Homework 4 Revision 1

Due: June 6, 2024; late due date is June 11 **Points:** 100

1. In problem 1, if the square is occupied, you need to give the error message. "`%c has played %d,%d\n`" (where "%c" is either "X" or "O", whichever is already in the square, and "%d,%d" are the co-ordinates of the occupied square).
2. In problem 1 and 2, the dimensions of the "O" were added; it is to be 5×5 centered in the square.
3. For all problems, the exit status code is 0.

1. (*25 points*) Enhance the tic-tac-toe game so 2 people can play. To do this, you need to add an"O" that, like the "X", is drawn in the square. Then prompt the user for a square identifier, and alternate between drawing "X"s and "O"s at those locations on the board. The first move is for "X". You need to detect and reject when a user plays a square that is already taken. Stop after the board is full (that is, 9 plays). You do not have to worry about who wins.

   The "O" is to be 5×5, centered in the middle of the square.

   Here are the messages your program should print to the standard output:

   - The tic-tac-toe board, with "X"s and "O"s as appropriate;
   - When it is "X"'s turn, print "`X's turn > `" (note the space after the ">"); and
   - When it is "O"'s turn, print "`O's turn > `" (again, note the space after the ">").

   Here are the error messages; all are to be printed on the standard error:

   - When the user enters only 1 co-ordinate: "`Need 2 co-ordinates\n`" (the '\n' is a newline);
   - When there is an illegal character in the input: "`Illegal character in input "%c"\n`" (the "%c" is to print the offending character);
   - When the square is already occupied: "`%c has played %d,%d\n`" (where "%c" is either "X" or "O", whichever is already in the square, and "%d,%d" are the co-ordinates of the occupied square);
   - When an invalid set of co-ordinates are entered:
     "`%d,%d is not a valid square; the numbers must be between 1 and 3 inclusive\n`"
     (each %d is one of the invalid numbers).

   If the program reads an end of file at the prompt, print a newline and quit.

   The program should exit with an exit status code of 0.

   Please call your program *ttt4a.c* and submit it through Gradescope. A sample executable, used to generate the Gradescope validation outputs, is available on the CSIF at /home/bishop/hw4/ttt4a.

2. (*25 points*) Now enhance what you did for question 1. Have your program determine when the game is over — that is, there is a winner (three in a row, column, or diagonal) or a tie (no winner and all 9 squares on the board are full). Print the results on the standard output:

   - If X wins, print "`Game over!  X won!\n`";
   - If O wins, print "`Game over!  O won!\n`"; or
   - If X wins, print "`Game over!  It's a tie!\n`".

   Please call your program *ttt4b.c* and submit it through Gradescope. A sample executable, used to generate the Gradescope validation outputs, is available on the CSIF at /home/bishop/hw4/ttt4b.

3. (*50 points*) The *birthday problem* asks how many people must be in a room so that the probability of two of them having the same birthday is 0.5. This problem has you explore it by simulation. Basically, you will create a series of lists of random numbers of length $n = 2, \ldots$, and look for duplicates. You will do this 5000 times for each length. For each length, count the number of lists with at least 1 duplicate number; then divide that number by 5000. That is the (simulated) probability that a list of n generated numbers has at least one duplicate. As the random numbers you generate are between 1 and 365 (each one corresponding to a day of the year), this simulates the birthday problem.

   Now, breathe deeply and calm down. We will do this in steps; you only have to turn the final program in to Canvas (*not* Gradescope).

   (a) First, detecting duplicates. Write a function called `hasduplicates(bday)` that takes an array `bday` and returns 1 if it contains a duplicate element, and 0 if it does not. For example, if `bday` is

   ```
   iint bday[] = { 1, 2, 3, 4, 5, 5, 2 };
   ```

   then `hasduplicates(bday)` returns 1 and if

   ```
   int bday[] = { 1, 2, 3, 4, 5, 6, 7 };
   ```

   then `hasduplicates(bday)` returns 0.

   (b) Now, deal with one set of birthdays. Write a function called `onetest(count)` that generates a list of `count` random integers between 1 and 365 inclusive, and returns 1 if it contains a duplicate element, and 0 if it does not. Please use the function `hasduplicates(bday)` to test for duplicates.

   (c) Now for the probability for *count* people. Write a function `probab(count, num)` that runs *num* tests of *count* people, and counts the number of tests with duplicates. It returns the fraction of the tests with duplicates; that is, the number of duplicates divided by *num*.

   (d) Now for the demonstration. Start with 2 people, and begin adding people until the probability of that many people having two people with a birthday in common is over 0.5. (In other words, start with a list of 2 elements, and increase the number of elements in the list until the simulation shows a probability of 0.5 that a number in the list is duplicated.) Print each probability; your output should look like this:

   ```
   For  2 people, the probability of 2 birthdays in common is 0.001400
   For  3 people, the probability of 2 birthdays in common is 0.006600
   For  4 people, the probability of 2 birthdays in common is 0.015200
   For  5 people, the probability of 2 birthdays in common is 0.025400
   For  6 people, the probability of 2 birthdays in common is 0.041400
   For  7 people, the probability of 2 birthdays in common is 0.053000
   For  8 people, the probability of 2 birthdays in common is 0.082000
   For  9 people, the probability of 2 birthdays in common is 0.092200
   For 10 people, the probability of 2 birthdays in common is 0.121800
   ```

   Hint: Don't be surprised if your probabilities are slightly different than the ones shown in the sample output. As randomness is involved, it is very unlikely your numbers will match the ones shown here.

   To turn in: Please call your program *bday.c* and submit it through Canvas (*not* Gradescope). A sample executable is available on the CSIF at /home/bishop/hw4/ttt4b.