中南大学

数据结构试验报告

题 目 实验三

学生姓名 王云鹏

学 号 8213180228

指导老师 郑 瑾

学 院 计算机学院

专业班级 物联网1802

完成时间 2020.6

指导老师评定 签名

实验三

1. 需求分析

本实验的目的是使读者熟悉串类型的实现方法和文本模式匹配方法，熟悉一般文字处理软件的设

计方法，较复杂问题的分解求精方法，在实验二的基础上，进一步强化这样一个观念：程序是数据结

构结合定义在其上的操作。此外还希望起到训练合作能力和熟悉文件操作的目的。本实验的难度较大。

1．文学研究助手（设计性实验）

问题描述

文学研究人员需要统计某篇英文小说中某些形容词的出现次数和位置。试写一个实现这一目标的

文字统计系统，称为“文学研究助手”。

基本要求

英文小说存于一个文本文件中。待统计的词汇集合要一次输入完毕，即统计工作必须在程序的一

次运行之后就全部完成。程序的输出结果是每个词的出现次数和出现位置的行号，格式自行设计。

测试数据

以源程序模拟英文小说，编程语言保留字集作为待统计的词汇集。

实现提示

设小说非空且以文件形式存放，其中的词汇一律不跨行。这样，每读入一行，就统计每个词在这

行中的出现次数和出现位置的行号，后者可以用链表存储。若某行中出现了不止一次，不必存多个相

同的行号。数据结构采用二维链表，单词结点链接成一个链表，每个单词的行号组成一个链表，单词

结点作为行号链表的头结点。

选作内容

(1) 模式匹配要基于KMP算法。

(2) 整个统计过程中只对小说文字扫描一遍以提高效率。

(3) 假设小说中的每个单词或者从行首开始，或者前置以一个空格符。利用单词匹配特点另写一

个高效的统计程序，与KMP算法统计程序进行效率比较。

(4) 推广到更一般的模式匹配问题，并设待查模式串可以跨行

思考题

怎样考虑分词问题？怎样考虑多模式匹配问题？

2．简单行编辑程序（综合性实验）

问题描述

文本编辑程序是利用计算机进行文字加工的基本软件工具，实现对文本文件的插入、删除等修改

操作。限制这些操作以行为单位进行的编辑程序称为行编辑程序。

被编辑的文本文件可能很大，全部读入编辑程序的数据空间（内存）的做法既不经济也不总能实

现。一种解决方法是逐段地编辑，任何时刻只把待编辑文件的一段放在内存，称为活区。试按照这种

方法实现一个简单的行编辑程序。设文件每行不超过320个字符，并且很少超过80字符。

基本要求

实现以下4条基本编辑命令。

(1) 行插入。格式：i<行号><回车><文本><回车>。将<文本＞插入活区中第<行号＞行之后。

(2) 行删除。格式：d<行号1>[□ <行号2>]<回车>。删除活区中第<行号1＞行（到第<行号2＞行）。

两种格式的例子如“d10↙”和“d10□ 14↙”。

(3) 活区切换。格式：n<回车>。将活区写入输出文件，并从输入文件中读入下一段，作为新的活

区。

(4) 活区显示。格式：p<回车>。逐页地（每页20行）显示活区内容，每显示一页之后请用户决定

是否继续显示以后各页（如果存在）。印出的每一行要包含该行的行号和一个空格符，行号固定占4位，

增量为1。

各条命令中的行号均须在活区中各行行号范围之内，只有插入命令的行号可以等于活区第一行行

号减1，表示插入当前屏幕中第一行之前，否则命令参数非法。

测试数据

由读者依据软件工程的测试技术自己确定。注意测试边界数据，如首行、尾行。

实现提示

(1) 设活区的大小用行数activemaxlen（可设为100）来描述。考虑到文本文件行长通常为正态分

布，且峰值为60～70，用320×activemaxlen大小的字符数组实现存储将造成大量浪费。可以以标准行

块为单位为各行分配存储，每个标准行块含81个字符。这些行块可以组成一个数组，也可以利用动态

链表连接起来。一行文字可能占多个行块。行尾可用一个特殊的ASCII字符（如012）标识。此外，还

应记住活区起始行号，因为行插入将引起随后各行行号的顺序下推。

(2) 初始化过程包括请用户提供输入文件名（空串表示无输入文件）和输出文件名，两者不

能相同。然后尽可能多地从输入文件中读入各行，但不超过activemaxlen−x。x的值可以自定，

例如20。

(3) 在执行行插入命令的过程中，每接收到一行时要检查活区大小是否已达到activemaxlen。如

果是，则为了在插入这一行之后仍保持活区大小不超过activemaxlen，应将插入点之前活区部分的第

一行输出到输出文件中。若插入点为第一行之前，则只得将新插入的这一行输出。

(4) 若输入文件尚未读完，活区切换命令可将原活区中最后几行留在活区顶部，以保持阅读连续

性；否则，它意味着结束编辑或开始编辑另一个文件。

(5) 可令前3条命令执行后自动调用活区显示。

选作内容

(1) 对于命令格式非法等一切错误做严格检查和适当处理。

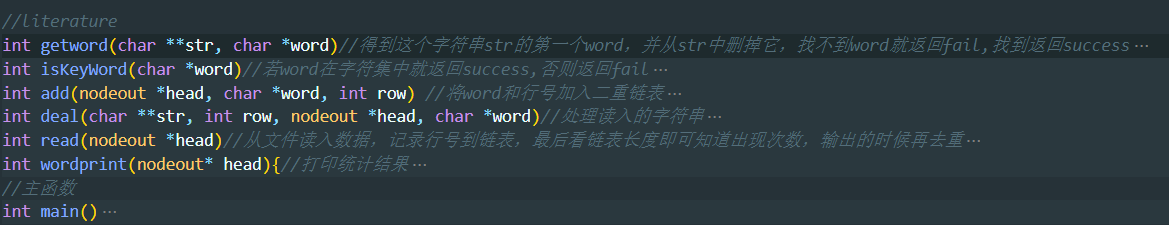
(2) 加入更复杂的编辑操作，如对某行进行串替换、在活区内进行模式匹配等，格式可以分别为

S<行号>@<串1>@<串2><回车>和m<串><回车>。

概要设计

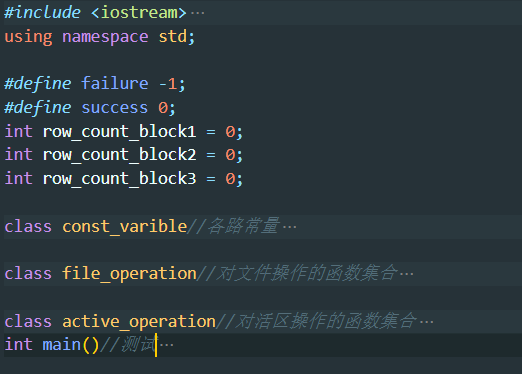
* 设计性实验

函数：



* 综合性实验

类：



详细设计

* 设计性实验

|  |
| --- |
| *#include* <stdio.h>  *#include* <stdlib.h>  *#include* <string.h>  *#define* success 0  *#define* fail -1  typedef struct infoin*//子链信息*  {      int row;  } infoin;  typedef struct nodein*//子链节点*  {      infoin row;      struct nodein \*next;  } nodein;  typedef struct infoout*//主链信息*  {      nodein \*headin;*//子链头节点*      char \*word;*//主链自身信息*  } infoout;  typedef struct nodeout*//主链节点*  {      infoout word;      struct nodeout \*next;  } nodeout;  int Initout(nodeout \*headout)*//主链初始化*  {  *if* (headout == NULL)      {          printf("malloc fail in main for headout\n");  *return* fail;      }      headout->next = NULL;  *return* success;  }  int lengthout(nodeout \*headout)*//返回主链长度*  {      int len = 0;      nodeout \*p = headout;  *while* (p != NULL)*//一直算到尾节点结束*      {          len++;          p = p->next;      }  *return* len-1;*//不计算头节点*  }  int lengthin(nodein\* headin)*//返回子链长度*  {      int len = 0;      nodein \*p = headin;  *while* (p != NULL)      {          len++;          p = p->next;      }  *return* len-1;*//不计算头节点*  }  int insertout(nodeout \*headout, infoout word,int pos)*//插入到主链第pos个位置*  {  *if* (pos > lengthout(headout)+1 || pos < 0)*//非法位置*      {          printf("illegal position in Insert\n");  *return* fail;      }      nodeout \*p = headout;  *for* (int i = pos; i > 1; i--)*//找到第pos个位置*      {          p = p->next;      }      nodeout \*x = (nodeout \*)malloc(sizeof(nodeout));*//需要插入的节点*  *if* (x == NULL)      {          printf("malloc fail in Insert");  *return* fail;      }      x->word=word;      x->next = p->next;*//插入链表*      p->next = x;  *return* success;  }  int InitIn(nodein\* headin)*//子链初始化*  {  *if* (headin == NULL)      {          printf("malloc fail in main for headin\n");  *return* fail;      }      headin->next = NULL;  *return* success;  }  int insertin(nodein\* headin,infoin row,int pos)*//插入到子链第pos个位置*  {  *if* (pos > lengthin(headin)+1 || pos < 0)*//非法位置*      {          printf("illegal position in Insert\n");  *return* fail;      }      nodein \*p = headin;  *for* (int i = pos; i > 1; i--)*//找到第pos个位置*      {          p = p->next;      }      nodein \*x = (nodein \*)malloc(sizeof(nodein));*//需要插入的节点*  *if* (x == NULL)      {          printf("malloc fail in Insert");  *return* fail;      }      x->row=row;      x->next = p->next;      p->next = x;  *return* success;  }  int printin(nodein \*head)*//打印子链*  {      nodein \*p = head;      printf("this list has nodes: %d\n", lengthin(head));      printf("the node from head to tail is:\n");  *while* (p->next != NULL)*//遍历打印*      {          p = p->next;          printf("row:%d\n",p->row.row);      }  *return* success;  }  int Inquireout(nodeout \*head, int pos,char\* word[])*//查询主链第pos个节点信息*  {  *if* (pos > lengthout(head) || pos < 0)*//非法位置*      {          printf("illegal position in Inquire\n");  *return* fail;      }      nodeout \*p = head;  *for* (int i = pos; i > 0; i--)*//找到位置*      {          p = p->next;      }      \*word=p->word.word;  *return* success;  }  int printout(nodeout\* head)*//打印主链*  {      nodeout \*p = head;      printf("this list has nodes: %d\n", lengthout(head));      printf("the node from head to tail is:\n");  *while* (p->next != NULL)*//遍历打印*      {          p = p->next;          printf("word:%s\n",p->word.word);      }  *return* success;  }  *// int main()*  *// {*  *//     nodeout \*headout = (nodeout \*)malloc(sizeof(nodeout));*  *//     Initout(headout);*  *//     nodein\* headin=(nodein\*)malloc(sizeof(nodein));*  *//     InitIn(headin);*  *//     infoin row={1};*  *//     insertin(headin,row,1);*  *//     infoin row2={2};*  *//     insertin(headin,row2,2);*  *//     printin(headin);*  *//     printf("lengthin:%d\n",lengthin(headin));*  *//     infoout word={headin,"hh"};*  *//     nodein\* headin2=(nodein\*)malloc(sizeof(nodein));*  *//     InitIn(headin2);*  *//     infoout word2={headin2,"what"};*  *//     insertout(headout,word,1);*  *//     insertout(headout,word2,2);*  *//     char\* word\_inquire;*  *//     Inquireout(headout,1,&word\_inquire);*  *//     printf("word:%s\n",word\_inquire);*  *//     printout(headout);*  *//     printf("lengthout:%d\n",lengthout(headout));*  *//     printin(headout->next->word.headin);*  *//     return 0;*  *// }* |

|  |
| --- |
| *#include* "list\_2d.c"  *//literature*  int getword(char \*\*str, char \*word)*//得到这个字符串str的第一个word，并从str中删掉它，找不到word就返回fail,找到返回success*  {  *//在函数中修改str传不回去，使用指向指针的指针str*  *//两个非字母之间的即视为一个word*      memset(word, 0, 20 \* sizeof(char));*//清空*  *if* (strlen(\*str) == 0) *//str为空时返回fail*      {  *return* fail;      }      int i = 0;*//记录str下标*      int pos = 0;*//记录word下标*  *while* (1)*//找出word*      {  *if* ((\*(\*str + i) >= 'A' && \*(\*str + i) <= 'Z') || (\*(\*str + i) >= 'a' && \*(\*str + i) <= 'z'))          {              word[pos++] = \*(\*str + i); *//若是字母就连接到word中去*              i++;          }  *else*          {  *break*; *//不是字母就选取结束*          }      }      \*str = \*str + i; *//删掉取出的word（对指针的修改需要十分注意）*  *if* (strlen(\*str) == 0)      { *//str第一次为空是刚取完时，故成功*  *return* success;      }      \*str = \*str + 1; *//再删掉不是字母的那个字符*  *return* success;*//成功返回*  }  int isKeyWord(char \*word)*//若word在字符集中就返回success,否则返回fail*  {      char esp[][31]={"int","long","short","float","double","char","unsigned","signed","const",      "void","volatile","enum","struct","union","if","else","goto","switch","case","do",      "while","for","continue","break","return","default","typedef","auto","register",      "extern","sizeof"};      int flag=0;*//标志，word为保留字就为1，否则为0*  *for*(int i=0;i<=31;i++){*//遍历比较*  *// printf("%s",esp[i]);*  *if*(strcmp(word,esp[i])==0){*//找到相同的*              flag=1;          }      }  *if* (flag==1)*//在保留字中返回success*      {  *return* success;      }  *else//不在保留字中返回fail*      {  *return* fail;      }  }  int add(nodeout \*head, char \*word, int row) *//将word和行号加入二重链表*  {*//先遍历主链看是否已有该word，有则直接加入子链，否则先在主链创建节点，再加入子链*      int flag = 0;*//标志，word已在主链中为1，否则为0*      nodeout \*p = head;*//工作指针*      int i = 0;*//查询主链的下标*  *while* (p->next!=NULL)*//遍历寻找*      {          p = p->next;          i++;          char\* word\_inquire=(char\*)malloc(sizeof(char));*//带出inquire的结果*  *if*(word\_inquire==NULL)          {              printf("malloc fail in add for word\_inquire\n");  *return* fail;          }          Inquireout(head,i,&word\_inquire);*//查询*  *if* (strcmp(word,word\_inquire)==0)*//比较*          {              flag = 1;*//找到相同，说明word在主链中*  *break*;*//退出循环*          }      }  *if* (flag == 1)*//若在主链中*      {          infoin rowin = {row};          insertin(p->word.headin, rowin, lengthin(p->word.headin) + 1);*//直接插入到对应子链*      }  *else//若不在主链中*      {          nodein \*headin = (nodein \*)malloc(sizeof(nodein));          InitIn(headin);          infoin rowin = {row};          insertin(headin,rowin,lengthin(headin)+1);*//先创建子链并插入到子链中*          char\* tmp=(char\*)malloc(20\*sizeof(char));*//不能把word直接插入，因为word是一个指针，在其他地方修改会导致此处被改变，建议const*  *if*(tmp==NULL){              printf("malloc fail in add for tmp");  *return* fail;          }          strcpy(tmp,word);*//用tmp代替word*          infoout wordout = {headin, tmp};          insertout(head, wordout, lengthout(head)+1);*//插入主链*      }  }  int deal(char \*\*str, int row, nodeout \*head, char \*word)*//处理读入的字符串*  {      char \*temp = \*str;                    *//由于\*str在getword中被修改，因此存下*  *while* (getword(str, word) == success) *//从str中取出一个word,若还能找到词*      {  *if* (isKeyWord(word) == success) *//若该词在保留字集中*          {              add(head, word, row); *//就将词和行号加入链表*          }  *else* *if* (isKeyWord(word) == fail) *//若该词不在保留字集中*          {  *continue*; *//就接着寻找*          }      }      \*str = temp;          *//处理完一行，改回\*str*  }  int read(nodeout \*head)*//从文件读入数据，记录行号到链表，最后看链表长度即可知道出现次数，输出的时候再去重*  {      char filename[]="./literature.c";*//相对地址*      FILE \*fp = fopen(filename, "r");*//打开文件*  *if* (fp == NULL)      {          printf("fail to open file\n");  *return* fail;      }  *//先读一行到str，取出单词，看是否在保留字符集中，是则加入到链中，否则继续扫描*      int row = 0;*//当前行号*  *//为了修改str可传回，使用指向指针的指针*      char \*\*str = (char \*\*)malloc(sizeof(char \*)); *//先为str申请空间*  *if* (str == NULL)      {          printf("malloc fail in read for str");  *return* fail;      }      \*str = (char \*)malloc(150 \* sizeof(char));*//再为\*str申请空间，假设一行不超过150个字节*  *if* (\*str == NULL)      {          printf("malloc fail in read for \*str\n");  *return* fail;      }      char \*word = (char \*)malloc(20 \* sizeof(char));*//从str中取出的word，假设word字长不超过20*  *if* (word == NULL)      {          printf("malloc fail in read for word\n");  *return* fail;      }  *while* (!feof(fp))*//一直到把文件读完*      {          row++;*//行数增加*          fgets(\*str, 150, fp);*//读入一行到str*          deal(str, row, head, word);*//处理数据*      }      free(word);*//释放空间*      word = NULL;      free(\*str);      \*str = NULL;      free(str);      str = NULL;      fclose(fp);  }  int wordprint(nodeout\* head){*//打印统计结果*      nodeout \*p = head;      printf("In this text, words and their rows are:\n");  *while* (p->next != NULL)*//遍历主链打印结果*      {          p = p->next;          printf("%10s appears %3d times: ",p->word.word,lengthin(p->word.headin));*//打印word*          nodein\* pin=p->word.headin;          int row\_many=0;*//记录输出的上个行数*  *while*(pin->next!=NULL){*////遍历子链打印行号*              pin=pin->next;  *if*(pin->row.row==row\_many){*//若改行号已经打印过*  *continue*;*//就查看下一个*              }*else* {*//若没打印过*                  printf("%d ",pin->row.row);*//就打印*                  row\_many=pin->row.row;*//记录这个刚打印的行号*              }          }          printf("\n");*//打完一行*      }  *return* success;  }  *//主函数*  int main()  {      nodeout \*head = (nodeout \*)malloc(sizeof(nodeout)); *//在主函数中申请头节点*      Initout(head);                                      *//初始化*      read(head);*//读入并处理数据*      wordprint(head);*//打印结果*  *return* 0;  } |

* 综合性实验

|  |
| --- |
| *#include* <iostream>  *#include* <fstream>  *#include* <string>  *#include* <math.h>  *#include* <cstring>  *#include* <string>  *#include* <iomanip>  *#include* <cstdio>  using namespace std;  *#define* failure -1;  *#define* success 0;  int row\_count\_block1 = 0;  int row\_count\_block2 = 0;  int row\_count\_block3 = 0;  class const\_varible*//各路常量*  {  public:      static const int active\_max\_len = 100;*//活区最大行块数，注意：是行块，不是行。*      static const int active\_readin\_len = 80;*//初始化时读入的行块数，留一点空间方便用户插入*      static const int line\_len\_array = 81;*//行块长度*      static const int line\_num\_array = 4;      static const int delete\_index\_flag = 0;  *// static const char line\_end\_flag = 12; //ascII码为12的字符，为非打印字符，用作行结束标志（没用到）*  };  class file\_operation*//对文件操作的函数集合*  {  private:      const string test\_file\_path = "C:\\Users\\刘润\\Desktop\\notepad\_test\_source.txt";*//测试文件路径，按需修改即可*  public:      char active\_area\_str[const\_varible::active\_max\_len][const\_varible::line\_len\_array];*//活区，存字符串（100行，81列）*      int active\_area\_index[const\_varible::active\_max\_len];*//下标对应的活区行块所在行号（size=100)*  */\**  *以上这两个数组的关系使这样的*  *比如*  *1（行） str................................*  *2（行） str................................................................*  *2（行） str.............................................*  *3（行） str................................................*  *其中第二行超过80个字符，所以占两个行块active\_area\_str，相应的，其行号也用了两个active\_area\_index*  *\*/*      int count = 0;*//活区数组尾部标志，记录活区当前尾部,注意：插入仅在活区尾部进行（由于有行号标注，所以不会乱掉）*      file\_operation()*//构造函数，初始化为数组置0*      {          memset(active\_area\_index, 0, sizeof(active\_area\_index));          memset(active\_area\_str, 0, sizeof(active\_area\_str));      }  *//打开文件*          int read\_text(int temp)*//从文件读80行块到活区*      {          count = 0;          ifstream in\_file;          in\_file.open(test\_file\_path, ios::in);  *if* (!in\_file)*//未成功打开抛出提示，返回*          {              cout << "文件打开失败" << endl;  *return* failure;          }  *for* (int i = 0; i < temp; i++) {              string s;              getline(in\_file, s);          }  *//从文件读取前80个行块（若需要打开其他行块，略作修改即可）*          string s;*//暂存读入的行的字符串*          int row = 0;*//记录行号*  *while* (getline(in\_file, s))*//只要不为空，就一直读取*          {              row\_count\_block2++;  *if* (s.length() < const\_varible::line\_len\_array)*//如果读入的行大小小于81，直接存入行块，并设置行号*              {                  str\_to\_char\_array(s, active\_area\_str[count]);*//将s存入活区数组*                  row++;*//行加一*                    active\_area\_index[count] = row;*//设置下标*                  count++;*//当前活区数组尾部标志往后移动*              }  *else//若读入的行大小超过81，则分割存入行块，并设置行号*              {                  row++;    *for* (int i = 0; i < ceil((double)s.length() / const\_varible::line\_len\_array); i++)*//分成几块*                  {  *// active\_area\_str[count]=s.substr(i\*(const\_varible::line\_len\_array-1),const\_varible::line\_len\_array-1);*                      str\_to\_char\_array(s.substr(i \* (const\_varible::line\_len\_array - 1), const\_varible::line\_len\_array - 1), active\_area\_str[count]);*//按顺序把从s分出来的块存入活区*                      active\_area\_index[count] = row;                      count++;                  }              }  *if* (count == const\_varible::active\_readin\_len)*//当前活区数组尾部标志等于80，即已经读入80个行块，退出循环*              {  *break*;                  in\_file.close();*//关闭文件*              }          }    *return* success;      }      void str\_to\_char\_array(string s, char c[])*//将字符串s存入字符数组c*      {  *for* (int i = 0; i < s.length(); i++)          {              c[i] = s[i];          }      }      int ip = 0;      void print\_active\_area\_str()*//打印当前活区数组*      {  *if* (count >= 80) {  *for* (int i = 0; i < count; i++)              {  *//cout << active\_area\_index[i] << " " << active\_area\_str[i] << endl;*                  cout << setw(4) << setfill('0') << active\_area\_index[i] << ':' << ' ' << active\_area\_str[i] << endl;              }              cout << endl;          }  *else* {  *if* (ip == 0) {                  ip = 1;  *for* (int i = 0; i < count; i++)                  {  *//cout << active\_area\_index[i] << " " << active\_area\_str[i] << endl;*                      cout << setw(4) << setfill('0') << active\_area\_index[i] << ':' << ' ' << active\_area\_str[i] << endl;                  }              }              cout << "已结束"<<endl;          }        }      void check\_count()*//检查当前活区数组尾部是否大于100，若是则说明溢出*      {  *if* (count >= 100)          {              cout << "活区溢出" << endl;              exit(-1);          }      }  };  class active\_operation*//对活区操作的函数集合*  {  public:      file\_operation f;*//为了从file\_operation弄到活区数组*      active\_operation()*//初始化*      {          total\_row();          f.read\_text(0);      }      void insert\_easy(int row, string s)*//简化接口，调用即可*      {  *//查看要插入的行是否符合条件*  *if* (row<min(f.active\_area\_index) || row == 0 || row>max(f.active\_area\_index))          {              cout << "不在活区行号范围内" << endl;  *return*;          }          insert\_row(row, s);*//插入到活区数组尾部*          clear\_active\_area();*//使紧凑*          sort();*//排序*      }      void delete\_easy(int row)*//简化接口，调用即可*      {  *if* (row<min(f.active\_area\_index) || row>max(f.active\_area\_index))          {              cout << "不在活区行号范围内" << endl;  *return*;          }          delete\_row(row);          clear\_active\_area();          sort();      }      void delete\_easy(int start\_row, int end\_row)*//简化接口，调用即可*      {  *if* (start\_row<min(f.active\_area\_index) || end\_row>max(f.active\_area\_index))          {              cout << "不在活区行号范围内" << endl;  *return*;          }          delete\_row(start\_row, end\_row);          clear\_active\_area();          sort();      }      int max(int a[])*//返回一个int[]中最大值*      {          int max = 0;  *for* (int i = 0; i < const\_varible::active\_max\_len; i++)          {  *if* (a[i] > max)              {                  max = a[i];              }          }  *return* max;      }      int min(int a[])*//返回一个int[]中最小值*      {          int min = a[0];  *for* (int i = 0; i < const\_varible::active\_max\_len; i++)          {  *if* (a[i] < min)              {                  min = a[i];              }          }  *return* min;      }      void insert\_row(int row, string s)*//s为要插入的行，插入到row行之后（仅放入活区数组尾部，设置行号）*      {  *if* (s.length() < const\_varible::line\_len\_array)*//若要插入的行长度小于81，直接放入活区数组*          {              f.str\_to\_char\_array(s, f.active\_area\_str[f.count]);*//放入活区数组的尾部*              f.active\_area\_index[f.count] = row + 1;*//设置行号*              f.count++;*//活区数组尾部标志加一*          }  *else//若大于，则分割后放入活区数组*          {  *for* (int i = 0; i < ceil((double)s.length() / const\_varible::line\_len\_array); i++)              {  *// active\_area\_str[count]=s.substr(i\*(const\_varible::line\_len\_array-1),const\_varible::line\_len\_array-1);*                  f.str\_to\_char\_array(s.substr(i \* (const\_varible::line\_len\_array - 1), const\_varible::line\_len\_array - 1), f.active\_area\_str[f.count]);                  f.active\_area\_index[f.count] = row + 1;                  f.count++;              }          }          f.check\_count();*//检查下是否溢出*  *// f.print\_active\_area\_str();//查看*      }      void delete\_row(int row)*//删除某行(仅将其行号置-1)*      {  *for* (int i = 0; i < f.count; i++)*//遍历寻找要删除的行*          {  *if* (f.active\_area\_index[i] == row)*//找到要删除的行号*              {                  f.active\_area\_index[i] = const\_varible::delete\_index\_flag;*//行号置-1*              }  *else* *if* (f.active\_area\_index[i] > row)*//要删除之后的行号需要减一*              {                  f.active\_area\_index[i] -= 1;              }          }  *// f.print\_active\_area\_str();*      }      void delete\_row(int row\_start, int row\_end)*//删除从row\_start到row\_end的行，包括两端（思路同其同名函数）*      {  *for* (int i = 0; i < f.count; i++)          {  *if* (f.active\_area\_index[i] >= row\_start && f.active\_area\_index[i] <= row\_end)*//要删除的行号*              {                  f.active\_area\_index[i] = const\_varible::delete\_index\_flag;              }  *else* *if* (f.active\_area\_index[i] > row\_end)*//要删除之后的行号*              {                  f.active\_area\_index[i] -= row\_end - row\_start + 1;              }          }  *// f.print\_active\_area\_str();*      }      void clear\_active\_area()*//在进行删除操作之后，活区中有无效区域，将有效区域填补到无效区域，使活区紧凑*      {  *for* (int i = 0; i < const\_varible::active\_max\_len; i++)          {  *//从上往下遍历，寻找空位置*  *if* (f.active\_area\_index[i] == const\_varible::delete\_index\_flag)              {  *//找到空位置，从后往前寻找，找到有效区域，填补进内*  *for* (int j = const\_varible::active\_max\_len - 1; j > 0; j--)                  {  *if* (j <= i)*//从后往前没找到时，说明已经填补完成，退出*                      {  *// f.print\_active\_area\_str();//查看填补效果*                          f.count = i;  *return*;                      }  *if* (f.active\_area\_index[j] != const\_varible::delete\_index\_flag)                      {  *//找到有效区域，将其行号拷到无效区域，再将其行块拷到无效区域，最后将已经复制完成的有效行块的行号置-1*                          f.active\_area\_index[i] = f.active\_area\_index[j];  *// f.active\_area\_str[i]=f.active\_area\_str[j];*                          strcpy(f.active\_area\_str[i], f.active\_area\_str[j]);                          f.active\_area\_index[j] = const\_varible::delete\_index\_flag;*//已搬移的置为无效*  *break*;                      }                  }              }          }      }      void sort()*//对活区排序，以行号为引，按顺序排列*      {  *for* (int i = 0; i < const\_varible::active\_max\_len; i++)*//选择排序*          {  *for* (int j = i; j < const\_varible::active\_max\_len; j++)              {  *if* (f.active\_area\_index[i] > f.active\_area\_index[j] && f.active\_area\_index[j] != const\_varible::delete\_index\_flag)*//忽略无效区域*                  {                      swap(f.active\_area\_index[i], f.active\_area\_index[j]);*//交换*                      swap(f.active\_area\_str[i], f.active\_area\_str[j]);                  }              }          }          f.print\_active\_area\_str();*//查看*      }      void swap(int& a, int& b)*//交换两个int*      {          int temp = a;          a = b;          b = temp;      }      void swap(char a[], char b[])*//交换两个char[]*      {          char temp[const\_varible::line\_len\_array];          strcpy(temp, a);          strcpy(a, b);          strcpy(b, temp);      }  *///////////////////////////////////*      void store\_active\_context() {          ofstream write\_file("C:\\Users\\刘润\\Desktop\\temp.txt", ios::out);          ifstream read\_file("C:\\Users\\刘润\\Desktop\\notepad\_test\_source.txt", ios::in);  *for* (int i = 0; i < row\_count\_block1; i++) {              string temp;              getline(read\_file, temp);              write\_file << temp;              write\_file << '\n';          }  *for* (int i = 0; i < f.count; i++) {              write\_file << f.active\_area\_str[i];              write\_file << '\n';          }          string s;  *for* (int i = 0; i < row\_count\_block2; i++) {              getline(read\_file, s);          }  *for* (int i = 0; i < row\_count\_block3; i++) {              string temp;              getline(read\_file, temp);              write\_file << temp;              write\_file << '\n';          }          write\_file.close();          read\_file.close();      }      void active\_area\_switch\_row\_count() {          row\_count\_block1 += row\_count\_block2;          row\_count\_block2 = 0;          f.read\_text(row\_count\_block1);          row\_count\_block3 = row\_count\_block3 - row\_count\_block2 - row\_count\_block1;      }        void total\_row() {          string s;          ifstream read\_file("C:\\Users\\刘润\\Desktop\\notepad\_test\_source.txt", ios::in);  *while* (getline(read\_file, s))          {              row\_count\_block3++;          }          read\_file.close();      }      void n\_option() {          store\_active\_context();          active\_area\_switch\_row\_count();          f.read\_text(row\_count\_block1);      }      void p\_option(int &start) {  *for* (int i = 0; i < 20; i++) {  *if* (start + i < row\_count\_block2) {                  cout << setw(2) << setfill('0') << i << ':' << f.active\_area\_str[i+start] << endl;              }  *else* {                  start = 0;  *break*;              }            }        }  };  int main()*//测试*  {      active\_operation a;      char option;      a.f.print\_active\_area\_str();      int start = 0;  *while* (cin >> option) {  *if* (option == 'n') {              a.n\_option();              cout << "已切换活区" << endl;          }  *if* (option == 'p') {              a.p\_option(start);              start += 20;          }  *if* (option == 'd') {              string s;              cin >> s;              int num1 = 0, num2 = 0;              int cp1 = 0;  *for* (int i = 0; i < s.length(); i++) {  *if* (s[i] < '9' && s[i]>'0'&&cp1==0) {                      num1 = num1\*10 + s[i] - '0';                  }  *if* (s[i] == '\0' || s[i] == '\t') {                      cp1 = 1;                  }  *if* (s[i] < '9' && s[i]>'0' && cp1 == 1) {                      num2 = num2\*10 + s[i] - '0';                  }              }  *if* (num2 != 0) {                  a.delete\_easy(num1, num2);              }  *else*{                  a.delete\_easy(num1);              }            }  *if* (option == 'i') {              int num;              cin >> num;              string s;              cin >> s;              a.insert\_easy(num, s);          }      }  *return* 0;  } |

4.调试分析

1. 采用IDE中自带的调试功能进行调试，手动添加断点和查看程序。
2. 对设计和编码的讨论和分析。该程序实现了顺序栈的操作。分析程序代码的质量，主要从以下几个方面考虑。

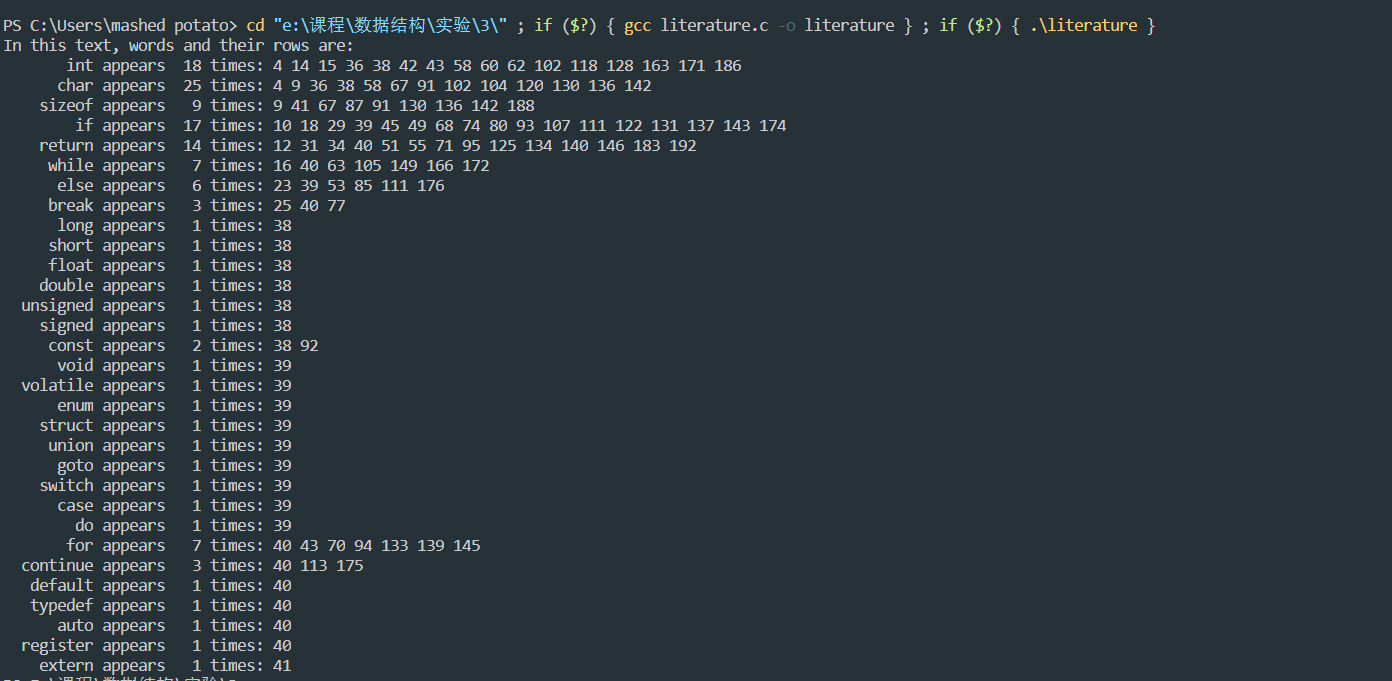
* 正确性。在一定的数据范围内，该程序能实现所需功能，所以正确性是没有问题的。
* 健壮性。在一定的数据输入范围内，该程序能较好的实现操作。但是如果输入数 据非法，该程序还是可能会产生一些预想不到的输出结构，或是不做任何处理。所以， 该程序的健壮性有待进一步的提高。要综合考虑一些情况，当输入有误时，应返回一个 表示错误的值，并中止程序的执行，以便在更高的抽象层次上进行处理。

5.使用说明

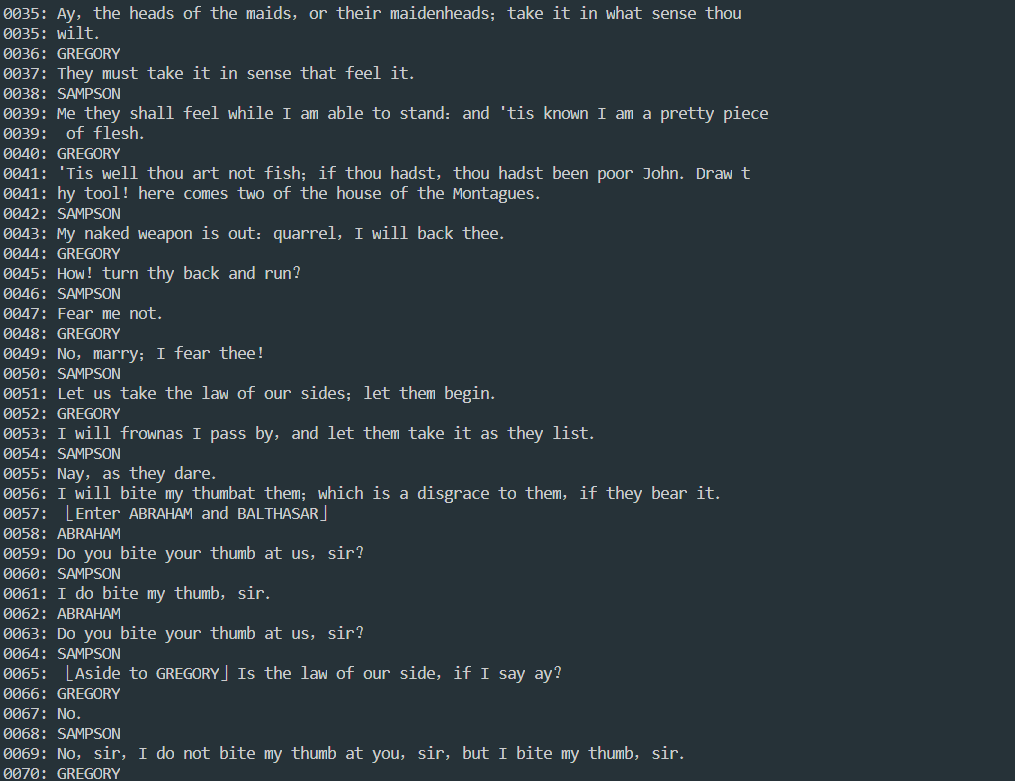
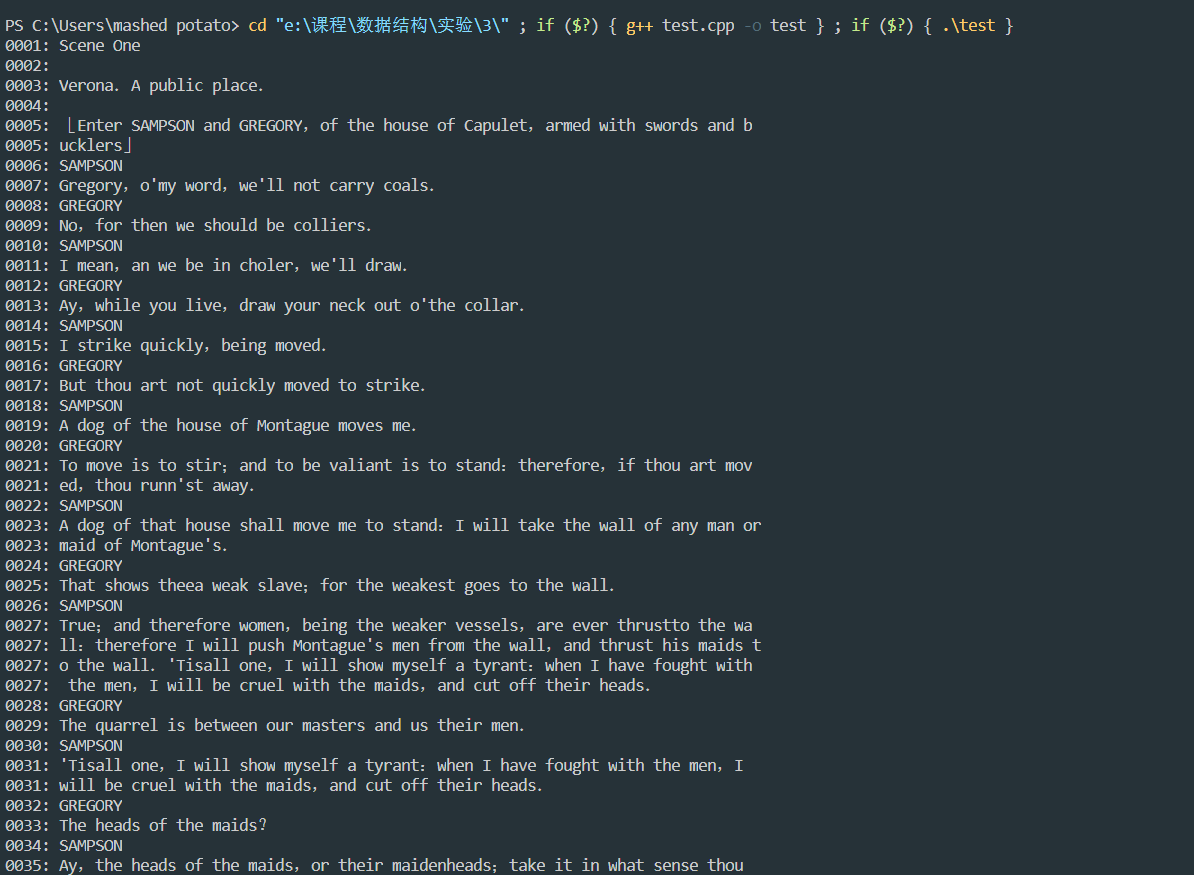
按照屏幕提示，选择想要的功能并输入对应数字，按下ENTER键后，根据屏幕提示进行输入，即可得到想要的结果。

6.测试程序运行结果

* 设计性实验



* 综合性实验



1. **心得体会**

通过本次实验，使我对数据结构有了更深的理解，对指针的运用更加熟练，熟悉了对函数的定义和操作。