ENCM 339: Programming Fundamentals
Lab-4 – Thursday October 12th , 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 the concept of pointer arithmetic, c-string manipulation, and an introduction to C preprocessor function-like macros.

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 19th, before 9:30 AM.
- **Lab Section B04:** Thursday October 19th, 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 (Total – 36 marks):

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

Exercise	Type	Marks
A	in-lab	3
B	in-lab	8
C	in-lab	2
D	post-lab	9
E	post-lab	14

Exercise A: Pointer arithmetic in C

This is an in-lab exercise

What to Do:

Download the file lab4exA.c from D2L. Draw an AR diagram for point one

What to Submit:

As part of your in-lab report submit your diagram.

Exercise B: More Practice on Pointers and Pointer Arithmetic

This is an In-lab exercise

Pointers and pointers arithmetic are important features of C and C++, and we decided to give you one more exercise on this topic. The file that you will be using for this exercise contains a small C program, which is not a good model of proper use of pointers, but its only designed to help you to concentrate on interpretation of different styles of dereferencing pointers and pointer arithmetic.

What to Do:

Download file `lab3exB.c` from D2L, and draw memory diagrams for point one and point two.

What to Submit:

As part of your in-lab report submit your diagrams.

Exercise C: A Simple Macro:

This is an in-lab exercise

What to Do:

Download the file `lab3exC.c` from D2L. If you try to compile this file you will get an error because the definition of ELEMENTS, which is supposed to be a C macro, is missing.

Your task in this exercise is to write the macro ELEMENTS. It should return the number of elements of an array. For obvious reasons this macro can only work with arrays, not with the pointers.

Submit your source code and your program output.

Exercise D: Duplicating library function, using pointer arithmetic

This is a post-lab exercise

Read This First:

The library header file `<string.h>` contains prototypes for several c library functions, such as `strlen, strcpy, strcat, strcmp, strnca`, etc. For example the cat in `strcat` stands for *concatenation*, which means adding one string to the end of another. Here is a very brief program that uses `strcat`:

```
int main(void)
{
  char s[8];
  printf(" s now contains %s and its length is %d.\n", strcpy(s, "ENCM"), strlen(s));
  printf(" s now contains %s and its length is %d.\n", strcat(s, "339"), strlen(s));

  return 0;
}
```

The output of this program will:

```
s now contains ENCM and its length is 4.
s now contains ENCM339 and its length is 7.
```

In this operation three steps are involved:

- Finding the '\0' character at the end of the destination string (the end of "ENCM" in the example).
- Copying all the characters from the source string ("339" in the example).
- Adding a '\0' character at the end of the modified destination string.

A similar function called `strncat` with the following function prototype:

```
char* strncat(char* dest, const char* source, int n);
```

Appends the first n characters of string source to string dest, and returns a char* to dest. If the length of the C string in source is less than *n*, only the content up to the terminating null character, '\0', is copied.

What to do:

This exercise is about writing your own version of some of the C library functions. In a practical programming project, writing a function that does the same job as a function in the library would be a serious waste of time. On the other hand, it can be a very helpful exercise for a student learning the fundamentals of C.

Important Note: when writing your own copy of library functions:

1. You should only use pointer-notation. It means you are not supposed to use any square brackets, `[]`, in your functions. In other words, your function must only use pointer arithmetic.
2. You are not allowed to call any library functions in your version of functions.

From D2L download the file `lab3exD.c`. Study the file and write down your prediction for the program output. Compile the program and run it to check your prediction. If your prediction was wrong, try to understand why.

Make another copy of `lab3exD.c`; call the copy `my_lab3exD.c`. In `my_lab3exD.c`, add a function definition for `my_strlen` that uses pointer arithmetic (`pointer - pointer`) to calculate the length of a c-string. Then, replace all of the calls to `strlen` in the function main with calls to `my_strlen`, and then make sure your modified program produces appropriate output.

Once `my_strlen` is working, add a function definition for `my_strncat` to `my_lab3exD.c`. *Don't make any function calls within your definition of* `my_strncat`. Replace all of the calls to `strncat` in main with calls to `my_strncat`.

Your last task in this exercise is to add the function prototype, function interface comment, and function definition for a function called `my_strcmp`. This function should work like the `strcmp` C library function. Here is what you need to know about this library function:

```
int strcmp(const char* str1, const char* str2); /* function prototype */
```

`strcmp` compares `str1` and `str2`, and returns 0 if the two strings are identical. Otherwise, returns a positive number if `str1` is greater than `str2`, and a negative number if `str2` is greater than `str1`.

Method one: In some platforms including in our ICT 320 lab the return value of `strcmp` is:
- 1, if str1 is “ADC” and str2 is “ABC”.
- -1, if the str1 is “ABC” and str2 is “ADC”.
- 0, if the str1 is “ABC” and str2 is also “ABC”.

Method two: Some other compiler may operate differently. They may return 0 when `str1` and `str2` are identical. Otherwise, the may returns the ASCII value differences of the first two characters that are different. For examples:
- If str1 is “ADC” and str2 is “ABC”, it returns 2.
- If the str1 is “ABC” and str2 is “ADC”, it returns -2.
- If the str1 is “ABC” and str2 is also “ABC”, it returns 0.

If you try the following code on one of the recent Mac computers:

```
char str1[10] = “ABCDE”;
char str2[10] = “ABCD”;
int y = strcmp (str1, str2);
```

The value of `y` will be 69, because `str1[4] == ‘E’` which its ASCII value is 69 and `str2[4] == 0`.

To come up with a general statement we should say:
`strcmp` returned value indicates the **lexicographical order** of the two strings as follows:
- If return value is < 0 then it indicates str1 is less than str2.
- If return value is > 0 then it indicates str2 is less than str1.
- If return value is == 0 then it indicates str1 is equal to str2.

For the purpose of this exercise, your function `my_strcmp` should use method two.

Don't make any function calls within your definition of `my_strcmp`. Replace all of the calls to `strcmp` in main with calls to `my_strcmp`.

Submit the completed source file and your program output(s) that shows your program work.

Exercise E: Reading Numeric Input as a String

Read This First:

The scanf library function may not be always the best tool to read the user’s numeric input. An alternative method is to read the user's input as a string of characters, and then, after a complete input error-checking, to convert it to a number. Consider a string of digits such as:

```
char string[4] =  "237";
```

'2'	'3'	'7'	'\0'

The above picture shows each digit is stored in one of the elements of the array *`string`*. The ASCII values for each of these characters are:

```
 '2'  == 50
 '3'  == 51
```

```
'7'  == 55
```

The following calculation can be used to convert this string digits to the integer value **237**. Here is the method of calculation

```
int num;
num =  (string[0]-'0')*100 + (string[1]-'0')*10
       + (string[2]-'0');
```

Note: You need to subtract the character code '0', or its ASCII value, 48, from each character in the string to get its actual value (a digit from 0 to 9). (Also, the above only works for three-digit numbers! Something smarter is needed to handle a digit sequence with a length that isn't known in advance.)

Read This Second:

A preferred function to read strings in C is the library function `fgets`.
The following example shows how to use this function.

```
char string [6];
printf("Enter a string: ");
fgets(string, 6, stdin);
```

The above statements reads a string of up to maximum five characters or up to the end-of-line ('\n' which is Enter on most keyboards), whichever comes first.
`stdin` as the third argument indicates that the input comes from the terminal. (A different third argument of type `FILE*` could be used to read a string from a file; more about that later in the course.) For example if user enters: "ABCDEFGH" it only stores the "ABCDE" followed by a '\0'. See the following picture:

'A'	'B'	'C'	'D'	'E'	'\0'

If user's input is "AB", it reads up to and including the '\n' (Enter).

'A'	'B'	'\n'	??	??	??

In this case, you may have to write a couple of lines of code to remove the '\n'. This means to replace it with a '\0', if it is not desired that '\n' be part of the input.

What to Do:

Download files `read_input.c, read_input.h, read_int.c,` and `prog_one.c` from D2L, and take the following steps:

Step one: Compile, the program and run the program. When asked to enter integer numbers, enter valid inputs (positive or negative integers). The program converts your input from a string of digits to an integer and displays the value.

Step two: Now enter several non-integer input such as:

```
12abc
12..9
23.4
.56
```

```
-0.45
```

The program indicates your input is “Invalid”.

Step three: Read the source code and try to understand how the program works.

Step four: Modify the program to be able to read real numbers such as:

```
23.4
.56
-.23
-0.45
-0.0000067
```

(It does not have to handle input in scientific notations, such as 1.23e+45, but it should accept integers as real values.)

Note: the program should still detect invalid (non-numeric) input such as:

```
12abc
12..9
123.4bb
bb123.4
$1235.99
```

To implement this step you should create a file called read_double.c that contains the following functions. (See also the function interface comments in the header file, read_input.h):

```
int read_real(char* digits, int n, double * num);
int is_valid_double(const char* digits);
double convert_to_double(const char *digits);
```

You should also create a file called `prog_two.c`, similar to `prog_one.c` that demonstrates your functions in the `read_double.c` work.

This is an exercise about working with strings, so we are NOT allowing you to call any library functions from your `convert_to_double` function.

Submit your files `prog_two.c, read_double.c`, and your program’s output that shows it works for all acceptable inputs such as

```
 23.4
.56
-.23
-0.45
-0.0000067
564469999
+8773469
+.5
```

and detects any invalid input such as:

- Too many decimal points (for example “`12..999`”).
- Invalid character(s) (for example “`23avb45`”, or `2,347`).
- Invalid space(s) between digits (for example “`+ 234 77`”).