CSci 2021 Fall 2018 Lab 2 – C Programming

Lab 2 will introduce you to the C Programming Language based on your knowledge of the Java Programming Language. C and Java share most all of the basic control, variable and method syntax. Therefore, Java methods will generally compile with a C compiler with very little modification. However, there are some major differences between C and Java that will require getting used to.

Some C and Java Differences that we will explore:

- C is *not* object oriented; therefore, C does not have classes--making C program structure quite different from that of Java
- Input, output and formatting are different between C and Java
- C naming conventions are different from those used in Java
- There is no designation of private/public for methods and variables in C
- C does *not* include length attributes for arrays and strings
- C does *not* have a specific string type
- C makes extensive use of pointers (memory addresses), but Java conceals this information
- C does not include the *boolean* type; instead, *false* is a 0 value, and *true* is any *non*-zero value

In the steps that follow, we will start to adjust to these differences. Sometimes, the differences will be learned by inspection, but in other cases, we will specifically provide instruction on adjusting to the differences. This week, we will start to address the first four points above; next week, we will start to address the remaining four points.

But first, some information about the format of the labs.

Lab Procedures for the Semester

Lab work is done in groups of *two*. You may also work alone if there is space, but generally working with a partner is more beneficial than working alone. Take some time to mingle with those around you and feel free to move around so that you are seated next to someone who you think will work well with you.

The labs will be different in structure from week to week. Sometimes, they will be a lab format consisting of 2-5 steps. Other weeks will be a discussion or question/answer format. In general, all lab work is to be completed within the lab time rather than outside of lab. Your TAs will be there to help you and check off your completion of the lab steps. The steps should be done in order. When you finish a step, have a TA check it. When working on lab steps, *both lab partners should equally contribute* to the solving of the problems. A guideline is to trade off the typing aspect and have each lab partner do half of the typing.

Lab Grading Policy – The labs are designed to be completed during lab time. Therefore, attendance will be taken, and steps will be checked off during lab time. A step will be checked off when you can demonstrate to a TA that the step has been completed correctly. Sometimes, a TA will ask you to answer questions about your work as part of the check off process. Any work that remains after lab time should be finished (for your own benefit) on your own. Then, you can have that remaining work checked off during any TA office hours up until the end of office hours on Monday of the following week. But, office hours check off is limited to a small amount of remaining work. For example, you

cannot have the first or second step of a lab checked off during office hours.

Step 1: C Program Format and Simple Output.

Using the editor of your choice, enter the following code and put it into a file called `io.c` . Unlike Java, the file name actually does *not* matter, but it is appropriate to use the "`.c`" suffix on all C programs.

```
#include <stdio.h>

int main() { /* note the int return designation on main() */
    int x = 100;
    int y = 200;
    int total = x + y;
    printf("\nWelcome to Lab 2\n");
    printf("\nThe sum of %i and %i is: %i", x, y, total);
    printf("\nOr another way: ");
    printf("\nThe sum of %i and %i is: %i", x, y, x + y);
    printf("\nOK\n");
}
```

Some things to observe:

- `#include <stdio.h>` includes the header file `stdio.h` . The "#include" is a compiler directive that adds the text of the header file `stdio.h` to the text of your program. Header files contain method *signatures*—in this case, signatures for the standard I/O methods. *Like* Java, I/O is *not* directly part of C syntax; it is accomplished through the calling of I/O methods. The C compiler needs to know what those method signatures look like in order to determine whether you are calling them correctly. The <> around `stdio.h` indicates that this header file is part of the standard C library, as opposed to a file of our own. Include `<stdio.h>` on all programs. A similar header file `<math.h>` exists for the math library which contains functions such as sqrt(), sin(), cos().

- The `main()` method looks simpler than in Java, but it accomplishes the same thing. Unlike Java, since there are no classes, the `main()` method is placed directly in the file. Other than the name of the file, there is no specific naming of C programs.

- Output here is via `printf()` rather than `println()`; note that `printf()` is also supported in current Java versions. The major difrerence between `printf()` and `println()`is that `printf()` uses format specifiers as temporary place holders within the print string. Some common place holders are:

  `%i` or `%d` for integers
  `%f` for float and double (with optional field width specifiers)
  `%s` for string (a string in C is an array of characters)

  The values for the place holders are listed in order, separated by commas after the print string.

We will be using the freeware C compiler called gcc. gcc is available for Linux, Windows and Mac systems. Compile and run your program with gcc as follows: `gcc io.c -o io`
For reference, the `-o` option on gcc allows you to specify the name of the (compiled) binary file created by gcc. In this case, the name `io` was chosen, but you could call it something else. The `-o` option is—an option. If `-o` is *not* used, the default binary file is called `a.out` .

Assuming a clean compile, run the compiled binary with: `./io`

Now, modify `io.c` to print out a *table* of the square roots of the number from 1 through 15. Your table should have two labelled columns. One column for the integer value and a second column for the square root. F*ield width specifiers* will allow you to align values. For example `%10d` will format an integer with a minimum field width of 10 right justified, and `%10.2f` will format a double precision floating point number in a field with of 10 (including the decimal point and potential leading minus) with two places to the right of the decimal point. Use a `for` loop to generate your values and square roots.

You will be using the `sqrt()` function from the math library to find the square roots, so be sure to include the `math.h` header file at the start of your file. In addition, when compiling, the math library must be linked into the resulting binary. So, your compile command will look like this:

```
gcc io.c -o io -lm
```

Now, have a TA check your work. Show the TA your program and your output. You may be asked questions.

Step 2: Simple Text Graphics.

Now work with `printf()` to write a program that will produce a "votebox" of size `n`, where `n` is a local integer variable that can be given a hardcoded value. But, write your code so that it is based upon `n`, so that if you change the hardcoded value of `n`, you can change the size of the votebox. Assume that *minimum* size for a votebox is 3. Call your program `votebox.c` .
An example votebox of size 6 is:

```
------
|\  /|
| \/ |
| /\ |
|/  \|
------
```

An example votebox of size 7 is:

```
-------
|\   /|
| \ / |
|  X  |
| / \ |
|/   \|
-------
```

Note that for voteboxes that are of odd size, there is an "X" in the middle.

You will notice that the votebox will require the printing of the backslash. The backslash is also the

escape character (used in `\tab`, `\n`, etc.). To print a backslash, use `\\` .

When you have your program working and have tested it thoroughly, have a TA check this step.

Step 3: Methods in C.

As mentioned earlier in this writeup, the C compiler relies on header files (`.h` files) to verify that methods are called properly within your program. However, like in Java, it is also possible to declare your own functions within the file you are writing, so as to break your code into smaller procedures that you can call multiple times. Unlike Java, however, C requires that you declare the method (or at least the method's signature) before the method is called. Method declaration in C is very similar to that in Java. For example, here is a method to print out the product of two arguments:

```
void multiply_two_values(int x, int y) {
    printf("\nThe product of %d and %d is %d\n", x, y, x * y);
}
```

[Note that instead of camelCase which is used for longer Java identifiers, the C convention is to use all lower case with "_" between words. Apart from that, naming of variable and methods in C is similar to naming in Java.]

Turn most of your votebox program from Step 2 above into a method that takes one integer parameter—the size of the votebox. Call the method `vote_box()`. It should be a `void` method. Your `main()` program method should just contain a simple call to `vote_box()` with an integer value. To further modularize your program, you might also make a method to print the top/bottom row of the votebox. Since the top row will always be the same as the bottom row, writing a single method that can be called for both the top and bottom row will eliminate the need to duplicate this code in your program.

When your program compiles cleanly, try several test cases and have a TA check this step.

That's it for today!