

# COSC 2P91 Assignment 1

## Winter 2012

**Due:** Feb 7th, 2012 at 3:00 PM.

**Late date:** Feb 10<sup>th</sup>, 2012 at 3:00 PM, subject to a 25% penalty. Later submissions receive 0.

This assignment is worth 5% of your final grade. Ensure that you read **all** assignment submission requirements (below) to avoid grade penalties on your submission.

The purpose of this assignment is to familiarize you with C syntax and program structure, and to compare C programming with Java programming. Wherever possible, code should be broken up into functions to maximize reusability. Part of your grade will depend on this.

### Part A – Tic-Tac-Toe

For this part, you are to implement the classic game of Tic-Tac-Toe in C. For a complete description of the game see Wikipedia, at [http://en.wikipedia.org/wiki/Tic\\_tac\\_toe](http://en.wikipedia.org/wiki/Tic_tac_toe).

#### Representation

The board should be represented as a 3x3 2D array of any type of your choosing. When displayed, the rows should be lettered, A, B and C. The columns should be numbered, 1, 2 and 3.

#### Required Functionality

Upon startup, the game should prompt for the number of players (i.e., 1 or 2). It should then repeatedly prompt the player to enter their move in the form of a row letter and a column number, and read this with a single *scanf*. However, your input code here should be as “bullet-proof” as possible – any *reasonable* combination of **exactly one** legal letter and **exactly one** legal number should be accepted (A1, 1A, a 1, 1 a, 2 B, C3, etc.).

Bad input should not crash the program – instead, it should prompt the user to re-enter their input. This could include things like “F 10” (an invalid letter, and/or number), “3 3” (no row letter – although you could reasonably interpret this as C 3 if desired), and garbage input like “hELlo###@!!!”. Consider looking at some of the standard C string functions such as *atoi*, *isalpha*, *isspace*, *toupper*, *tolower* and *strlen*. For simplicity, assume the input string will be no longer than 50 characters.

After entering a move, the board should update and prompt the other player to take their turn. Computer players can simply make a random move using *rand*. This should repeat until the game is finished. Players should also be able to quit by entering a “Q”, or restart the game by entering an “R”.

*(Hint: This is trickier than it sounds. Start by making a working tic-tac-toe game that assumes perfect input all the time. Then adjust the input routines to detect/compensate for bad input as described above.)*

### Part B – C/Java Speed Comparison

Investigate if there’s any truth to the age-old rumor that C programs are faster than Java programs. You are to write two virtually identical programs, one in C and one in Java. Both will use a brute force approach to find prime numbers. They should be as similar as possible, and utilize identical algorithms for finding primes. In fact, due to the syntax similarity between the languages, you most likely can re-

use parts of the programs in each other. Of course, the I/O code, and basic structure of the programs will be different.

You should provide the run-time (in milliseconds) for finding the first 10,000, 100,000 and 1,000,000 primes for both programs. Print these results on a separate page, and clearly mark which time belongs to which program and which trial. For example, something like “Java 10,000 ---> xxxx ms” for each output would be ideal. The corresponding trial times from all submitted assignments will be compared as a group, and the results of the comparison will be presented in class.

*(Hint: A very simple brute force algorithm to find primes involves trying to divide the number being tested by every number from 2 to its half. If the number divides evenly by any of these, it's not a prime. For example, if 10 were being tested, try dividing it by 2, 3, 4 and 5. Since it's divisible by 2, it's not a prime. However, testing 11 would try division by 2, 3, 4, 5 and 6. Since it doesn't divide evenly by any of these, it is a prime. Although this algorithm can be easily improved upon, please don't – we actually want these to take some processing time to provide useful numbers for the comparison.)*

### **Assignment Submission Requirements**

All assignments must be submitted BOTH in hardcopy (print out of any source code and a few samples of its execution) AND in softcopy (electronically) using the command **submit2p91** on Sandcastle. This program will copy the contents of the current directory to the marker's account.

All hardcopy submissions must satisfy the following:

- The submission must be **typed and printed** on a laser/inkjet printer
- All the papers must be stapled together
- A standard assignment cover-page (<http://www.cosc.brocku.ca/forms/cover>) should be printed, filled out completely, signed and stapled to the front of the outside of the assignment (optionally in an 9x12 envelope).

All electronic submissions should include **only** your C source files. Your source files should be named `parta.c` and `partb.c` for Parts A and B respectively.

Submit your assignment to the Assignment Box outside of J332, in COSC 2P91 slot, before the due time indicated above. Only one submission should be made per assignment.

- Unless otherwise specified, all assignments are due at 3:00 PM on the due date.
- Both the hardcopy and the softcopy submissions must be done before the deadline.
- Differences between the hardcopy and the softcopy submissions will be penalized.
- Familiarize yourself with the departmental and university policies on plagiarism and academic misconduct. In particular, all referenced sources (code, written material, etc.) **must** be cited.

**Assignments that do not follow these requirements will NOT be marked!**