Computer Programming Final

2020.01.08

- You cannot access Internet when exam. After you complete the exam, please raise your hand and TA will plug the cable for you and submit your codes to moodle.
- Makefile is undoubtedly necessary. If there is no Makefile or Makefile does not work, **you will get nothing**.
- You file should be [id]_mid.zip .
- The code name should be fin01.c, ..., fin05.c respectively.
- The binary executives should be fin01 ..., fin05 respectively.
- **Input check is necessary**.

1 Equivalent Resistance (25 pts)

The electrical resistance of an object is a measure of its opposition to the flow of electric current. The resistance (R) of an object is defined as the ratio of voltage across it (V) to current through it (I).

$$R = \frac{V}{I}.$$

In a given combination of resistors (series, parallel, or combination of series/ parallel), the equivalent resistance is that value of resistance, which when replaced in place of the combination, will continue to give the same performance for the part of circuit outside this combination. I believe that you know how to calculate the equivalent resistance. If no, do not worry. I will teach you how to calculate the equivalent resistance in series and parallel resistor combinations. Please see the figure 1.

Now, given a circuit as figure 2. Please write a program to calculate the equivalent resistance. You need to let a user to input R and n, which are 32-bits integers. All resistors have the same resistance value.

```
$ ./fin01
Please enter the resistance (1-100): 1
Please enter n (1-100): 1
Ans: 2
```

Again, precision is not our concern.

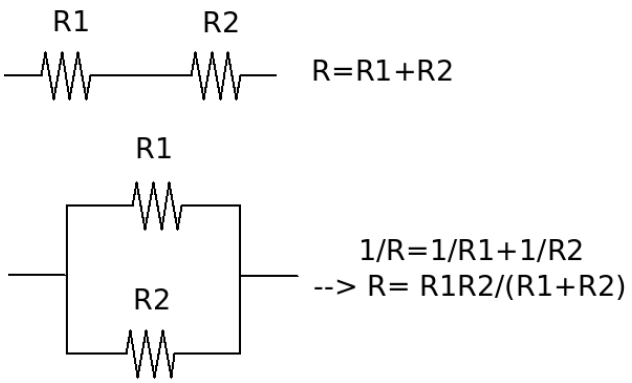

Figure 1: Equivalent Resistance.

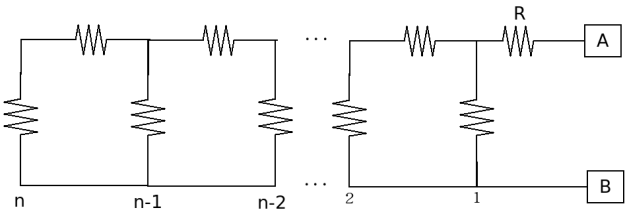

Figure 2: Equivalent Resistance Problem.

2 Maze (25 pts)

Given you a maze, please develop a program to find a way from Entrance to Exit. A maze id presented in one two-dimensional array as follows.

```
int32_t array[m][n];
```

For your simplicity, we set $(0, 0)$ as the entrance point and $(m-1, n-1)$ as the exit point. If array[x][y] $= 0$, it means (x, y) is walkable; otherwise, (x, y) is not walkable. Of course, you cannot break the boundary like $(-1, -1)$. Once you find a way, you must use an animation to present how to walk to the exit. The step interval should be 200 msec. The walkman should be presented as *, the walkable point should be presented as a space and the unwalkable point should be presented as a +. You also need to draw all boundaries as unwalkable points. I will give you an example. Given a maze as follows:

```
int32_t array[2][2] ={ { 0, 0 },
                       { 1, 0 } };
```

The output of your program should be

```
$ ./fin02
++++
+* +
++ +
++++
```

Then

```
++++
+ *+
```

```
++ +
++++
```

and

```
++++
+  +
++*+
++++
Exit
```

I will give you a maze in **maze.h**, including **m and n**. Please include it in your code. Of course, TA will evaluate your code with a different maze.h.

3 Move Zeroes(25 pts)

Given an array nums, write a function to move all 0's to the end of it while maintaining the relative order of the non-zero elements. The array and the array size will be put in **zeroes.h**. Please include it in your code. Of course, TA will evaluate your code with a different zeroes.h.

```
int32_t array[5] = {0,1,0,3,12};
```

The output of your program should be

```
$ ./fin03
1 3 12 0 0
```

Note that memory allocation is not allowed, including copying the array into a new array.

4 Bulls and Cows (25 pts)

This is a very popular game. The game is usually played with 4 digits. The players each write a 4-digit secret number. The digits must be all different. Then, in turn, the players try to guess their opponent's number who gives the number of matches. If the matching digits are in their right positions, they are "bulls", if in different positions, they are "cows". Example:

- Secret number: 4271
- Player's try: 1234
- Answer: 1 bull and 2 cows. (The bull is "2", the cows are "4" and "1".)

This time, you need to implement both Examiner and Player. For the examiner, you need to implement the following functions:

```
int32_t setup();
int32_t check( int32_t guess[4], int32_t *bull, int32_t *cow );
```

setup is used to pick a 4-digit secret number. You must use **random** in this function. Of course, you need to make sure that all digits must be different. **check** is used to derive bull and cow. If the guess matches the secret, return 1. If the guess includes duplicated digits or any other unreasonable values, return -1. Otherwise, return 0. For the player, you need to implement the following functions:

```
int32_t gen_guess( int32_t guess[4], int32_t bull, int32_t cow );
```

gen_guess is used to generate a 4-digit secret number where **bull** and **cow** are derived from the last round check.

I will give you a fin04.c which contains a main function. Do not modify fin04.c. However, you need to write Makefile on your own. Your program must gets the correct answer no more than 100 times.

5 Bonus (10pts)

Do you have any suggestions about this class? Just write down your opinions in fin05.txt.