****

**数据结构课程设计报告**

**题目名称： 文献检索工具**

**专 业： 信息与计算科学**

**班 级： 21级信计二班**

**学 号： 2107020203**

**姓 名： 龙威旭**

**指导教师： 胡杏**

**20 23 年至20 23 年第 二 学期**

**数学与计算科学学院**

目录

[内容摘要 3](#_Toc137633803)

[正文 3](#_Toc137633804)

[一、 课题简介 3](#_Toc137633805)

[二、 设计思路 3](#_Toc137633806)

[三、 数据结构与算法设计 3](#_Toc137633807)

[四、 程序设计 4](#_Toc137633808)

[五、 程序调试与参数测试 8](#_Toc137633809)

[六、 总结 11](#_Toc137633810)

[参考文献 12](#_Toc137633811)

内容摘要

本文介绍了一个简单的文本搜索程序的设计和实现思路。该程序通过输入一个文件名和关键字，可以打开指定的文本文件，并统计该文件中关键字的出现次数以及每一行中关键字的个数和位置。设计思路主要包括定义数据结构、使用KMP算法进行字符串匹配、初始化搜索结果、释放内存、统计关键字出现次数和位置、输出搜索结果等步骤。

正文

1. 课题简介

本课题是对文献查找“关键字”的简单模拟，通过输入文件名打开文本文件，再输入“关键字”，实现查找出所给文本文件中“关键字”的总个数，以及每一行的“关键字”个数和位置。

1. 设计思路

本课题的实质是通过给定一个文本文件和一个关键字，统计该关键字在文本中的出现次数，并输出相关信息，包括关键字出现的行号、在该行中出现的次数以及位置。可以首先定义项目的数据结构，然后将查询，输出等各个功能写成一个函数来完成操作，最后完成主函数得出运行结果。

1. 数据结构与算法设计

首先定义了两个结构体，LineInfo和SearchResult。LineInfo用于表示存储每行文本的信息，内容包括指向存储文本内容的字符数组的指针text，行号lineNumber，在该行中出现的次数occurrenceCount，关键字在该行出现的位置指针positions。SearchResult包含指向存储每行信息的LineInfo结构体数组的指针lines，文本中的行号统计lineCount，关键字在文本中的总出现次数keywordCount。

然后使用了KMP算法作为关键字的匹配算法。KMP算法通过计算关键字的next数组，在匹配过程中实现快速回溯，提高匹配效率。代码中的computeNext函数用于计算关键字的next数组，strStr函数使用KMP算法进行字符串匹配。

再然后定义了初始化搜索结果的函数initSearchResult和释放搜索结果内存的函数freeSearchResult。以及实现了统计关键字在文本中出现次数并输出相关信息的函数countOccurrences，该函数打开指定的文件，逐行读取文件内容，并在每行中查找关键字的出现次数和位置信息，返回搜索结果。最后实现输出搜索结果的函数printSearchResult，用于打印搜索结果的行号、出现次数和位置信息。

最后在主函数main中，首先获取用户输入的文件名和关键字，然后调用countOccurrences函数进行搜索，并根据返回的结果调用printSearchResult函数打印搜索结果，最后调用freeSearchResult函数释放内存。

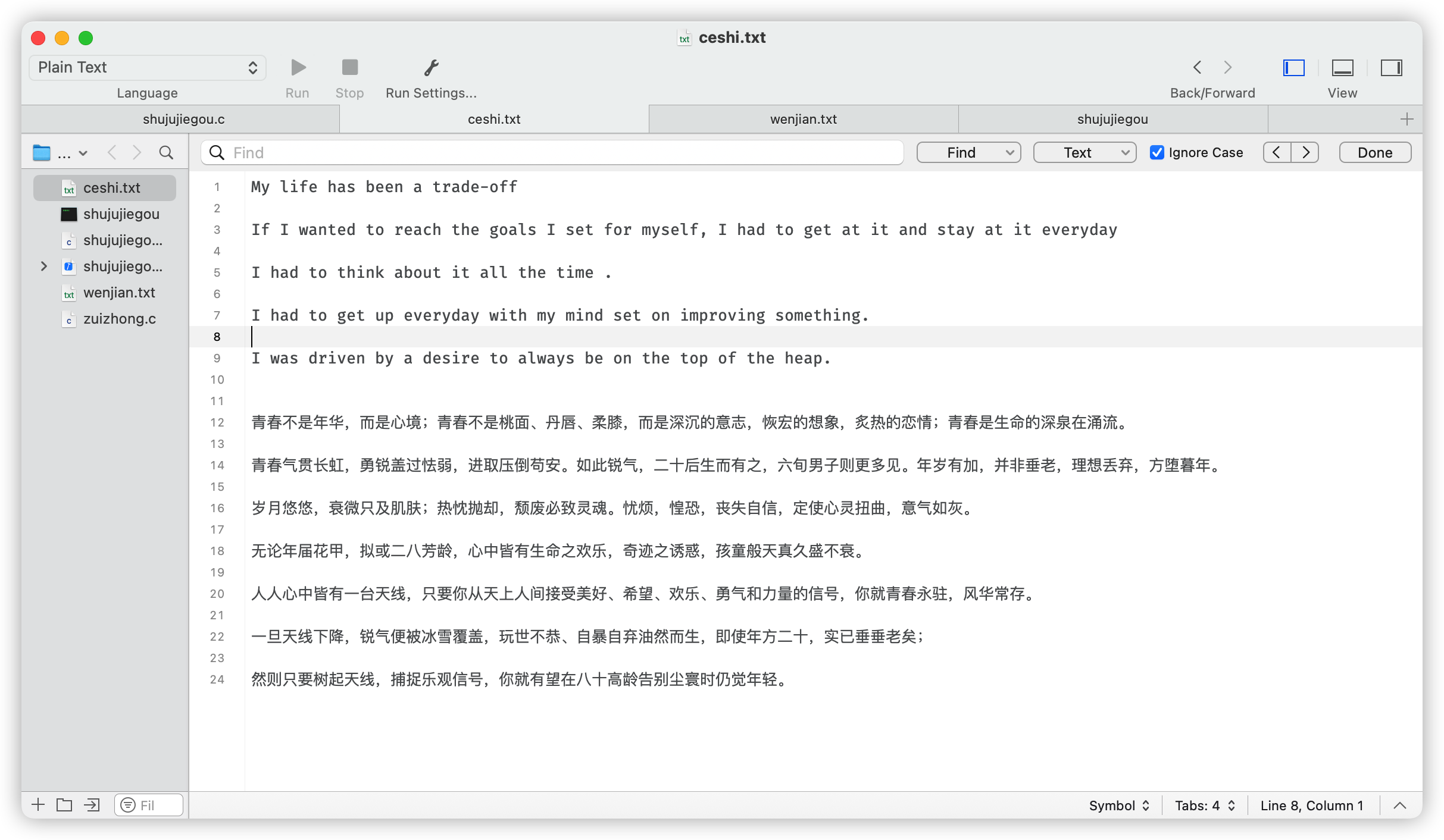
整体来说，这段代码实现了一个简单的文本搜索功能，使用了KMP算法进行字符串匹配，可以帮助用户在给定的文件中查找指定的关键字，并输出匹配结果。

1. 程序设计

|  |
| --- |
| #**include** <stdio.h>  #**include** <stdlib.h>  #**include** <string.h>  // 结构体定义  // LineInfo表示每行文本的信息  typedef struct {  char \*text; //指向存储文本内容的字符数组的指针  int lineNumber; //行号  int occurrenceCount; //在该行中出现的次数  int \*positions; //关键字在该行中出现的位置  } LineInfo;  // SearchResult表示搜索结果的信息  typedef struct {  LineInfo \*lines; //指向存储每行信息的LineInfo结构体数组的指针  int lineCount; //文本中的行号统计  int keywordCount; //关键字在文本中的总出现次数  } SearchResult;  // KMP算法中计算next数组的函数  void computeNext(char \*pattern, int \*next) {  int patternLen = strlen(pattern);//计算所匹配的字符串的长度  int i = 0, j = -1;  next[0] = -1; //将next[0]设置为-1，表示在匹配过程中的初始状态。  **while** (i < patternLen) {  **while** (j >= 0 && pattern[i] != pattern[j])  j = next[j]; //实现快速回溯  i++;  j++;  **if** (pattern[i] == pattern[j]) //判断pattern[i]和pattern[j]是否相等  next[i] = next[j];  **else**  next[i] = j;  }  //最终需要继续下一轮循环，直到i达到patternLen  }  // KMP算法的字符串匹配函数，实现KMP算法中的字符串匹配过程  char \*strStr(char \*text, char \*pattern, int \*next) {  int textLen = strlen(text);//文本字符串的长度  int patternLen = strlen(pattern);//关键字符串的长度  int i = 0, j = 0;  int matchStart = -1;    //嵌套循环  **while** (i < textLen && j < patternLen) {  **while** (j >= 0 && text[i] != pattern[j]){  j = next[j];  }  i++;  j++;  }    //在循环外部，通过判断j是否等于关键字字符串的长度patternLen来确定是否找到了完全匹配的关键字  **if** (j == patternLen)  **return** text + i - patternLen;//如果j等于patternLen，表示找到了匹配，返回匹配起始位置的指针，即text + i - patternLen  **else**  **return** **NULL**;//如果没有找到完全匹配的关键字，即j不等于patternLen，返回NULL表示未找到匹配。  }  // 初始化搜索结果，接受lineCount，表示结果的行数  SearchResult \*initSearchResult(int lineCount) {  SearchResult \*result = (SearchResult \*)malloc(sizeof(SearchResult));//分配存储搜索结果的内存空间  result->lines = (LineInfo \*)malloc(lineCount \* sizeof(LineInfo));//分配存储行信息的内存空间  //初始化行数和关键字数为0  result->lineCount = 0;  result->keywordCount = 0;  **return** result;  }  // 释放搜索结果内存  void freeSearchResult(SearchResult \*result) {  //遍历每一行，在循环内部，使用free函数释放每行中的文本字符串text和positions的内存  **for** (int i = 0; i < result->lineCount; i++) {  free(result->lines[i].text);  free(result->lines[i].positions);  }  free(result->lines);//使用free函数释放存储行信息的内存空间result->lines  free(result);//释放搜索结果结构体的内存空间result本身  }  // 统计关键字符串在文本中出现的次数并输出相关信息  SearchResult \*countOccurrences(char \*filename, char \*keyword) {  FILE \*file = fopen(filename, "r");//只读方式打开    //检查文件指针是否为NULL，如果为NULL表示文件打开失败  **if** (file == **NULL**) {  printf("文件打开错误.\n");  **return** **NULL**;  }  SearchResult \*result = initSearchResult(100); // 假设最多100行  // 计算关键字符串的next数组  int keywordLen = strlen(keyword);//关键字符串长度  int \*next = (int \*)malloc((keywordLen + 1) \* sizeof(int));  //分配的内存大小为(keywordLen + 1) \* sizeof(int)，即关键字长度加一的整数倍。  computeNext(keyword, next);//调用函数computeNext，并传递关键字keyword和数组next的指针作为参数。  // 逐行读取文件并查找关键字符串  char line[256];//存储从文件中读取的一行文本  int lineNumber = 1;//此时为第一行  //fgets函数被用于逐行读取文件中的内容，存储到字符数组line中  **while** (fgets(line, sizeof(line), file)) {  //创建一个指向搜索结果中当前行的LineInfo结构体的指针lineInfo，通过result->lineCount索引来获取  LineInfo \*lineInfo = &(result->lines[result->lineCount]);  //将当前行号lineNumber赋值给lineInfo->lineNumber，表示该行在文件中的行号  lineInfo->lineNumber = lineNumber;  //将occurrenceCount设置为0，表示当前行中关键字符串的出现次数。  lineInfo->occurrenceCount = 0;  //使用malloc函数为lineInfo->positions动态分配内存  lineInfo->positions = (int \*)malloc(100 \* sizeof(int)); // 假设最多100次出现  int lineLen = strlen(line);  int position = 0;  **for** (int i = 0; i < lineLen; i++) {  **if** (line[i] == keyword[0]) {  int match = 1;//第一个关键字匹配上了就是初始化match=1  **for** (int j = 1; j < keywordLen; j++) {  **if** (line[i + j] != keyword[j]) {  match = 0;//如果后面的每匹配上则匹配失败，返回match=0  **break**;  }  }  //match=1时成立  **if** (match) {  lineInfo->occurrenceCount++;//成功匹配的个数增加，表示当前行中关键字符串的出现次数增加  lineInfo->positions[lineInfo->occurrenceCount - 1] = position + 1;  //lineInfo->occurrenceCount - 1用于获取正确的索引位置。    //将当前成功匹配的位置存储在lineInfo->positions数组中  }  }  position++;  }  //检查当前行是否存在关键字符串的匹配  **if** (lineInfo->occurrenceCount > 0) {  lineInfo->text = (char \*)malloc((strlen(line) + 1) \* sizeof(char));  strcpy(lineInfo->text, line);  result->lineCount++;  result->keywordCount += lineInfo->occurrenceCount;  }  //如果当前行不存在关键字符串的匹配  **else** {  free(lineInfo->positions);//释放当前行的positions数组内存  }  lineNumber++;  }  fclose(file);//关闭文件  free(next);//释放用于KMP算法的next数组的内存  **return** result;  }  // 输出搜索结果  void printSearchResult(SearchResult \*result, char \*keyword) {  printf("关键字符串 '%s' 在文本中的出现情况:\n", keyword);  printf("总共出现次数: %d\n\n", result->keywordCount);  **for** (int i = 0; i < result->lineCount; i++) {  LineInfo line = result->lines[i];//使用循环遍历搜索结果中的每一行  printf("行 %d: 出现%d次, 位置为: ", line.lineNumber, line.occurrenceCount);  //对于每一行，获取行号、出现次数以及位置数组    //使用另一个循环遍历当前行的每个匹配位置  **for** (int j = 0; j < line.occurrenceCount; j++) {  printf("%d ", line.positions[j]);  }  printf("\n%s\n", line.text);  }  }  int main() {  char filename[100];  char keyword[100];  printf("输入文件名: ");  scanf("%s", filename);  printf("输入关键字: ");  scanf("%s", keyword);  //声明一个指向SearchResult类型的指针变量result  SearchResult \*result = countOccurrences(filename, keyword);  **if** (result != **NULL**) {  printSearchResult(result, keyword);//调用printSearchResult函数，打印搜索结果  freeSearchResult(result);//调用freeSearchResult函数，释放搜索结果的内存  }  **return** 0;  } |

1. 程序调试与参数测试

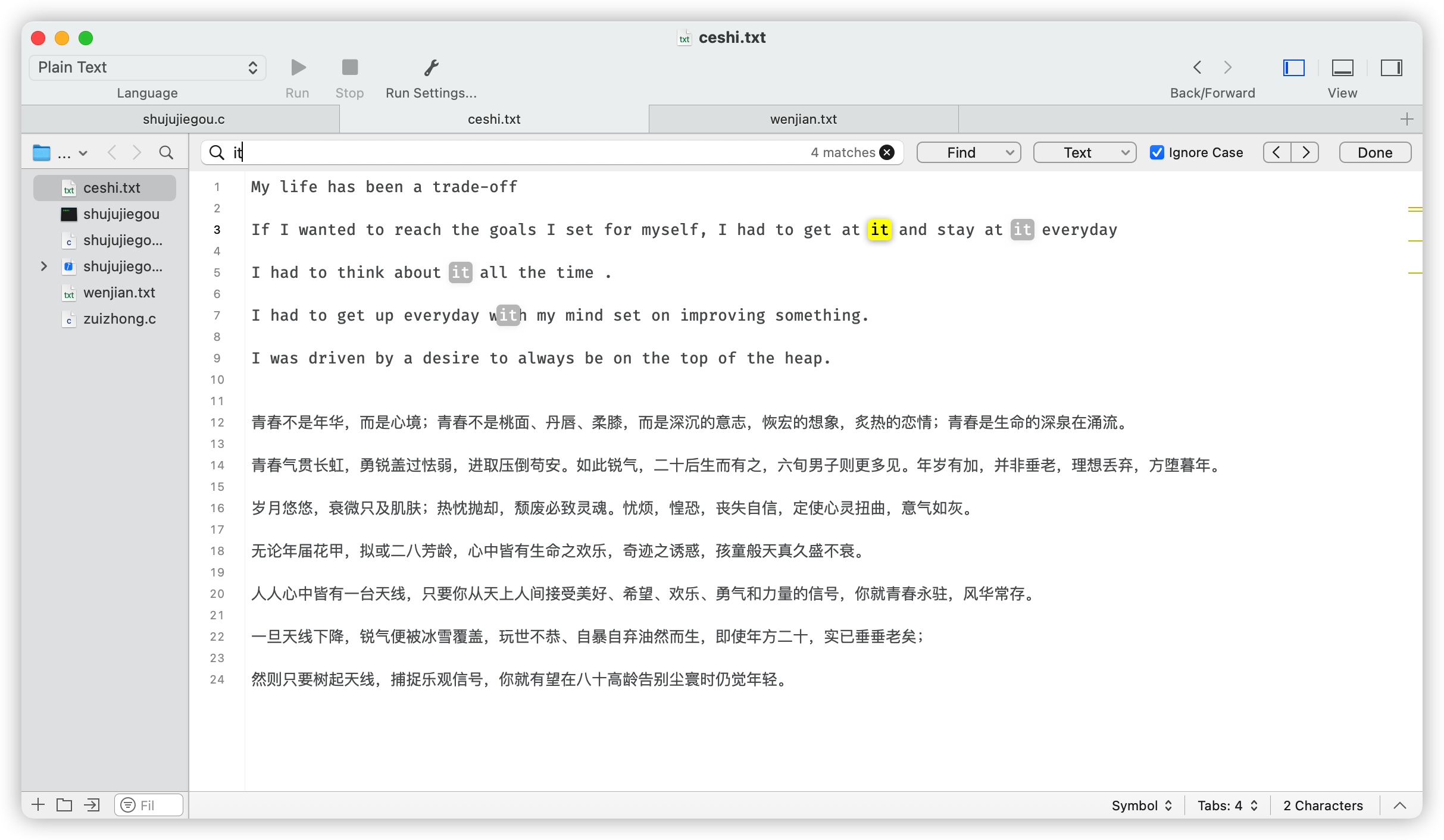
给定的ceshi.txt:



运行结果如下：



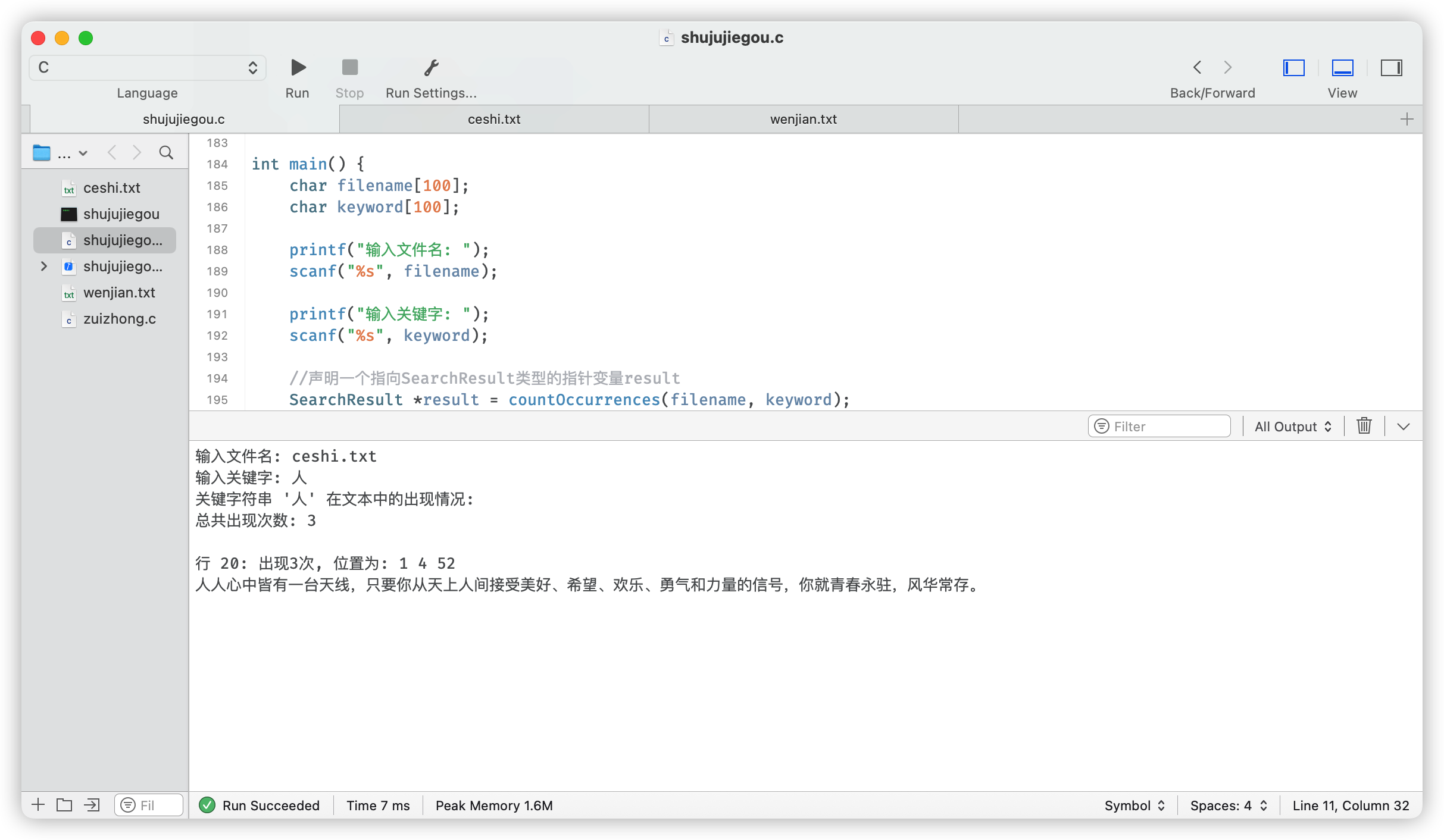
在文本中进行验证如下：



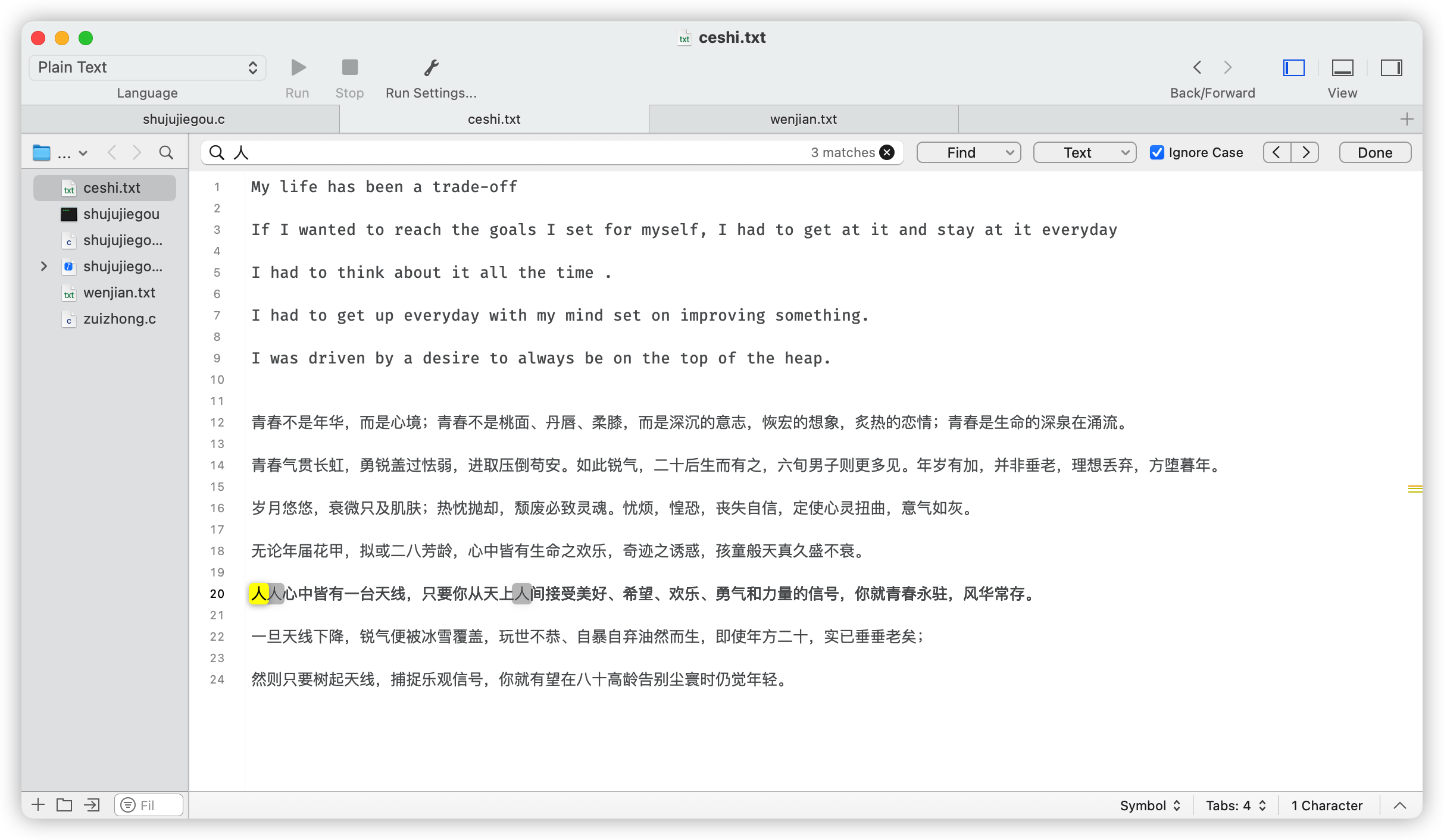
it在ceshi.txt中出现了4次：分别为第3行第66和第81位置，第5行第22位置，第7行第27位置。与运行结果完全匹配。

下面再试试中文

运行结果如下：

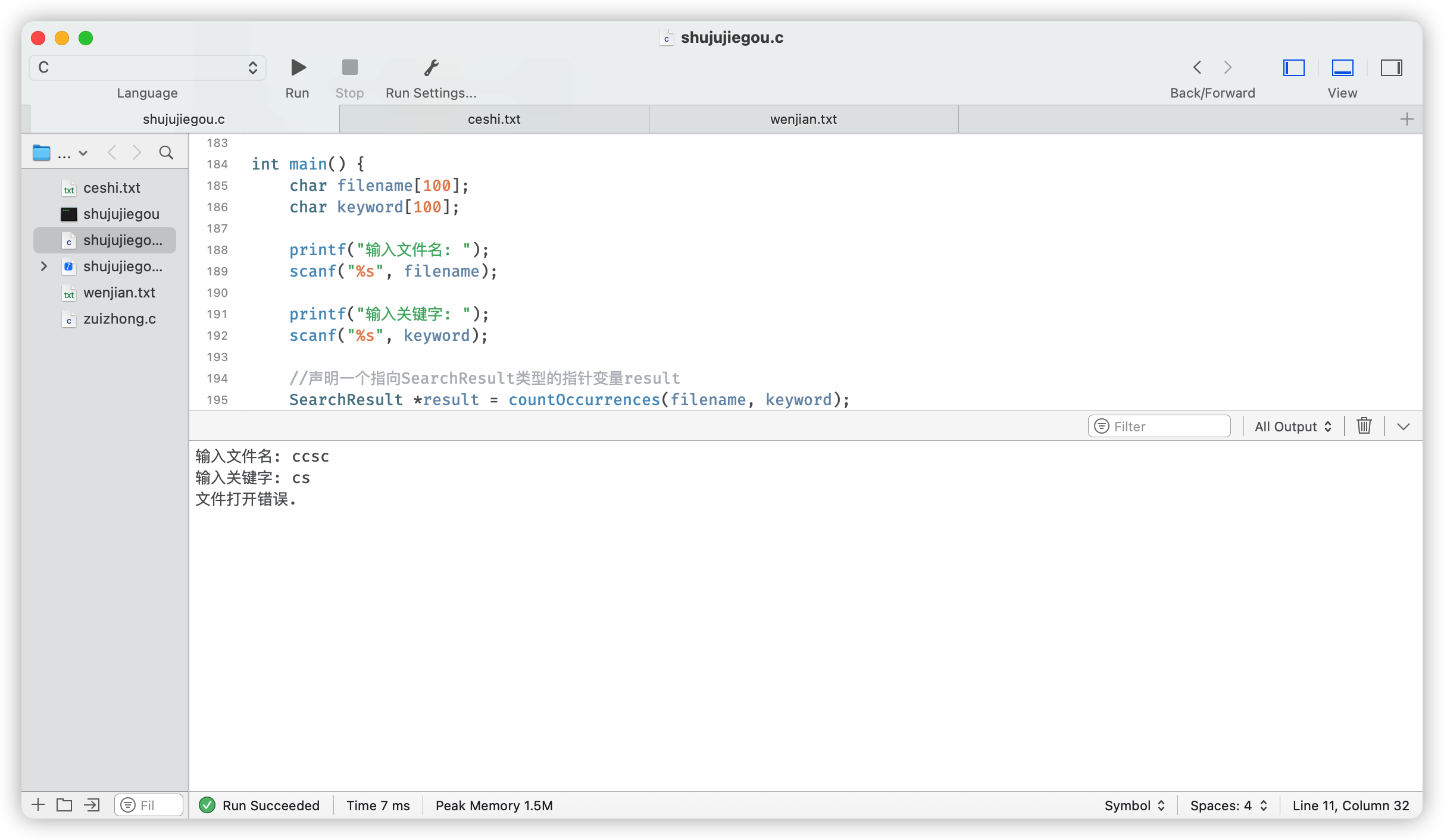


在文本中进行验证如下：



人在ceshi.txt中出现了三次，分别为第20行第1，4，52位置(一个中文占三个位置)，与运行结果完全匹配。

如若输入文件不存在，则有以下内容：



1. 总结

通过编写这个程序，我学到了如何定义结构体以及如何在程序中使用结构体变量，学到了KMP算法的基本原理和具体实现方法，包括计算next数组和在文本中查找关键字，学到了对文件的操作，内存的管理，字符串的处理，函数的调用与使用，并了解了如何实现文本搜索功能。提高了一定的程序编写能力。

参考文献

[1]孙娟红.一种基于KMP算法思想的字符串匹配算法的研究与实现[J].电脑知识与技术,2019,15(26):196-197.

[2]陈天一,郑闻悦,邹健,邱修峰.基于KMP算法的字符串查找匹配研究[J].科技创新导报,2019,16(23):242-243.

[3]姚秀情.浅析KMP算法中next数组值计算[J].数字技术与应用,2019,37(03):131-132.

[4]王晓波.基于KMP算法Next数组的分析与优化[J].电子世界,2017,(20):196+198.

[5]田华,鄢喜爱.信息检索中模式匹配算法的分析[J].高校图书馆工作,2007,(06):41-43.