Assignment 2 CMPT 125

INDIVIDUAL or PAIR ASSIGNMENT:
COLLABORATION PERMITED ONLY WITH YOUR OWN PARTNER

SUBMISSION

This assignment consists of two problems. For each problem you will submit one text file containing the commented C code that solves that problem.
For the first problem you will submit a file mastermind.c.
For the second problem you will submit a file with the name robots.c.

MARKING SCHEME

80% of the marks for each problem will be based on the results produced by running your code for a series of tests.

- Some tests will be provided with the problems in this assignment and as text files
- Some tests will be added for grading and will be provided when your graded assignment is returned.
- Any functionality, constraint or other specification of the program requested in the problem description in this document may be tested when the code is graded.
- Some functionalities, constraints, or other specifications of any program described in this document may not be tested by the provided tests.

20% of the marks will be for direct examination of the code. This will include points for

- Following the class coding standard. (see file posted with this assignment)
- Implementation of specific portions of the solution.

EXPECTATIONS FOR EACH PROBLEM

1. The description of each problem (in this document) will include a series of tests.
   a) Each test will specify, in red, the values input to your code.
   b) For this assignment only the input verification specified or is necessary to satisfy other requirements of the problems is necessary.
   c) Each test will also specify all the expected output values and the formatting of the prompts, inputs and outputs that your code will be expected to produce.
2. For each test included in each problem, at least one file containing the expected results of running your program will be posted. These files will define the expected details of wording, spacing and other formatting for each test and the expected values of each output.
3. Your output for a test and the provided output, for the same test, must match character by character to receive full marks for a test. Please watch the short video tutorial “Checking your outputs” for the details of how to compare your output to the corresponding provided output file.

You will use the environment specified below to build your solutions to assignments. Why we need a common environment and how to access the environment four this course is explained in the “Remote Access tutorial”.
Ubuntu 22.04, gcc 11.4.0, gdb 12.1, vscode 1.82.2 (to be updated before 1st class)

To help you understand and generate random numbers needed for the two problems in this assignment please begin by watching the video tutorial "Pseudo-Random Numbers" posted in the assignment 2 module. This tutorial will serve two purposes. First, to explain the pseudo random numbers you will generate in your program. Second, to provide an example of how to use the pseudo-random number generator. The example discussed is the development of the function used to generate the random stripes pattern used in problem 2.

PROBLEM 1 (35 POINTS): MASTERMIND Game Variant in C

You will implement a variant on the game MASTERMIND as a C program. The player of this game tries to guess a sequence of six digits from 0 to 5. The player has a maximum of 20 turns in a game.

Your program will create the solution, an ordered list of six random digits from 0 to 5 inclusive. The player tries to guess solution, needing to get both the correct digits and their order to win.

Your program will meet the following specifications:

1. Use the pseudo-random number generator functions, rand() and srand(), to generate the solution for the game.
2. The player will provide the value of the unsigned integer, ***initSeed***, used to initialize the random number generator (Prompted for, then read into the program.)
3. Your program will place the six digits of the solution into a 1-D solution array, in the same order it generates them.
4. Each guess made by the player may include any character and may include any number of characters followed by a \n. The \n is created by pressing the enter key.
   4.1. Your program will ignore any space or tab characters in the guess.
   4.2. The guess is valid if it contains only spaces, tabs, \n, and the digits 0, 1, 2, 3, 4, and 5.
   4.3. Any guess found to be invalid will cause your program to print an error message. Then your program should request a new guess to replace the invalid one.
   4.4. Any valid guess that does not contain enough valid digits to fill the guess array will cause your program to prompt the player to enter the remaining digits.
   4.5. After your program has read 6 digits into the guess array, it will ignore the rest of the guess.
5. To determine the number of partial and exact matches that are in a guess, keep track of which elements in the answer and which elements in the guess have already been paired.
   5.1. A pair forms when a digit in the answer is matched with the same digit in the guess.
      5.1.1. A digit in the answer array can be paired with ONE element in the guess array
      5.1.2. A digit in the guess array can be paired with ONE element in the answer array
      5.1.3. An element may also remain unpaired.
6. An exact match occurs when a digit in element N in the solution array is paired with a digit in element N in the guess array. A partial match occurs when a digit in element N of the guess array is paired with element M of the answer array, and $M \neq N$.
7. After reading a valid guess into the guess array, your program will determine the number of exact matches then the number of partial matches.
8. The numbers of exact and partial matches will be printed immediately after they are counted.
9. Display the history of guesses and the system's responses before the user makes a new guess.
10. Exit the program when the player correctly guesses the sequence of digits or exhausts her maximum of 20 turns.

Game 1

```
Enter the integer value of the seed for the game: 234
For each turn enter 6 digits 0 <= digit <= 5
Spaces or tabs in your response will be ignomagenta.

Enter your guess, 6 digits
555555
4 matches 0 partial matches


Enter your guess, 6 digits
Previous guess 1: 5 5 5 5 5 5 - 4 matches 0 partial matches
4444
You need to enter 2 more digits to complete your guess
4
You need to enter 1 more digits to complete your guess
4
1 matches 0 partial matches


Enter your guess, 6 digits
Previous guess 1: 5 5 5 5 5 5 - 4 matches 0 partial matches
Previous guess 2: 4 4 4 4 4 4 - 1 matches 0 partial matches
 3 3 3   3   3   3
1 matches 0 partial matches


Enter your guess, 6 digits
Previous guess 1: 5 5 5 5 5 5 - 4 matches 0 partial matches
Previous guess 2: 4 4 4 4 4 4 - 1 matches 0 partial matches
Previous guess 3: 3 3 3 3 3 3 - 1 matches 0 partial matches
123456
ERROR: A character in your guess was a digit that was too large
Reenter your guess, 6 digits
00000h
ERROR: A character in your guess was not a digit or a white space
Enter your guess, 6 digits
553455
6 matches 0 partial matches
YOU DID IT!!
```

This file is posted along with this document. An output file for a second game is also posted. It is too long to include here.

PROBLEM 2 (65 points):

You will write a C program (a simulation) that records the path taken by each robot, in a group of robots. The robots are wandering in a rectangular room. The floor of the rectangular room consists of square coloured tiles. When a robot moves it paints each tile it steps on with the colour of paint it carries.

There are eight parameters that will determine the patterns made by the moving robots. Your program will

- Request and read the name of an input file containing the values of these parameter. Check if the file was opened. Re-prompt and reread the filename if necessary. Do not try to re-read more than 5 times.
- Read the values of these parameters from the specified input file.
- For each parameter, your program will verify the data, before reading the next parameter

The details of these steps are outlined in the section, DATA INPUT AND VERIFICATION, below.

Next, your program will use the values of five of the parameters to initialize the simulation. There are three steps the initializing the simulation. First, your program must allocate memory to hold the colour of each tile on the floor. Second, your program must assign initial colour to each tile on the floor. Third, for each robot, your code will choose a tile to place the robot on, a direction that the robot is facing, and the colour of the paint the robot is carrying. The section below, called INITIALIZATION, discusses the details of each of these three steps.

Each time one robot moves it will move four tiles in one direction, painting each tile it steps on before moving to the next tile. When it reaches the fourth tile the robot will turn to face a particular direction based on the colour of the square it is standing on. Then, the robot will paint that tile. During each iteration Robot N will finish moving and painting, before, robot N+1 begins to move. The robots will move in the same order during each iteration. Iteration 0 is the initialization step.

DATA INPUT AND VERIFICATION

Your program will read each of the quantities in the following list from the input file. For quantities in items 1) to 7) Your program must use defined constants for the limits of the ranges given below, A list of Error messages are given at the end of this section.

1. The number of rows of tiles in the room, 12 <= ***numRows*** <= 100.
2. The number of columns of tiles in the room, 12 <= ***numCols*** <= 100.
3. The number of robots, 1 <= ***numRobots*** <= 10.
4. The index indicating the pattern for initialization for the floor tiles, 1 <= ***initTypeValue*** <= 3.
5. The unsigned int, ***initSeed***, for the choosing of random colours and locations in the room, 10 <= ***initSeed*** <= 32767.
6. The number of iterations to calculate, 5 <= ***numIterations*** <= 2000. Note that calculating N iterations means that you have iteration 0, which is the initialization, plus N calculated iterations.
7. The output will be printed to the screen every ***interval*** iterations, 1 <= ***interval*** <= ***numIterations.*** Your program will print the output for iteration 0 (initial state), iteration 1, iteration ***interval***, iteration 2x***interval*** ... and finally iteration (***numIterations***)
8. The name of the output file that will contain the results.

List of error messages and prompts

Your program will check if:

- The input value has been read, that is, you had not reached the end of file before this read.
- The input value has been read, that is, have any characters been read by the scanf.
- The value that was read is in the specified range.

If any of these tests fail you will print the correct error message, chosen from those in the list below, to stderr. After printing the error message, you will clean up, then you will terminate the program. Each error message should end with a newline.

Your program MUST use defined constants for the bounds of the ranges for each variable. When you print the error messages that include the range your program MUST use the defined constants to print the error messages given below. In other words, if ONLY the value of your defined constant is changed, the value of the corresponding bound in your error message must also change.

```
ERROR: The number of rows was not in the input file (reached eof)
ERROR: The number of rows could not be read due to corrupt data in the file
ERROR: The number of rows was outside the specified range (12 to 100 inclusive)
ERROR: The number of columns was not in the input file (reached eof)
ERROR: The number of columns could not be read due to corrupt data in the file
ERROR: The number of rows was outside the specified range (12 to 100 inclusive)
ERROR: The number of robots was not in the input file (reached eof)
ERROR: The number of robots could not be read due to corrupt data in the file
ERROR: The number of robots was outside the specified range (1 to 10 inclusive)
ERROR: The initialization type was not in the input file (reached eof)
ERROR: The initialization type could not be read due to corrupt data in the file
ERROR: The initialization type was outside the specified range (1 to 3 inclusive)
ERROR: The initialization seed was not in the input file (reached eof)
ERROR: The initialization seed could not be read due to corrupt data in the file
ERROR: The initialization seed was outside the specified range (10 to 32767 inclusive)
ERROR: The number of iterations was not in the input file (reached eof)
ERROR: The number of iterations could not be read due to corrupt data in the file
ERROR: The number of iterations was outside the specified range (10 to 5000 inclusive)
ERROR: The print interval was not in the input file (reached eof)
ERROR: The print interval could not be read due to corrupt data in the file
ERROR: The print interval was outside the specified range (1 to 1000 inclusive)
```

INITIALIZATION

Your program must use a 2-D integer array with dimension M rows by N columns, $12 < M < 100$, $12 < N < 100$, to represent the room in which the robots are wandering. Each element in the array represents one tile on the floor of the room containing the robots. Each element in the array will hold the colour of the tile it represents.

- 1 represents yellow.
- 2 represents cyan.
- 3 represents green.
- 4 represents blue .
- 5 represents magenta.
- 6 represents white.

A structure of type, robot, will hold the information describing one robot.

```
struct Robot{
int x;
int y;
int direction;
int paint Colour;
};
```

b) The first step of initialization is to dynamically allocate the necessary memory for
   - 1-D array of Robots, with length ***numRobots.***
   - 2-D array of integers that holds the colours of the tiles on the floor of the room containing the robots. Dimensions of the array are ***numRows*** by ***numCols.***

   Check that each memory allocation has worked and print one of the following error messages if an allocation fails. After printing an error message, clean up, and terminate the program.

   ERROR: Array of pointers for 2-D array could not be allocated
   ERROR: Array storage for the 2-D array could not be allocated
   ERROR: Array of robots could not be allocated

c) To implement the second step, you must know that the floor the 2-D array represents will have one of three tiled patterns immediately after the second step of initialization.
   Your program must use the following global enumeration to manage these three patterns.

   enum initTypeList{ RANDOM_STRIPES = 1, CHECKERBOARD, ALL_MAGENTA};

   The patterns and function prototypes are:
   - **All magenta:** All tiles on the floor must have colour magenta.

   void InitFloorAllMagenta(int **floor, int numRows, int numCols);

   - **Magenta and white checkerboard pattern:** The upper left square of the checkerboard is white. Most squares on the checkerboard have a size of four tiles by four tiles. If ***numRows*** is not a multiple of four the rightmost square in a row may have one, two or three columns. If ***numCols*** is not a multiple of four the bottommost square in a column may have one, two or three rows.

   void InitFloorChecker(int **floor, int numRows, int numCols);

   - **Random coloured vertical stripes: (**Please use the function provided)

   This function randomly chooses a colour for each tile in the first row of tiles. This function must choose from the six colours listed. To produce vertical stripes, every other row will be a copy of the first row.

   void InitFloorRandStripe(int **floor, int numRows, int numCols, unsigned int seed);

d) To implement the third step an algorithm is provided below. This algorithm determines the x then y coordinates of the tile the robot will be placed on, the direction the robot is facing, and the colour of paint the robot is carrying.
   IMPORTANT: If you generate the random numbers in a different order from that given in the algorithm your program will give different results from the reference program.

```
Initialize the random number generator.
   For each robot  (robot(1), robot(2), ... , robot(number of robots) {
           Generate random x coordinate for robot;
           Generate random y coordinate  for robot;
           Generate random direction to be the direction the robot is facing;
           Generate the colour of paint the robot is carrying  (chosen from colours 1,2,3, 4)
```

<u>EACH ITERATION</u>

1) When the robot moves forward, it moves in the direction it is facing at the start of the iteration (the end of the previous iteration).
   - If moving forward moves the robot off the floor's northernmost tile, it will reappear in the southernmost tile in the same column.
   - If moving forward moves the robot off the floor's southernmost tile, it will reappear on the northernmost tile in the same column.
   - If moving forward moves the robot off the floor's easternmost tile, it will reappear on the westernmost tile in the same row.
   - If moving forward moves the robot off the floor's westernmost tile, it will reappear on the easternmost tile in the same row.

2) One robot will complete its actions before the next robot begins its actions.

3) Each robot will complete the following actions each iteration:
   - Step forward onto the next tile and paint that tile. Repeat two more times.
   - Step forward onto the next tile a fourth time.
   - Based on the colour of the tile the robot is standing on the robot will rotate
     - 90 degrees clockwise when standing on a yellow tile.
     - 180 degrees clockwise when standing on a cyan tile.
     - 270 degrees clockwise when standing on a green tile.
     - 0 degrees (360 degrees) when standing on a blue tile.
     - 90 degrees clockwise when standing on a magenta tile.
     - 180 degrees clockwise when standing on a white tile.
   - Paint the tile the robot is standing on. (the colour the robot is carrying)

4) Your program will print the floor to an output file for every iteration, including iteration 0.
   Your program will print the floor to the screen for iteration 0 (initial state), iteration 1, iteration interval, iteration 2xinterval ..., and finally iteration ***numIterations***.

5) You will be provided with a print function that will display the floor in colour on your screen. This print function prints in colour only when printing to the screen. You will need to write your own print function to print to the output file.

You may use any additional structures and enumerations you wish. You can only change the parts of the structure definition outside of the {}.