#**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;

}