ENCM 339: Programming Fundamentals
Lab 1 – Thursday September 21, 2017

Department of Electrical & Computer Engineering
University of Calgary

M. Moussavi

Acknowledgment: There are some materials and documents in this lab that were originally written by Dr. S. Norman for a previous version of this course.

Objectives:

This lab consists of several exercises, mostly designed to help you to:

- Get familiarized with the program development environment in our ICT 320 lab, using Cygwin, Linux commands, and `gcc`.
- Review some of the basic programming features in C such as control structures and functions.
- Practice simple uses of the C library functions `printf` and `scanf`.
- Practice drawing AR diagrams

Please heed this advice:

- You are strongly advised to study lab documents before coming to your scheduled lab sessions.
- Some exercises in this lab and future labs will not be marked. Please do not skip them, because these exercises are as important as the others in learning the course material.
- In courses like ENCM 339 some students skip directly to the exercises that involve writing code, postponing the diagram drawing until later. That's a bad idea for two reasons:
  - Drawing diagrams is an important part of learning how to visualize memory used in C and C++ programs. If you do diagram-drawing exercises at the last minute, you won't learn the material very well.
  - If you do the diagrams first, you may find it easier to understand the code-writing exercises, so you may be able to finish them more quickly.

Due Date:

In every scheduled lab session in ENCM 339 you will find two types of lab exercises:

- **In-lab exercises** that must be always handed in on paper **by the end of your scheduled lab period**, in the assignment boxes located on the second floor of the ICT building, in the hallway on the west side of the building near the elevators.
- Post-lab exercises normally must be submitted electronically using the D2L Dropbox feature. All of your work should be in a single PDF file that is easy for your TA to read and mark. For instructions about how to make a post-lab PDF file, study the posted document on the D2L under the folder “Help Documents for Lab Assignments”. These documents were originally written for the fall 2015 semester, but they are all applicable to this term. Please ignore if some of the examples refer to September 2015.

  Due dates and times for post-lab exercises are:
  - **Lab Section B03:** Thursday September 27, before 9:30 AM.
  - **Lab Section B04:** Thursday September 27, before 2:00 PM.

  20% marks will be deducted from the assignments handed in up to 24 hours after each due date. That means if your mark is X out of Y, you will only gain 0.8 times X. There will be no credit for assignments turned in later than 24 hours after the due dates; they will be returned unmarked.

Marking Scheme:

You shouldn't submit anything for the exercises that are not marked.

Exercise	Type	Marks
A	in-lab	no marks
B	in-lab	8 marks
C	post-lab	4 marks
D	post-lab	4 marks

Exercise A: Creating a C source file, building and running an executable

This is an in-lab exercise and will not be marked

Read This First:

There are no marks for this exercise, but if this is your first attempt to develop a C program, please do it so that you start to become comfortable with program development in C and particularly using the Cygwin Terminal and Notepad++.

Before starting this exercise, please find the document titled "Editing and Running Programs in the ENCM 339 Lab" on D2L, and read it carefully. (To find the document, first find the "Help Documents for Lab Assignments" sub-module within the "Lab Assignments" module.) It might be a good idea to print the document so that you can refer to it while your computer screen is filled with other material.

What to Do:

Examples and explanations for all of the following steps can be found in "Editing and Running Programs in the ENCM 339 Lab".

- If you haven't already done so, make a folder called ENCM339 in your H: drive.
- Within your ENCM339 folder, make another folder called lab1.
- Start up Notepad++ using the Windows Start button. Within Notepad++ you should see a blank editing tab called new 1 (or some similar name). Use 'Save As' to save this tab as a file called `lab1exe_A.c` , in the lab1 folder that you just created.
- Into the same edit tab, type in all the following C code:

```
#include <stdio.h>

int main(void)
{
  int a = 0, b = 0;
  printf("Please enter a value for variable a:\n");
  scanf("%d", &a);
  printf("Please enter a value for variable b:\n");
  scanf("%d", &b);
  printf("The values of a and b are %d for a and %d for b.\n", a, b);
  printf("The value of a %% b is %d.\n", a % b);
  return 0;
}
```

- After you do all that typing, save your work.
- In Cygwin Terminal, make sure that your lab1 directory is the working directory. Then enter the command

```
gcc -Wall lab1exe_A.c
```

If the command succeeds, an executable file called `a.exe` will have been created. If the command fails -- which will be indicated by one or more error Messages--go back to Notepad++ and fix the code that you have typed incorrectly and try `gcc` again.

- Once you have an executable, run it a few times by using the command

```
        ./a.exe
```

over and over. Try different inputs each time; see what happens if you enter letters or punctuation instead of numbers.

Hint: You don't need to type `./a.exe` over and over! You can use the up arrow on your keyboard to retrieve previously entered commands.

If you get stuck on any of the above steps, or if they all seem to work but you're not sure what is happening, please don't hesitate to ask for help!

What to Submit:

There is nothing to be submitted for this exercise.

Exercise B: Drawing AR Diagrams for a Simple C Program

This is an in-lab exercise.

Read This First:

This is a very simple exercise on activation records that we discussed during the lectures.

Note: To learn more about activation records, pleaser read the slides 42-46, in the set of slides posted on the D2L, called: “03_Basic_Constructs”. If you need further help study the document “Activation Records and the Stack” posted under the “Help Documents for Lab Assignments”, on the D2L. This document was written by Dr. Steve Norman for an earlier version of this course.

What to Do:

Step-one: Read the following program carefully and draw memory diagrams for points 1, 2, 3, and 4.

```
#include <stdio.h>
int foo (int x);
int jupiter(int x);
int mercury(int x, int y);

int main(void){
    int x = 20, y = 30, z = 10;
    y = foo(x++);
    y = jupiter(z/x);
    // point 4
    return 0;
}

int mercury(int x, int y) {
    int z;
    z = x + 2 * y;
    // point 2

    return z;
}

int jupiter(int x){
    int y = mercury(3 % 4, ++x);
    // point 3
    return y++;
}

int foo(int x){
    // point 1
    return ++x;
}
```

Step-two: type the given code into your Notepad++ editor and save your file with the name `lab1exe_B.c.`
Step-three: add calls to the `printf` function at each point (1 to 4) to display the values of the arguments and the local variables that are accessible at each point. Then, compile and run the program to check if the values in your AR diagrams match with the program output. If not, and you're not sure why your values do not match, please don't hesitate to ask for help!

What to Submit:

Submit your AR diagrams, your modified code with calls to the printf function, and finally your program output as part of your in-lab report.

Exercise C: Using Logical Expressions in C

This is a post-lab exercise

Note: The C program in this exercise must be compiled and run from the command line. If you are not sure how to do that on the operating system on your computer, you are recommended to try it in our ICT 320 lab, within the Cygwin environment.

Read This First - A brief note on the scanf library function:

`scanf` is a standard C Library function that can be used to read terminal input into variables. The first argument to `scanf` is a string constant with one or more format specifiers, like `%d` or `%lf`, and the remaining arguments are the addresses of the variables into which `scanf` will put the input data. Here is the list of format specifiers for **some** of the simple C data types:

C Data type	Format Specifier
`int`	`%d`
`float`	`%f`
`double`	`%lf`
`char`	`%c`
`long int`	`%ld`

For example, if **x** is a double data type, you should use the following syntax to put the input into **x**:

```
        scanf(“%lf”, &x);
```

To use this library function you need to have the following pre-processor directive at the top of your file:
`#include<stdlib>`

`scanf` also returns an integer number which indicates the number of input items that `scanf` has successfully read from the keyboard. For example in the following code segment `scanf` is supposed to read three integer numbers and put them into the variables a, b, and c.

```
int a, b, c, nscan;

printf("Please Enter three integer values: ");
nscan = scanf("%d%d%d", &a, &b, &c);

if (nscan != 3) {
   printf("Error: invalid input(s). I quit.\n\n");
   exit(1);
}
```

If the user enters invalid input, the return value of the `scanf, nscanf` will have a value of zero, one, or two depending on how many of the inputs were properly entered. In other words, if `nscanf` is not equal to 3 the program will terminate.

The point of this exercise is to give you a little easy practice creating expressions with && and ||.

Some students are unclear about the meaning of the term *non-decreasing*. In mathematics, a non-decreasing sequence is defined to be a sequence in which each number is *less than or equal to* the following number. So (10, 20, 30) is non-decreasing, and so are (5, 5, 15) and (20, 20, 20). But (10, 20, 15) is *not* non-decreasing.

Read This Second – I/O Redirection

A common method that users can send input to a C program is by entering the input on the keyboard, and pressing the 'enter/return key'. However the other powerful feature is to redirect the program input to a file. In other words, we can enter a text-file name on the command line after an input-redirection operator, <, and when the program is expecting to read some input from the keyboard it will use the content of the given file as the user's entry. Similarly, you can use the output-redirection operator, >, to send the program's standard output (the program output that normally is displayed on the screen) to a text file.

Let's assume we have a text file called `input.txt` that contains the following data:
```
red
pink
yellow
apple
junk
pear
123
orange
```

Now, while you are in the same directory that file `input.txt` is located, use the Linux sort command (on the Cygwin command line), in the following format to produce a text file called `sorted_input.txt`:

```
sort input.txt<input.txt>sorted_input.txt
```

Now if you open the file `sorted_input.txt`, the content of the file will be:
```
123
apple
junk
orange
pear
pink
red
yellow
```

We are going to use this feature in this exercise to use the data in a text file as inputs for a C program.

What to Do:

Download the file `lab1exe_C.c` from D2L. Create an executable by using the following compilation command:

```
gcc –Wall lab2exe_C.c
```

Then run the program by entering the following command (if running in our ICT 320 lab):

```
./a.exe
```

The program will prompt you to enter three **positive** integer numbers for three variables a, b, and c, it then gives a message whether the three numbers are the same or not. It repeatedly prompts you to enter three integers, unless that you enter a negative value of any the three variables. In that case it terminates. Here is a sample dialogue between program and the user:

```
Please enter a positive value for the int variable a: 2

Please enter a value for the int variable b: 2

Please enter a value for the int variable c: 2
The numbers are: a = 2, b = 2, c = 2. They are the same


Please enter a positive value for the int variable a: -3



Program terminated....
```

As this sample run shows the program stopped when user entered -3 for variable a.

Now you should download the file `input_1.txt`, and this time you should run it by entering the following command:

```
./a.exe < input_1.txt > output_1
```

File `input_1.txt` contains the following data (each row has three values and they will be used by the program as user inputs for a , b, and c). Also notice that last line has a negative number to terminate the loop (pay attention, how break stops the program when finds a negative number for any of the three inputs.)

```
2   2    2
3   3    3
3   5    6
-1  6    7
```

The indirection operator > writes the program's output in a text file that we called it `output_1`. In this run you will not see the program output because it goes into the file called `output_1`. Now if you open the file `output_1` in a text editor such as notepad++ you will see the program's output shown in the following figure.

```
Please enter a positive value for the int variable a:
Please enter a value for the int variable b:
Please enter a value for the int variable c: The numbers are: a = 2, b = 2, c = 2. They are the same


Please enter a positive value for the int variable a:
Please enter a value for the int variable b:
Please enter a value for the int variable c: The numbers are: a = 3, b = 3, c = 3. They are the same


Please enter a positive value for the int variable a:
Please enter a value for the int variable b:
Please enter a value for the int variable c: The numbers are: a = 3, b = 5, c = 6. They are NOT the same


Please enter a positive value for the int variable a:
Program terminated...
```

What to Do Next:

In the file `lab1exe_C.c` there is another function prototype where its implementation is missing. Your next task is to write the missing implementation of this function. Here is its prototype:

```
int is_non_decreasing(int x, int y, int z);
```

If you read the given interface comments for this functions in the file `lab1exe_C.c`, it is supposed to return true (one) if the values of x, y, and z are in non-decreasing order. Otherwise it should return false (zero).

Once the definition of your function is complete and compiles with no errors, test it again by running your program with the command:

```
./a.exe < input_1.txt > output_1
```

What to Submit:

Submit your lab1exe_C.c file and your program output that shows your program works with the given input file.

Exercise D - Projectile Time and Distance Calculator

This is a post-lab exercise

Read This First

Please go to the “Help Documents for Lab Assignments" module on D2L and find the document called “Function Interface Comments." This document describes the format we will use to document C and C++ functions in lectures, labs, and tutorials, so it's important that you read this document carefully.

For Lab 1, you can skip Section 4.3 of “Function Interface Comments". This section is about pointers and arrays, which is a Lab 2 and Lab 3 topic.

Read This Second

In physics, assuming a flat Earth and no air resistance, a projectile launched with specific initial conditions will have a predictable range (maximum distance), and a predictable travel time.

The range or maximum horizontal distance traveled by the projectile can be approximately calculated by:

$$\boldsymbol{d} = \frac{\boldsymbol{v}^2}{\boldsymbol{g}} \,\mathbf{sin}\,(\mathbf{2\Theta})$$

Where:

*g is gravitation acceleration (*9.81 m/s^2)
θ: the angle at which the projectile is launched in degrees
v: the velocity at which the projectile is launched
d: the total horizontal distance travelled by the projectile

To calulate the projectile travel time (t), when it reaches the maximum horizontal distace the followig formaula can be used :

$$\boldsymbol{t} = \frac{\boldsymbol{2vsin\Theta}}{\boldsymbol{g}}$$

In this exercise you will complete a given C source file called `lab1exe_D.c` that prompts the user to enter a projectile’s initial launch velocity (v), and displays the table of maximum horizontal distance and travel time for the trajectory angles of 0 to 90 degrees.

What to Do:

First, download file `lab1exe_D.c` from D2L. In this file the definition of function main and the function prototypes for four other functions are given. Your job is to complete the definition of the missing functions as follows:

Function create_table: which is called by the main function, receives the projectile initial velocity and displays a table of the projectile's maximum travel distance (d) and time (t), for trajectory angles of 0 to 90 (degrees), with increments of 5 degrees. Here is the sample of the required table:

```
 Angle          t            d
(deg)        (sec)          (m)
0.000000     0.000000     0.000000
5.000000     1.778689     177.192018
10.000000    3.543840     349.000146
```

You don't have to worry about the format or the number of digits after the decimal point. The default format is acceptable.

Function projectile_travel_time: receives two double arguments, the trajectory angle (θ), and the initial velocity (ν) and returns the projectile travel time (t).

Function projectile_travel_distance: receives two double arguments, the trajectory angle (θ), and the initial velocity (ν) and returns the projectile maximum horizontal distance (d).

Function degree_to_radian: receives an angle in degrees and converts to radian. This function is needed, because the C library function `sin` needs its argument value to be in radian.

Notes:

- To use the C library function `sin`, you need to include header file `math.h` and use `–lm` option to link the math library:

  ```
  gcc -Wall –lm lab1exe_D.c
  ```

- For constant values of π, and gravitation acceleration, `g`, the following lines are already included in the given file:

  ```
  #define 3.141592654
  #define G 9.8
  ```

What to Submit:

Submit your program and its output as part of your post-lab report.