ENCM 339: Programming Fundamentals
Lab-3 – Thursday October 5th , 2017

Department of Electrical & Computer Engineering
University of Calgary

M. Moussavi, PhD, PEng.

Objectives:

This lab consists of several exercises, mostly designed helping you to understand using and manipulating arrays and C-strings.

Due Date:

In-lab exercises that must be always handed in on paper **by the end of your scheduled lab period**, in the hand-in boxes for your section. The hand-in boxes are on the second floor of the ICT building, in the hallway on the west side of the building.

When you hand in your in-lab exercises, make sure that course name (ENCM 339) and your name and lab section are written in a clear and easy-to-spot way on the front page| it is a waste of time for both you and your TAs to deal with an assignment that doesn't have a name on it. Also make sure that your pages are stapled together securely |pages held together with paperclips or folds of paper tend to fall apart.

Post-lab exercises 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.

Due dates and times for post-lab exercises are:
- **Lab Section B03:** Thursday October 12th, before 9:30 AM.
- **Lab Section B04:** Thursday October 12th, before 2:00 PM.

Important Notes:

- Some post-lab exercises may ask you to draw a diagram that most of the students prefer to hand-draw them. In these cases you need to scan your diagram with an appropriate device and insert the scanned picture of your diagram into your PDF file (the post-lab report).
- 20% marks will be deducted from the assignments handed in up to 24 hours after each due date. It 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	4
B	in-lab	8
C	post-lab	6
D	post-lab	8
E	post-lab	6

Exercise A: AR Diagrams with Arrays
This is an in-lab exercise

What to do:

Download the file `lab2exe_A.c` from D2L. Read the file carefully. Predict the program output; build and run an executable to check your prediction.

Make memory diagrams for point one, which appears within the definition of `reverse`.

Submit your memory diagram as part of your in-lab report.

Exercise B: AR Diagrams with C-String

This is an in-lab exercise

What to do:

Download the file `lab2exe_B.c` from D2L. Read the file carefully. Predict the program output; build and run an executable to check your prediction.

Make memory diagrams for point one and two. Please notice for point one you need to draw the diagram when program reaches this point for the first time.

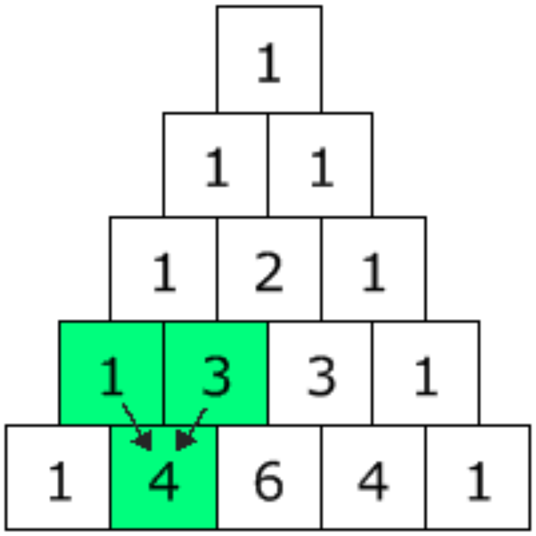
Submit your memory diagrams as part of your in-lab report.

Exercise C - Problem Solving:

This is post-lab exercise

Read This First:

`Pascal's triangle` is a famous arrangement of numbers in mathematics. The first 5 rows of this triangle look as follows (row 0 to 5):



Let $\mathbf{P}_{i;\,j}$ be a number in row **i** of the triangle, and let the index **j** go from 0 up to **i**. Then:

$$\mathbf{P}_{i,\,j} = \begin{cases} 1, \text{ if } j = 0 \text{ or } j = i \\ \mathbf{P}_{i-1,\,j-1} + \mathbf{P}_{i-1,\,j}, \text{ otherwise} \end{cases}$$

For example:

$P_{4,1} = P_{3,0} + P_{3,1} = 1 + 3 = 4$

What to Do:
Download lab3exe_C.c from D2L. Your job is to complete the definition of a missing function called `pascal_triangle`. This function is supposed to write the first N rows of Pascal's triangle on the screen, for values of N with N > 0 and N <= 20. The number of rows will be received from user with the given main function. Function `pascal_triangle` that receives the number of rows from main, as its argument, should print the triangle in the following format (figure shows Pascal's triangle for 6 rows):

```
Row 0:    1
Row 1:    1     1
Row 2:    1     2     1
Row 3:    1     3     3     1
Row 4:    1     4     6     4     1
Row 5:    1     5    10    10     5     1
```

Hint: Always you need to keep track of the values in two rows (current row and previous row of the triangle). Therefore in function `pascal_triangle` you can declare two integer arrays (say current-array and previous-array), with maximum N elements, plus some pointers that point to these arrays. After calculation of values in each row the current-array should be used as previous array to calculate the value of current-array. Also notice that when j == 0 or j == i the values in the table are 1.

Submit the printout of your `pascal_triange` function and the program output for the case that triangle has 9 rows.

Exercise D: Writing functions that work with arrays

This is post-lab exercise.

Read This First:

In ENCM 339 exam we use function interface comments as an important part of program documentation. In this section a brief explanation in this regard is given. For more details please read the Function Interface Comment posted under the "Help Documents for Lab Assignments" on the D2L.

Lets consider the following function interface comment:

```
int largest(const int *a, int n);
/* REQUIRES
 *   size_a > 0.
 *   Elements a[0], a[1], ..., a[n - 1] exist.
 * PROMISES
 *   Return the largest value of a[0], a[1], ... a[n - 1]. */
```

This function obviously requires that value of n not to be negative. However the next line:

```
      Elements a[0], a[1], ..., a[n - 1] exist.
```

says that array a is big enough to have elements with indices up to and including size_a-1. When certain requirements are mentioned under the required, it means that, this function is not responsible to test if the value of n is negative or if the array's memory space is adequate or not (no error checking for those requirements are needed). As we discussed during the lectures, C compilers cannot check if someone tries to do something with the elements of an array beyond the boundaries of an array. Nor there is any way that you can check such a condition.

What to do:

Download the file `lab2exe_D.c` from D2L. In this exercise the definition of two functions are missing and as result the program cannot do something useful. Your job in this exercise is to take the following steps to complete the definition of the missing functions:

Step 1. Read the file `lab2exe_D.c` carefully. Pay attention to the syntax of the function prototypes, function calls, and function definitions.

- Note that whenever any function uses values from an array but does not modify those values the corresponding argument type is `const int*` instead of simply `int*`.
- Note that when reading input, if there is an invalid entry, it ignores those characters and removes them from the input buffer.

Step 2. Compile it and run it. Find out how the program works.

Step 3. The function definition for `select_negatives` is incomplete. Read its interface comment and complete the definition of this function. Compile, run, and test it to make sure it works.

Step 4. The function definition for `substring` is also incomplete. Read its interface comment and complete the definition of this function. Compile, run, and test it to make sure it works.

Submit your source code and the program outputs to demonstrate that your program works for different set of data.

Exercise E: More Practice with Strings

This is a post-lab exercise

Read This First –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 keyboard it will use the content of the given file as 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_nput.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 read a text file into a C program and indicate if each sentence in the given input file is a palindrome phrase or not.

Read This Second:

A **palindrome** is a word, phrase or number which has the property of reading the same in either direction. The word "palindrome" comes from the Greek words *palin* ("back") and *dromos* ("racecourse"). Here are some examples:

```
"Madam I'm Adam."
"Radar"
123321
Can I attain a 'C'?
Can I attain a $$C?
```

As the example shows the adjustment of leading spaces, trailing spaces and spaces between letters is generally permitted. The removal of any non-alphanumeric characters (such as punctuation or random $ signs) is also allowed. In the above examples if we remove all of the non-alphanumeric characters and convert all letters to lower case, the strings will be spelled the same from both ends:

```
madamimadam
radar
123321
caniattainac
caniattainac
```

What to do

1. Copy files `palindrome.c`, and `palindrome.txt`, from D2L.
2. Read the files and understand what the program is doing.
3. Build and run the program.
4. If you are running the program in ICT 320, while you are in your working directory enter the following command (I assume your executable file is `a.exe`):

   ```
   ./a.exe < palindrome.txt
   ```

   If you are working on a Windows machins click Start Button, and type 'cmd' into the box: Search Programs and Files. Then you should see a window with a black background and allows you to enter commands. Again you have to make sure you are in the working directory for this exercise before running your program.

   The program should produce an output which its first three lines are:

```
"Radar":  is not a palindrome.
"Madam I'm Adam":  is not a palindrome.
"Alfalfla":  is not a palindrome.
…
```

Certainly something is wrong in this program! None of the above strings are indicated as palindromes. If ignoring the lower/upper case and spaces, etc. the first two statements must been detected as palindrome.

Note: Although not necessary in this exercise, but you could also redirect the program output into another file if you wish.

5. Take the following steps to fix this problem:

   a. Uncomment the lines commented out in the main function, confined between `#if 0` and `#endif`. It means you should change `#if 0` to `#if 1` (Note: we will discuss and learn more about how `#if 0` works in near future).
   b. Read the function interface comments in `palindrome.c` and write the definition of the the two functions: `is_palindrome` and `strip_out`.
   c. Compile the program and run it again. If your functions work, the following output should appear on the screen:

```
"Radar":  is a palindrome.
"Madam I'm Adam":  is a palindrome.
"Alfalfa":  is not a palindrome.
"He maps spam, eh?":  is a palindrome.
"I did, did I?":  is a palindrome.
"       I prefer pi.":  is a palindrome.
"Ed is on no side":  is a palindrome.
"Am I loco, Lima?":  is a palindrome.
"         Bar crab.":  is a palindrome.
"A war at Tarawa.":  is a palindrome.
"Ah, Satan sees Natasha":  is a palindrome.
"     Borrow or rob?":  is a palindrome.
"233332":  is a palindrome.
"324556":  is not a palindrome.
"Hello world!!":  is not a palindrome.
"     Avon sees nova  ":  is a palindrome.
"Can I attain a 'C'?":  is a palindrome.
"Sept 29, 2005.":  is not a palindrome.
"Delia failed.":  is a palindrome.
"Draw nine men $$  inward":  is a palindrome.
```

Note: In this exercise you can use C library functions such as:

```
strlen(s);                 /* returns length of string s */
islower(ch);               /* returns 1 if character ch is a lowercase letter, 0
                              otherwise */
isupper(ch);               /* returns 1 if character ch is an uppercase letter,
                              0 otherwise */
ch = tolower(ch);          /* converts ch to lower case  */
isalnum(ch);               /* Returns 1 if ch is an alphanumeric character,
                              0 otherwise */
```

As part of you post-lab report, submit your source code, and your program's output that show your program works.