Introduction to C - Programming Assignment #4

Due date: *October 28, 2011, 11:59pm*

Objectives

1. To review using arrays to store information.
2. To learn how to follow given function prototypes.
3. To learn how to integrate different functions into a single application.
4. To learn how to call functions and utilize function reuse.

Problem: Computer Games (games.c)

Everything with the wedding planning is going smoothly now, but a new problem has arisen. Whenever Arup gets home from work, he can't get any grading done because his fiance's daughter insists on playing games with him.

This is very bad because his C programming students are having to wait very long to get their grades on their assignments and they are getting antsy about how they are doing in the class.

Arup has heard the complaints and has come up with a brilliant idea. He thinks that if the class can write him some fun (solitary) computer games that his fiance's daughter can play on her own, then he can be left alone at home to grade assignments!

Of course, because Arup cares very much about mathematics education, he wants both of the games you design to foster mathematical skills, so that his fiance's daughter can get ahead in mathematics.

He has chosen the following two games:

(1) Guessing Game
(2) Square Game

Guessing Game

Most people are familiar with the guessing game. A random number is chosen from 1 to 100 and the user makes a guess as to what that number is. If the number is correct, they win! If they don't the user is told whether their guess was too low or too high. This feedback should help the user make a better guess the next time around. The sequence continues until the user guesses the correct value.

In this version of the game, the user can choose the maximum value. **If the user chooses a value that isn't in between 2 and 32767, reset the value to 100.**

Square Game
The user is presented with a random 3 x 3 square of integers, all of which are in between 1 and 20 inclusive, at the beginning of the game. The goal of the game is to change some of the numbers in the square so that the sum of all three rows and all three columns is the same. Note that this is different than a Magic Square because the numbers are not required to be distinct AND there is no restriction on what the two diagonals must sum to.

For each turn, the user is asked which entry they want to change (row and column). If the user does not enter a valid row, column combination, they will be reprompted to enter a new row and column. This will continue until they enter a valid (row, col). Then, they are asked what value they want to change that entry to.

The game continues until the resulting square contains rows and columns such that they all sum to the same exact value.

Implementation Requirements
You must implement this program by filling in the scaffold, *games-scaffold.c*, provided with the assignment and you must follow all the function preconditions and postconditions given. If you can't see how all the functions fit together, go see a TA or your instructor for some hints. **When you turn in the completed project, please name the file games.c.**

Input Specification
The users will enter the correct type of information requested at all times. However, the user may not enter a valid menu choice (1, 4) or valid row and column in the Square Game.

Output Specification
For each game, the appropriate feedback is given. When the user wants their total score, this should be printed to the screen. The user should be able to quit as well.

Output Sample
Two of these are provided separately in the files cop3223-p4-output1.txt and cop3223-p4-output2.txt.

Deliverables
The source file, *games.c*, containing the completed project, should be submitted over WebCourses. While this file may contain extra functions not defined in *games-scaffold.c*, it must contain completed versions of all the functions in *games-scaffold.c*.

Restrictions
Although you may use other compilers, your program must compile in gcc and run in the Code::Blocks environment. Your program should include a header comment with the following information: your name, course number, section number, assignment title, and date. Also, make sure you include comments throughout your code describing the major steps in solving the problem.

Grading Details
Your programs will be graded upon the following criteria:

1) Your correctness

2) Your programming style and use of white space. Even if you have a plan and your program works perfectly, if your programming style is poor or your use of white space is poor, you could get 10% or 15% deducted from your grade.

3) Compatibility to gcc in Code::Blocks. If your program does not compile in this environment, you will get a **sizable** deduction from your grade.