //每个单词出现的总量

  int order, row; //row 为行，order 为当前行的第几个单词

} List\_word;

int statistics\_file\_word(List\_word lword[], FILE \*fp); //统计文本文件的所有单词 频度 首次出现位置

void sorting(List\_word lword[], FILE \*fp, int sum); //字典排序

int main() {

  int total\_word\_num; // 是所有单词种类

  List\_word lword[TEXT\_WORD];

  //对结构体数组进行初始化

  int i = 0;

  while (i < TEXT\_WORD) {

    lword[i].tag = 0;

    lword[i].total = 0;

    i++;

  }

  // open file

  char file\_name[FILE\_NAME\_LEN];

  printf("请输入文件名(文件内字符数小于%d)：\n", FILE\_NAME\_LEN);

  scanf("%s", file\_name);

  FILE \*fp = fopen(file\_name, "r");

  if (fp == NULL) {

    printf("打开文件失败。");

    exit(-1);

  }

  total\_word\_num = statistics\_file\_word(lword, fp);

  sorting(lword, fp, total\_word\_num);

  fclose(fp);

  return 0;

}

int statistics\_file\_word(List\_word lword[], FILE \*fp) {

  int i = 0, loc = 0, // loc 为当前单词记录到的位置，i 计数结构体(单词的总数),

      sum = 0,          // 是所有单词出现的总次数

      n, m = 0;         // n 子串，m 主串

  char c[WORD\_NUM]; //子串

  char a[TEXT\_CONTENT]; //主串

  int row = 0, order = 0;

  while (1) {

    n = 0;

    if (a[m - 1] == '\n') {

      m = 0;

      row++;

      order = 0;

    }

    if ((c[n] = fgetc(fp)) == -1 || a[m - 1] == -1)

      break;

    else

      a[m] = c[n];

    while (c[n] != ' ' && c[n] != '\n') {

      n++, m++;

      if ((a[m] = c[n] = fgetc(fp)) == -1)

        break;

    }

    c[n] = '\0';

    i = 0; //新输入的单词和已输入的挨个比较

    while (lword[i].tag == 1) {

      if (strcmp(c, lword[i].word) == 0) { //有匹配单词

        lword[i].total++;

        sum++;

        i = loc; // 目的跳出循环

        loc--;   //使重复的单词不再进入结构体中

      }

      i++;

    }

    //没有匹配单词

    if (lword[loc].tag == 0) {

      strcpy(lword[loc].word, c);

      lword[loc].tag = 1;

      lword[loc].total++;

      sum++;

      lword[loc].order = order;

      lword[loc].row = row;

    }

    loc++;

    order++;

    m++;

  }

  //计算频度

  for (i = 0; i < loc; i++) {

    lword[i].frequency = lword[i].total \* 1.0 / sum;

  }

  return loc;

}

void sorting(List\_word lword[], FILE \*fp, int sum) {

  List\_word temp;

  int i\_1 = 0, i\_2 = 0, j = 0;

  for (; i\_1 < sum; i\_1++) {

    for (; i\_2 < sum; i\_2++) {

      if (strcmp(lword[i\_1].word, lword[i\_2].word) > 0) {

        temp = lword[i\_1];

        lword[i\_1] = lword[i\_2];

        lword[i\_2] = temp;

      }

    }

  }

}