60-140 INTRODUCTION TO ALGORITHMS AND PROGRAMMING I
Dr. Ziad Kobti

Lab 9: Pointer Variables and Pass by Reference

Objectives:
1. Understand pointer variables basics.
2. Learn how to code pass by reference vs. pass by value.

Instructions:
Recall in the previous lab you learned how to create a directory, change into it, and view its contents. (i.e. the UNIX commands you need to understand here are `mkdir` *`directory_name`*, `cd` *`directory_name`*, and `ls`)

In this lab you are asked to create one complete C program that includes several functions, but all inside the same source code file named **Lab9_q.c** in a directory called **lab9** which should be located inside the **cs140** directory you have already created in the first lab. The reason for this is to keep your files organized.

At the end of this lab class, you will have 2 source code files saved in a folder called lab9 under the cs140 folder. Recall that for the purpose of evaluating and grading your work by the GA's, you would need to create one script file which holds the source code, compilation result and the output of all programs. The process of creating such a file is described below:

```
>> script lab9.txt
>>  cat lab9_q1.c     // will show and add the source code for the question to the script file
>>cc lab9_q1.c   // will compile lab9_q1.c  -- repeat for lab9_q2.c
>>./a.out        // will run the program and appends its output to lab9.txt file
>>exit          //closes the script file. Your work is ready to be graded!
```

Submission:
Your lab is graded either at the end of your current lab class (or at the very beginning of your next regular lab class without penalty). Late labs without a valid excuse (eg. illness) receive 0. You are to present your code on the computer in the lab to your lab instructor to receive a grade as follows:

0= not satisfactory or no documentation; 1=incomplete; 2=complete and well documented

Lab questions

When a variable is passed to a function it is passed by-value. Meaning, a copy of this variable is passed to the function. The function modifies the copy and when it terminates the local copy is lost, leaving the original variable from the caller unchanged.
Now if you use call-by-reference, this is where you pass the address of the variable to the function. This way the function will modify the original variable. The catch here is that the function needs to receive the address of the variable (a pointer) and access it using the dereferencing operator.

Part I: Introduction – type the following program and document every line by explaining its functionality and if there are any errors; in which case you would comment that specific line and explain the nature of the error. Write this in a file called lab9_q1.c

```
#include <stdio.h>
int main()
{
      int a = 7;
      int *aPtr;
      aPtr = &a;
      printf("%p", &a);
      printf("%p", aPtr);
      printf("%p", &aPtr);
      printf("%d", a);
      printf("%d", *aPtr);
      printf("%p", *&aPtr);
      printf("%p", &*aPtr);
      printf("%d", *&a);
      printf("%d", &*a);
      return 0;
}
```

Part 2: Parameter Passing by Reference (call the file lab9_q2.c – this program has 3 functions including main)

Write, document and test each of the following function specifications:

1. A function called **Swap** that takes two integer pointers and exchanges the values of each. It returns void. Example: given two integers 'a=2' and 'b=4', after Swap(&a, &b) is called, 'a' will be 4 and 'b' will be 2.

2. A function called **Triple** that takes a single parameter of type pointer to integer and triples its value. It also returns integer (the tripled value). Example: if 'd=10', then after we call Triple(&d) we have 'd=30'.

Make sure you test these functions by calling them from main and printing the results before and after each call.