(DASF004) Basis and Practice in Programming
Homework Assignment #2

Instructor: Hyungjoon Koo (kevin.koo@g.skku.edu)

Your Name : _____

Student ID : _____

The homework assignment must be submitted on an individual basis. The following states several rules for both submission and grading policy. Please read the descriptions of each exercise carefully before you make a final submission.

- You must submit a single zipped file via `iCampus`. The submission file name should be your student ID with the file extension of `.zip`.
- Unlike the first assignment, we use the `Make` utility for practice. However, `Makefile` will be given, so you do not need to modify the file at all.
- We will consider the last submission only in case of multiple submissions.
- You write a program per each question where a naming rule is the question number with the file extension of `.c`. For example, the program name of the first question would be `1.c`.
- You have (up to) two days for a late submission with a 25% delay penalty per day.
- If your program fails to be compiled, no point would be given. This is a strict rule.
- You may obtain partial points based on test sets, which will be judged by the TAs.
- You must follow code of conduct. Keep in mind that any activity of copying, publishing, sharing your code with others is *not allowed*. Violating code ethics may lead to failing this course.

Exercise 1.

(25 Points) In this exercise, you write a program that checks if a password follows the following rules, which must be robust against a password guessing attack (Listing 1).

- A password is case-sensitive.
- The length of a password (P) must range from 10 to 30. (i.e., `10 <= len(P) <= 30`). Your program should be terminated otherwise.
- A password must contain a number, a lower case letter, a upper case letter, and a special character.
- The special character represents one of the following nine letters: !, %, _, #, &, +, -, *, /
- Your program will exit when a user provides "`quit`".

```c
// Skeleton code for q1.c
#include <stdio.h>
#include <string.h>

#define PASSWD_SZ 31
#define TRUE 1
#define FALSE 0

char quit[] = "quit";
char special_letters[] = "!%_&#+-*/";

// Write a function for checking a password
int check_passwd(char* password, int passwd_len);

int main(void)
{
        int res;
        char password[PASSWD_SZ];

        while (1) {
                printf("Create your password (up to 30 letters): ");
                scanf("%s", password);

                // Your code here

                if (res == TRUE)
                        printf("Good password!\n");
                else
                        printf("Bad password!\n");
        }

        return(0);
}

// Sample outputs
Create your password (up to 30 letters): 5vskjf // Short length
Bad password!
Create your password (up to 30 letters): 6Ynbrksk!DFK // All good
Good password!
Create your password (up to 30 letters): #$dnck875555 // Upper letter missing
Bad password!
Create your password (up to 30 letters): quit // Terminating a program
```

Listing 1: Skeleton code and its output examples for Exercise 1.

Exercise 2.

(35 Points) In this exercise, your write a program that solves a maze game (Listing 2).

- Your program should be able to read a maze in `maze.txt`, which consists of 'x' and 'o'.
- 'x' represents a wall that one cannot pass through, whereas 'o' represents a path that one can pass through.
- The start coordinate of the maze is (1,1), and the end coordinate is (5,5).
- One goes to the right or the bottom at all times (not the other way around).
- If one fails to find a path from (1,1) to (5,5), the program will be terminated with `"No exit!"`.
- (Hint) Implementation in a recursive way would be helpful.

```c
// Skeleton code for q2.c
#include <stdio.h>
#define TRUE 1
#define FALSE 0
#define VISIT 'v'   // Mark 'v' if you visit a path
#define SZ 7

/*
SAMPLE MAZE in maze.txt
x x x x x x x
x o o o o o x
x o x o x o x
x o o x o x x
x x o x o x x
x o o o o o x
x x x x x x x
*/

char maze[SZ][SZ];
int si, sj, ei, ej;
int success;

int visit(int i, int j);
void read_maze();

int main(void) {
  success = FALSE;
  read_maze();

  // Start coordinate
  si = 1;
  sj = 1;

  // End coordinate
  ei = SZ-2;
  ej = SZ-2;

  if (visit(si, sj) == FALSE)
    printf("No exit!\n");
  else
    printf("\n");

  return 0;
}

// Sample output (either should be fine)
(5, 5) (5, 4) (5, 3) (5, 2) (4, 2) (3, 2) (3, 1) (2, 1) (1, 1) // or
(1, 1) (2, 1) (3, 1) (3, 2) (4, 2) (5, 2) (5, 3) (5, 4) (5, 5)
```

Listing 2: Skeleton code and its output examples for Exercise 2.

Exercise 3.

(40 Points) Reading a given text file, (e.g., `words.txt`) in Listing 3, write a program that answers the following questions for `q3.c`.

(1) How many words are there in the file? (5 points)

(2) What is the longest word of all in length? (5 points)

(3) A word may include special characters such as "." or "-". What is the ratio of the words that consist of pure alphabets (i.e., $[a-zA-Z]$)? Use the `double` type when declaring a variable. The ratio should end with % rounded to two decimal places (e.g., $33.1865\% \rightarrow 33.19\%$). (10 points)

(4) A word may have pairs of consecutive double letters. For example, the word "tooth" has one pair of double letters where the word "committee" has three pairs of consecutive double letters. Your task is to count the words that contain such consecutive double letters. Note that you should count each word as one even when the word has multiple pairs. (10 points)

(5) By taking a single string, count the number of sub-words that includes that string from the whole words. (10 points)

```c
// Skeleton code for q3.c
#include <stdio.h>
#include <string.h>

char* file_path = "words.txt";

int main(void) {
        char subword[20];

        printf("Enter a subword to find: ");
        scanf("%s", subword);

        // You may write your own function(s) if needed;
        int ans_31 = 0;
        int ans_32 = 0;
        double ans_33 = 0.0;
        int ans_34 = 0;
        int ans_35 = 0;

        // Your code here

        printf("%d %d %.2lf%% %d %d\n", ans_31, ans_32, ans_33, ans_34, ans_35);

        return 0;
}
```

Listing 3: Function prototypes for Exercise 3.

- During evaluation, we may feed several files to check the correctness of your answers.
- You should read each line only once.
- Each line may have multiple words. A word can be separated by only a space. For example, "`I am a student!`" has four words: "`I, am, a, students!`". In case of "`He is a three-year-old boy.`", it has five words: "`He, is, a, three-year-old, boy.`". It is noted that a special character is part of a word.
- A sub-word is a sequence of letters that can be part of a word. For example, Both "`university`" and "`version`" include a sub-word "`ver`" within.

- As a line feed (LF) for a new line may differ depending on operating systems, you should assume a Linux environment, that is, `\n`.
- You may use any string relevant function (including the one you learned in class) that has been defined in the `string.h` header.
- Read the file only when a file descriptor (pointer) is successfully generated, that is, a `file_ptr` is not NULL.
- You may assume that the number of subwords to be entered is less than 20 letters.