Assignment 1 CMPT 125

This assignment consists of two problems.
For the second problem you will submit a file with the name robots.c, for the first problem you will submit a file Mastermind.c.
80% of each problems mark will be based on running tests (see next paragraph for details) and the remaining 20% will be for specific specifications of the code in the problem that are not testable by running the code. For example, did you correctly allocate and deallocate dynamic memory rather than using automatic arrays, or did you properly clean up before each place the code is terminated.
The functionality of each problem will be graded by comparing the output produced by your program to the reference solutions.

- The given error messages should be used and should match character by character to those in , and the outputs should be matched character by character by your output to receive all points.

Parts of the outputs of the tests will be chosen for grading. Points will be allocated only for those chosen parts. Each part of the output chosen will represent a requirement stated in the description of the problem. Some of the tests provided to you in the test plan will be modified by changing input values and used for grading. Not all functionalities of the program are tested in the provided test plan. Some additional functionalities not tested in the provided test plan will be graded using additional tests.

The codes will be graded using the following environment available on the SFU VDI and the LINUX compute servers used when you remotely access the SFU LINUX compute servers as explained in the remote access tutorial. The environment on the workstation in the CSIL on the CSIL workstations in not the same as this environment.
Ubuntu 22.04, gcc 11.4.0, gdb 12.1, vscode 1.82.2

PROBLEM 1 (50 POINTS): MASTERMIND Game Variant in C

You will enhance a variant on the game MASTERMIND using C. This variant will use the digits 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. The computer (your program) will create an ordered group of six of these digits. Each of the six digits will be randomly selected, with each digit potentially appearing any number of times in the group (from 0 to 6). Your program will use at least three arrays, each with six locations to hold six digits, which will hold integers. You will also need to use additional arrays to **keep track of and display the history of guesses and the system's feedback** for each guess.

The player tries to guess the group of digits, needing to get both the correct digits and their order to win. To aid the player in discovering the correct group of digits, you will provide feedback after each guess, indicating how many digits are exactly right (correct digit in the correct place) and how many are partially correct (correct digit but in the wrong place). Additionally, after each guess, display all previous guesses along with their feedback to help the player make informed decisions. **You may assume that the player may take a maximum of 100 turns in a game.**

Gameplay and Feedback Example: If the sequence chosen by the computer is **1 2 4 3 2 2** and the player's guesses and the system's responses were as follows:

- First guess: **2 3 4 5 6 7**, Response: 1 digit correct (the 4) and 2 digits partially correct (the 2 and the 3).
- Second guess: **2 3 2 0 1 9**, Response: 0 digits correct and 4 digits partially correct (2, 3, 2, 1).

Before the player makes their third guess, the game will display:

- Guess 1: **2 3 4 5 6 7** - 1 correct, 2 partially correct.
- Guess 2: **2 3 2 0 1 9** - 0 correct, 4 partially correct.
- Enter your next guess, 6 digits:

Your game should:

1. Read the seed number at the beginning of the game.
2. Use the pseudo-random number generator (functions rand() and srand()) to create a random sequence of 6 digits between 0 and 9 inclusive. Place these digits, in the order they are generated, into your answer array.
3. Read the characters the user entered as their guess.

    3.1. The guess will be one line of characters terminated by moving to a new line.
    3.2. The guess may contain any characters.
    3.3. Any space or tab characters in the guess will be ignored.
    3.4. After reading six digits from the guess, the remaining characters in the guess will be ignored.
    3.5. If any characters in the guess, that are not digits, spaces, or tabs, are encountered before 6 digits have been read, the guess will be considered invalid.
    3.6. Any guess that does not contain enough digits to complete the guess, and does not contain any characters other than digits, spaces, or tabs, will cause the user to be prompted to enter the remaining digits.
    3.7. The values in each guess will be stored in an array of length six.

3.8. Any guess found to be invalid will cause an error message to be printed, followed by a request to enter a new guess. Before requesting the new guess, any remaining characters in the stream should be cleared.
3.9. **NOTE**: you can read one character using scanf with %c, or by using getc().

4. Keep track of which elements in the answer and which elements in the guess have been paired.

    4.1. A pair forms when an element in the answer array can be matched with an element in the guess array if the two elements hold the same value.
    4.2. An element in the answer array can be paired with only one element in the guess array and vice versa.
    4.3. An element may also remain unpaired.

5. Determine the number of exact matches (correct digit in the correct place) and partial matches (correct digit but in the wrong place) for the current guess. This feedback will be provided to the user after each guess.
6. Display the history of guesses and the system's responses before the user makes a new guess.
7. Exit the program when the user correctly guesses the sequence of digits.

Debugging Tip: To make it easier to debug your program, you may wish to use the same value for the seed so that you have the same solution each time you run the program.
Ensure all output matches the formats and messages illustrated below in the test plan. Spacing and formatting are crucial for automated marking. Copies of the files will be posted on Canvas to ensure clarity on spacing, as many browsers change the spacing shown in text.

Game 1

```
Enter the integer value of the seed for the game: 144
For each turn enter 6 digits 0 <= digit <= 9
Spaces or tabs in your response will be ignored.

Enter your guess, 6 digits
  3 57
You need to enter 3 more digits to complete your guess
6 9 4 9
Guess 1: 3 5 7 6 9 4 - 0 correct, 2 partially correct

Enter your guess, 6 digits
zzz234
ERROR: A character in your guess was not a digit or a white space

Enter your guess, 6 digits
1 2 8                   7 9 4
Guess 1: 3 5 7 6 9 4 - 0 correct, 2 partially correct
Guess 2: 1 2 8 7 9 4 - 2 correct, 0 partially correct

Enter your guess, 6 digits
5 5 t 8 9 5
ERROR: A character in your guess was not a digit or a white space

Enter your guess, 6 digits
  3 5
You need to enter 4 more digits to complete your guess
1       2
You need to enter 2 more digits to complete your guess
7 2
Guess 1: 3 5 7 6 9 4 - 0 correct, 2 partially correct
Guess 2: 1 2 8 7 9 4 - 2 correct, 0 partially correct
Guess 3: 3 5 1 2 7 2 - 1 correct, 2 partially correct

Enter your guess, 6 digits
0 2 2 7 7 6
YOU DID IT!!
Guess 1: 3 5 7 6 9 4 - 0 correct, 2 partially correct
Guess 2: 1 2 8 7 9 4 - 2 correct, 0 partially correct
Guess 3: 3 5 1 2 7 2 - 1 correct, 2 partially correct
Guess 4: 0 2 2 7 7 6 - 6 correct, 0 partially correct
```

Game 2

```
Enter the integer value of the seed for the game: a2435wt23
Try again you made an error
Please enter an integer to be the seed for the game: 333333
For each turn enter 6 digits 0 <= digit <= 9
Spaces or tabs in your response will be ignored.

Enter your guess, 6 digits
345     345     345
Guess 1: 3 4 5 3 4 5 - 1 correct, 2 partially correct

Enter your guess, 6 digits
4 4 4   5 7 9 8 0 42    a
Guess 1: 3 4 5 3 4 5 - 1 correct, 2 partially correct
Guess 2: 4 4 4 5 7 9 - 3 correct, 2 partially correct

Enter your guess, 6 digits
744489asdf;lkjag
Guess 1: 3 4 5 3 4 5 - 1 correct, 2 partially correct
Guess 2: 4 4 4 5 7 9 - 3 correct, 2 partially correct
Guess 3: 7 4 4 4 8 9 - 5 correct, 0 partially correct

Enter your guess, 6 digits
        7       4       4       469     asdl
Guess 1: 3 4 5 3 4 5 - 1 correct, 2 partially correct
Guess 2: 4 4 4 5 7 9 - 3 correct, 2 partially correct
Guess 3: 7 4 4 4 8 9 - 5 correct, 0 partially correct
Guess 4: 7 4 4 4 6 9 - 4 correct, 0 partially correct

Enter your guess, 6 digits
 5 4 4 4 8 9
YOU DID IT!!
Guess 1: 3 4 5 3 4 5 - 1 correct, 2 partially correct
Guess 2: 4 4 4 5 7 9 - 3 correct, 2 partially correct
Guess 3: 7 4 4 4 8 9 - 5 correct, 0 partially correct
Guess 4: 7 4 4 4 6 9 - 4 correct, 0 partially correct
Guess 5: 5 4 4 4 8 9 - 6 correct, 0 partially correct
```

PROBLEM 2 (50 points):

You will consider a problem with the following constraints.

- You have a dynamic array of size M rows by N columns. 20 < M < 300, 20 < N < 300
  - The array represents an enclosure containing one to ten robots.
  - Each element in the array represents one large tile on the floor of the enclosure containing the robot or robots.
  - Each element in the array will hold the colour of the tile it represents. There are four possible colours for tiles, white, red, green and blue.
  - Before the robots take their first move the colours of the tiles are initialized to one of the four colours. Your program will implement a selection of three initialization algorithms to initialize the colours of tiles. The initialization functions will be discussed below.

- Consider each turn in the process. In one turn
  - Each robot carries a can of paint. The colour of this can of paint is randomly chosen from white, red, green, and blue at the time the robot is placed on the board.
  - all the robots move from one tile to an adjacent tile (north, south, east, or west) by going forward.
  - When a robot moves onto a tile it will rotate to face one of the 4 surrounding squares (to the north, south, east or west). The amount the robot rotates will be determined by the colour of the tile the robot is standing on. Rotation will be clockwise. White 90 degrees, Red 180 degrees, green 270 degrees, and blue 360 degrees (or 0 degrees).
  - The robot will then paint the tile it is standing on with the colour of paint it is carrying.

Here are some rules that constrain your program:

- Do not use global variables.
- Use named constants, not literal values.
- You may use Global constants for constants that are needed in multiple functions.
- You should follow the class coding standard.
- Use the following structures and enumerations. This is a minimum set, you may use more if you wish. You can only change the parts of the structure definition outside of the {}.

```
enum initTypeList{ Random = 1, Checkerboard, AllWhite};

struct Robot{
   int x;
   int y;
   int direction;
   int paintColour;
};
```

1) Begin your program by reading the name of the input file containing the parameters for the game (into a character array). Try a maximum of five times. Use the following prompts and error messages.
   Enter the name of the input file:
   ERROR: Input file not opened correctly.
   ERROR: Failed to open the input file 5 times. Terminating the program.
2) Read the following quantities from the input file. These values must be read in the order they are mentioned in the following list.
   - The number of rows in the board you wish to use (20 to 300 inclusive)
   - The number of columns in the board you wish to use (20 to 300 inclusive)
   - The number of robots to include (1 to 10 inclusive)
   - The index indicating the type of initialization for the board (Random=1, Checkerboard=2 or AllWhite=3). USE A GLOBAL ENUM FOR THE TYPE OF INITIALIZATION.
   - Read the seed initSeed for the choosing of random colours (unsigned int between 10 and 32767 inclusive)
   - The number of turns (10 to 5000 inclusive)
   - The output will be saved every interval turns. The interval must be in the range one to number of turns inclusive. The output will be printed for turn 0 (initial state), turn interval, turn 2xinterval …. turn number of turns -1
   - The name of the output file in which the results will be saved.

   For each integer value, you should check if the value was read, if the value was corrupted, and if the value is in range. If the value supplied fails any of these tests you should print an error message to stderr then clean up and terminate the code. **All error messages generated should be printed to stderr. Error messages in step 3 and later (after opening the output file) should also be printed to your output file.** Use the appropriate messages from the following list for each error message.
   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 (20 to 300 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 columns was outside the specified range (20 to 300 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 initTypeValue was not in the input file (reached eof)
   ERROR: The initTypeValue could not be read due to corrupt data in the file
   ERROR: The initTypeValue was outside the specified range (1 to 3 inclusive)
   ERROR: The initSeed was not in the input file (reached eof)
   ERROR: The initSeed could not be read due to corrupt data in the file
   ERROR: The initSeed was outside the specified range (10 to 32767 inclusive)
   Follow the same pattern for the error messages for the number of turns and the printing interval.

3) Dynamically allocate one 2-D integer array with identifier boardpp and with dimensions number of rows by number of columns. Check that the allocations have worked and print the following error messages if the allocation fails. After printing an error message, clean up, and terminate the program. This array represents the gameboard. This array will hold the integer representations of the colours of the squares (White=1, Red=2, Green=3, Blue=4)
ERROR: array of pointers for 2-D array could not be allocated
ERROR: Array storage could not be allocated
Dynamically allocate one 1-D array of structures of length numRobots.

4) You will then initialize the board by calling function **InitBoard** with the following prototype. This function will call the selected individual initialization functions.

```
int InitBoard( int **boardpp, int numRows, int numCols,
               enum initTypeList mylist, unsigned int seed);
```

The initialization function will be chosen to create one of the three possible initial board types described below

Initial Board type Rand: The colour of each square randomly chosen from the four available colours. For each square, the probability of any one of the colours being chosen will be equal. This function is posted with the description of the assignment. The provided function initializes using srand(initSeed) immediately before determining the colour of the first square (0,0). The colour of the squares are initialized by one, using one call to rand for each square. The first square (0,0) will use the first random number generated by rand(). Initialization will proceed along a row, then on to the start of the next row. This is the same ordering as the ordering of elements in memory. To assure a matching random board please use the provided function for this initialization method.

```
void InitBoardRand(int **board, int numRows, int numCols,
               unsigned int seed);
```

Initial Board type Checkerboard: White and red checkerboard pattern with upper left square (0,0) initialized to white.

```
void InitBoardChecker(int **board, int numRows, int numCols);
```

Initial Board type AllWhite: All squares in the board initialized to white.

```
void InitBoardAllWhite(int **board, int numRows, int numCols);
```

5) Randomly place each robot on the board, assign the direction it is facing, and assign the colour of paint it is carrying. To do these things use the following algorithm. To compare to the provided output (get the same random numbers) order is very important.

Initialize the random number generator srand(seed)
For each robot (robot(1), robot(2), … , robot(number of robots -1)
    Robot x coordinate = rand() % numCols;
    Robot y coordinate = rand() % numRows;
    Direction robot is facing= rand() % NUM_DIRECTIONS + 1;
    Colour of paint the robot is carrying = rand() % NUM_COLOURS + 1;

6) Run the game for the number of turns requested. For each turn each robot will move forward, then rotate, and then change the colour of the square it has moved onto.
    a. **If moving forward moves the robot off the top of the board in will reappear in the bottom row (the column will not change)**
    b. **If moving forward moves the robot off the bottom of the board it will reappear on the top row of the board (the column will not change)**
    c. **If moving to the left moves the robot off the side of the board then it will reappear on the other side of the board (the row will not change)**
7) You will print the board into the output file and to the screen every interval turns. The output will be printed for turn 0 (initial state), turn interval, turn 2xinterval …. turn number of turns -1. The final turn need not be a multiple of interval.

8) I am providing you with a print function that will display the board in colour on your terminal (it works only for the terminal, it does not work for writing into a file). This should help you when you are debugging.

9) Sample outputs for comparison will be posted in the next few days.