

Program #4: Basic Shell Scripting and C Arrays

Due Date: September 22nd, 2011, at the beginning of class

Overview: We start on scripting, and encounter 2D arrays.

Assignment: There are two parts to this assignment, each detailed below. Submit each part separately using the `turnin` utility on `lectura`, as you've done with previous programs. The submission directory for all parts is `cs352p04`.

1. Really Basic Bash Scripting.

- *Task:* Write a Bash script named `prog04p1.bsh` that uses only `echo` statements to display this following information, in this order, one piece per line, with a label ahead of each piece: username, shell, date, time, current working directory, and the content of the file `/etc/timezone`. Some of this information can be acquired from utilities and some from shell variables (and some from both!).
- *Data:* None! Your script will need no input.
- *Output:* For example, your first two lines of output might look like this:

```
Your username is stdntlm
Your shell is /bin/bash
[...]
```

- *Turn In:* Submit your script file with `turnin` to the submission directory (given above).

2. Merely Basic Bash Scripting.

- *Task:* Write a Bash script named `prog04p2.bsh` that accepts the name of a text file and two integers (a line number and a count) from the command line, and displays from the file 'count' lines from the file starting with line 'line number.' For example, if the integers are 4 and 3, lines 4, 5, and 6 from the file are displayed.

The following are the only error checks that are required: (1) If the user does not supply three arguments, (2) if the file isn't readable, and (3) if either integer is not positive. In all cases, display a suitable message and halt.

To write this script, you'll need to learn about `head`, `tail`, and command substitution, in addition to the basics of command line arguments and selection statements that we've covered in class.

- *Data:* The filename and integers are supplied on the command line (that is, they are not read from `stdin`).
- *Output:* Apart from the error situations, the only output will be the lines of the given text file.
- *Turn In:* Submit your script file with `turnin` to the submission directory (given above).

3. Magic Squares.

- *Task:* Write a complete, well-documented C program named `prog04p3.c` that creates and verifies a magic square.

Magic squares come in a variety of forms. For our purposes, we'll define a magic square to be an $n \times n$ array (where n is an odd number and $n \geq 3$; see also the Output section) which contains the integers from 1 through n^2 arranged so that the sum of each column, each row, and each of the two diagonals is the same. For example, this is the only 3 x 3 magic square (excluding rotations, etc.):

8	1	6
3	5	7
4	9	2

As you can easily verify, the sum of each of the rows, columns and diagonals is 15. The sum of the rows, columns, and diagonals of a magic square of size $n \times n$ is called the magic constant and is equal to $\frac{n(n^2+1)}{2}$.

There are a variety of methods for creating magic squares. The best known is the Siamese method (also known as de la Loubere’s method). Begin by placing 1 in any cell of the square. Now place 2 in the cell diagonally above and to the right of the 1. If that cell is outside of the square, “wrap” it around to the other side. For example, in the 3 x 3 magic square shown above, the cell above and to the right of 1 is outside the square. We imagine that when we move off the top of the square we reappear at the bottom of the column; as you can see, 2 is at the bottom of the column to the right of 1’s column. To place 3, we move above and to the right of 2, which takes us off the right side. We simply “wrap” to the left side and place the 3. When we try to place 4, we see that our rules want to place it where 1 is. In this case, we simply place the 4 below the last value (below the 3) and continue. The process continues until all n^2 cells have been filled.

Unfortunately, the Siamese method doesn’t work with every choice of a starting cell. For example, if we start with 1 in the center of a 3×3 cell, we generate this result, which has a problem:

4	9	2
8	1	6
3	5	7

The rows and columns all add to 15, but the diagonals do not. (This is called a semimagic square.) Thus, it is prudent to test our generated $n \times n$ matrix to verify that it is ‘fully’ magic.

The user is to be able to tell the program, via stdin, the size of the square and the location of the cell containing the value 1. For example, if the user invoked the program like this:

```
$ echo "5 0 2" | a.out
```

the program would generate a 5×5 magic square with the 1 at row 0, column 2. (Assume that the upper left cell is at row 0, column 0.) The program is to display the resulting square, its Magic Constant, a message indicating whether or not the square is truly “magic,” and, if the square is not magic, an explanation as to why it is not magic. See the Output section for an example.

- *Data:* The program accepts three integer values: n (the size of the square), and the location of the value ‘1’ as a row index and column index pair. Each value is separated from the next by at least one white-space character.
- *Output:* As stated above, the program is to display the magic square, the magic constant, whether or not the square is magic, and, if not, why not. Use the following impossible example as a demonstration of the format to follow. Note that the format can handle only cell values less than 1,000 (so what does this imply about legal inputs?), and that the message after the ‘No’ can be in your own words.

```

4 49 2
888 1 36
93 5 147
5005 is the magic constant
No - The upper-left-to-lower-right diagonal has a sum of 152.
```

- *Turn In:* Use `turnin` to submit your completed program to this assignment’s submission directory, which is given at the top of this handout.

General Requirements and Hints:

- Yes, we expect that your C program will perform basic error checking, and will stop with a suitable error message when necessary. This should no longer even be a question; “Just Do It!” (tm)
- Start early, do your own work, keep checking Piazza, etc.