```
======================================================================
            /local/submit/submit/comp10002/ass1/xuliny/src/ass1sol3.c
======================================================================

/*comp10002 assignment1 by Xulin Yang 904904, September 2017*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <math.h>

#define LOG2(x) log(x)/log(2.0) /*calculate log(x) with base 2*/

#define MAX_CHARACTER 1001 /*maximum char in one line*/
#define MAX_LINE 5 /*maximum line can be stored*/
#define ONE_CHAR 1 /*number of single character*/
#define NO_INPUT 1 /*no input query*/

#define VALID 1 /*True*/
#define INVALID 0 /*False*/

#define NO_QUERY "No query specified, must provide at least one word"
        /*error information for no query input*/
#define INVALID_CHARACTER ": invalid character(s) in query"
        /*error information for invalid character input*/

#define END "---" /*seperate line*/
#define STAGE_ONE "S1: " /*stage 1 output indication*/
#define STAGE_TWO "S2: " /*stage 2 output indication*/
#define STAGE_THREE "S3: " /*stage 3 output indication*/
#define STAGE_FOUR "S4: " /*stage 4 output indication*/

#define DECIMAL_ZERO 0.0 /*initial score in double type*/
#define DECIMAL_ONE 1.0 /*convert an int to double by multiplying 1.0*/
#define DENOMINATOR_CONSTANT 8.5
        /*constant in denominator when calculating score*/

typedef struct proc_line proc_line_t;
typedef struct stored_line stored_line_t;

struct stored_line {
    char sentence[MAX_CHARACTER]; /*the line in file*/
    double score; /*query score for each line*/
    int line; /*line number*/
};

struct proc_line {
    int bytes; /*number of characters in one line*/
    int words; /*number of words in one line*/
    stored_line_t details; /*characters, score, line number*/
};

void check_argc(int argc);
void check_query(int argc, char *argv[]);
int check_query_string(char *query);
void print_query(int argc, char *argv[]);
void initialize_stored_lines(stored_line_t stored_line[]);

int mygetchar();
void get_sentence(stored_line_t stored_line[], char *query[], int query_num);
void stage_output(stored_line_t stored_line[], proc_line_t proc_line);

double cal_line_score(char *query[], int q_num, int w_num, char* sentence);
int query_appearance(char *query, char *sentence);
int similar_char(char *query, char *sentence);
void print_proc_line(proc_line_t proc_line);

void ranking(stored_line_t stored_lines[], proc_line_t proc_line);
void print_rank(stored_line_t stored_lines[]);

int main(int argc, char *argv[]) {
    /*S1*/
    check_argc(argc);
    print_query(argc, argv);

    check_query(argc, argv);

    /*S2*//*S3*/
    stored_line_t stored_line[MAX_LINE];
    initialize_stored_lines(stored_line);

    get_sentence(stored_line, argv, argc);

    /*S4*/
    print_rank(stored_line);

    return 0;
}

/*S1*/

/*check correct number of input query*/
void check_argc(int argc) {
    if (argc == NO_INPUT) {
        printf("%s%s", STAGE_ONE, NO_QUERY);
        exit(EXIT_FAILURE);
    }
    return;
}

/*check which input query in invalid*/
void check_query(int argc, char *argv[]) {
    int i, validation = VALID;

    for (i = 1; i < argc; i++) {
        if (!check_query_string(argv[i])) {
            printf("\n%s%s%s", STAGE_ONE, argv[i], INVALID_CHARACTER);
            validation = INVALID;
        }
    }

    if (!validation) {
        exit(EXIT_FAILURE);
    }

    return;
}

/*a query in invalid when it is not a lower case alphabet or a digit*/
int check_query_string(char *query) {
    int i, validation = VALID;

    for (i = 0; i < strlen(query); i++) {
        if (!isdigit(query[i]) && (!islower(query[i]))) {
            validation = INVALID;
        }
    }

    return validation;
}

/*print out all invalid query*/
void print_query(int argc, char *argv[]) {
    int i;

    printf("%squery =", STAGE_ONE);
    for (i = 1; i < argc; i++) {
        printf(" %s", argv[i]);
    }

    return;
}

/*initialize score and line number for each stored line*/
void initialize_stored_lines(stored_line_t stored_line[]) {
    int i;
    for (i = 0; i < MAX_LINE; i++) {
        stored_line[i].score = DECIMAL_ZERO;
        stored_line[i].line = 0;
    }

    return;
}

/*S2*/

/*function strip '\r' in file from LMS*/
int mygetchar() {
    int c;
    while ((c=getchar())=='\r') {
    }
    return c;
}

/*print non-empty processing line and its details,
  calculate score for each line,
  find top5 query matched line*/
void get_sentence(stored_line_t stored_line[], char *query[], int query_num) {
    int line_num = 0, line_len = 0, in_word = 0;
    char c;
    proc_line_t proc_line;
    proc_line.words = 0;

    while((c = mygetchar(stdin)) && (line_len < MAX_CHARACTER)) {
        /*if it is an alphabet or digit then a word starts*/
        if (isalnum(c)) {
            in_word = 1;
        }
        /*if turns from alphabet or digit to other character, then a word
        has ended*/
        else if (in_word) {
            proc_line.words++;
            in_word = 0;
        }

        /*end of one line*/
        if ((c == '\n') || (c == EOF)) {
            /*assign processing value*/
            proc_line.details.sentence[line_len] = '\0';
            proc_line.bytes = line_len--;
            proc_line.details.line = ++line_num;
            proc_line.details.score = cal_line_score(query, query_num,
                proc_line.words, proc_line.details.sentence);

            stage_output(stored_line, proc_line);

            /*initialize for next line*/
            line_len = 0;
            proc_line.words = 0;

            /*end of file*/
            if (c == EOF) {
                break;
            }

            continue;
        }

        /*store character in line*/
        proc_line.details.sentence[line_len++] = c;
    }

    return;
}

/*output processing line*/
void stage_output(stored_line_t stored_line[], proc_line_t proc_line) {
    if (proc_line.bytes > 0) {
        /*print non-empty line's stage2 and stage3*/
```

```c
        print_proc_line(proc_line);

        /*store processing line when its score > 0.0*/
        if (proc_line.details.score > DECIMAL_ZERO) {
            ranking(stored_line, proc_line);
        }
    }

    return;
}

/*S3*/

/*calculate and return score for each line*/
double cal_line_score(char *query[], int q_num, int w_num, char* sentence) {
    int i;
    double score = DECIMAL_ZERO;

    for (i = 1; i < q_num; i++) {
        score += LOG2((DECIMAL_ONE + DECIMAL_ONE *
            query_appearance(query[i], sentence)));
    }
    score /= LOG2(DENOMINATOR_CONSTANT + DECIMAL_ONE * w_num);

    return score;
}

/*return number of times that the query is a case-insensitive prefix match
  against the word that appears in that input line*/
int query_appearance(char *query, char *sentence) {
    int sent_len = strlen(sentence), i, in_word = 0, similar = 0;
    int q_len = strlen(query), q_appear = 0;

    for (i = 0; i < sent_len; i++) {


        /*first alnum after non-alnum character is the first char of word*/
        if (isalnum(sentence[i]) && !in_word) {
            /*found word*/
            in_word = 1;

            /*skip known similar prefix character in a word*/
            similar = similar_char(query, &sentence[i]);
            if (similar > ONE_CHAR) {
                i += similar;
            }

            /*add 1 to query appearence in line when have same word prefix*/
            q_appear += (similar == q_len);

        } else if(!isalnum(sentence[i]) && in_word) {
            /*start searching new word*/
            in_word = 0;
            similar = 0;
        }
    }

    return q_appear;
}

/*return similar prefix character of query and word*/
int similar_char(char *query, char *sentence) {
    int similar = 0;

    /*stop compare when the word or the query stops*/
    while((*sentence) && isalnum(*sentence) && (*query)) {
        /*compare character one by one case-insensitive and
          stops when found one different*/
        if (!strncasecmp(sentence, query, ONE_CHAR)) {
            similar++;
        } else {
            break;
        }
```

```c
            query++;
            sentence++;
        }

        return similar;
    }

    /*print stage two and three when line is not empty*/
    void print_proc_line(proc_line_t proc_line) {
        printf("\n%s", END);
        printf("\n%s", proc_line.details.sentence);
        printf("\n%sline = %d, bytes = %d, words = %d",
            STAGE_TWO,
            proc_line.details.line,
            proc_line.bytes,
            proc_line.words);
        printf("\n%sline = %d, score = %.3lf",
            STAGE_THREE,
            proc_line.details.line,
            proc_line.details.score);

        return;
    }

    /*S4*/

    /*find appropriate ranking for processing line to be stored*/
    void ranking(stored_line_t stored_lines[], proc_line_t proc_line) {
        int i;
        stored_line_t tmp;

        for (i = 0; i < MAX_LINE; i++) {
            /*if processing line's score > stored line's score or
            they have same score but proc_line's line number is before
            stored line's line number insert it*/
            if ((proc_line.details.score > stored_lines[i].score) ||
                ((proc_line.details.score == stored_lines[i].score) &&
                (proc_line.details.line < stored_lines[i].line))) {

                tmp = stored_lines[i];
                stored_lines[i] = proc_line.details;
                proc_line.details = tmp;
            }
        }

        return;
    }

    /*print out stage four*/
    void print_rank(stored_line_t stored_lines[]) {
        int i;

        for (i = 0; i < MAX_LINE; i++) {
            /*only print out non-zero-line-number record which is valid*/
            if (stored_lines[i].line > 0) {

                if (i == 0) {
                    printf("\n---------------------------------------------");
                }

                printf("\n%sline = %d, score = %.3lf", STAGE_FOUR,
                    stored_lines[i].line, stored_lines[i].score);
                printf("\n%s\n%s", stored_lines[i].sentence, END);
            }
        }

        return;
    }

    /*algorithms are fun*/
```