Assignment 2 (55 points)
Fall 2014
Shlomo Hershkop
CIS 542
Out: Sept 24, 2014
Due: Midnight Oct 15

Programming II

Step 1 - Puzzles (35 points)

In this step, you will develop code that computes a magic square. A magic square is a matrix filled with integers in the range [1 – n^2] so that each entry in the matrix has a unique value. The sum of each of the rows and each of the columns and each of the two diagonals must be equal the same number. The number is known as the magic constant Example for n=3 the magic constant is 15, for n=4 it will be 34 etc. Also Note that there not a single unique solution to the magic square problem.

think of Typedef as a shortcut so instead of having to type out a full type you can use the shortcut. For example

typedef int XX[15];
XX yyy;

will make yyy a shortcut for a 15 array int. So can say yyy[0] etc

Create a file called **Maintest1.c/Magicsquare1.c/Magicsquare.h** and (to help you get started) start by putting the following code/declarations in each file (where appropriate):

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define MAGICSIZE 3

typedef int MSQUARE_TYPE[MAGICSIZE][ MAGICSIZE];

typedef MSQUARE_TYPE * MagSquare_PTR;

void initSquare( MagSquare_PTR, int magicsquaresize );
void printSquare( MagSquare_PTR, int magicsquaresize );


int main() {
```

```
srand( time( NULL )); /* seed the random number gen */

MagSquare_PTR mptr = /* replace with something with malloc
to create space to point at*/;

initSquare( mptr, MAGICSIZE );

printSquare( mptr , MAGICSIZE);

} // end of main()
```

Define the function **initSquare(**`MagSquare_PTR, int size`**)** which takes the magic square pointer argument (as above) and the size of the n by n array and fills it with values in the range [1 - 9]. These values must be assigned RANDOMLY, **but** make sure that no number appears more than once in the array.

Also Define the function **printSquare(**`MagSquare_PTR, int size`**)** which takes the magic square pointer argument (as above) and the n for the n by n size, and prints out its contents to the screen in a nice n rows by n columns format.

You should test your code with just these 2 functions to make sure each is working (before adding the rest of the functions).

Hint: To get random numbers look up lookup rand() and srand() from the stdlib libary.

1. maintest1.c – this will contain the main function
2. magicsquare.h – this will contain the functions declarations that you will be using.
3. magicsquare.c – this will contain the function definitions from above including the following methods:
    a. void initSquare(`MagSquare_PTR, int`)
       (see above)
    b. void printSquare(`MagSquare_PTR, int`)
       (see above)
    c. int sumColumn(int column, `MagSquare_PTR, int  size`)
       The function takes the 2-dimensional magic square array and a column number and returns the sum of the 3 values in that column
    d. int sumRow(int row, `MagSquare_PTR, int size`)
       see above.
    e. int sumDiagonal(int diagonal, `MagSquare_PTR, int size`)
       The function takes the 2-dimensional magic square array and a diagonal number and returns the sum of the n values in that diagonal. Use diagonal = 0 to refer to the diagonal that runs from the northwest corner down to

the southeast corner. Use diagonal = 1 to refer to the diagonal that runs from the northeast corner down to the southwest corner.

f. Create a function called int isMagic(`MagSquare_PTR, int size`) which takes the magic square pointer as an argument and checks each of the rows and columns and diagonals to see if all the sums equal 15. The function should return 0 if the argument is not a magic square and 1 if it is a magic square.
g. Modify the above main() so that it calls isMagic and prints out whether the square is a magic square or not.
h. Modify the main() program so that after it prints out the square, it computes (using the functions above) and prints out the sum of each of the 3 rows and each of the 3 columns and each of the 2 diagonals.

Compile and test your program.
Here's a sample output:

```
the square is not a magic square of size 3
here is your square:
     2    8    3
     1    4    5
     7    9    6
sum of row 1 = 13
sum of row 2 = 10
sum of row 3 = 22
sum of column 1 = 10
sum of column 2 = 21
sum of column 3 = 14
sum of diagonal 0 = 12
sum of diagonal 1 = 14
```

Note: you will need to create a makefile for this lab too. For example
`make step1_run`
would compile and execute the step1 code.
Hint: here is a sample of a more complicated Makefile:

```
CC = gcc
CCFLAGS = -Wall -lm

step1_run: maintest1.c magicsquare1.o
     $(CC) $(CCFLAGS) -o test1 maintest1.c  magicsquare1.o
     ./test1

magicsquare1.o:   magicsquare1.c  magicsquare1.h
     $(CC) $(CCFLAGS) -c  magicsquare1.c
```

```
clean:
     rm -f *.o test1
```

(`make clean`
will just clean up all the temp .o files and executables, should run it before submission, if you used emacs, you should probably include *~ also).

Step 2 create a running program (20 points)

Now copy maintest1.c to **maintest2.c** and magicsquare1.c to **magicsquare2.c**, and magicsquare1.h to **magicsquare1.h** modify them as follows:

1. Create a function called **permuteSquare(**`MagSquare_PTR`**)** which takes the magic square pointer as an argument and randomly switches two entries in the array. Do this by randomly picking two sets of row and column indices (in the range [0 - 2]) and then swapping the entries located at each pair of indices. Check to make sure you aren't picking the same spot .. that is swapping the same location with itself
2. Modify the **main()** so that after it calls **isMagic()**, if the square is not magic, then it calls **permuteSquare()** to switch around two entries in the square and then test again to see if the square is magic. Do this repeatedly until a magic square is found. When a magic square is found, call **printSquare()** again to print out the magic square.
3. Have the program count the number of times it has to permute the square in order to find a magic square. Print that out too (Hint: count it where you call method permutesquare).

Compile and test your code. Please ask for help as you need etc

Good luck.