Simon Fraser University
School of Computing Science

Lab Week 10

(No submission required for this lab.)

Objective

In this week's lab, our objective is to understand and practise recursion.

Notes

1. For this lab, you are free to work on our own or with a partner.
2. This is a long lab. If you do not have enough time to finish these lab exercises within our lab session, no problem! You can finish the lab on our own outside the lab session either
   a. by coming back to CSIL and using a CSIL workstation in a lab room where there is no scheduled lab taking place, or
   b. by using our own computer at home or on campus or elsewhere.
3. Let's make sure we show our work to one of the TA's in the lab to ensure we understand the concepts (see Objectives above) exercised in this lab

---

Lab Prep

Once we have logged into a CSIL Windows workstation, let's access our **U: drive/CMPT120** and create a directory called **Lab7**. Let's store any files we create as part of this lab into this directory.

In all the exercises of this lab, feel free to use the **visualizer** (see its link on the Home page of our course web, week 8) to view the execution flow as it executes our functions.

Exercise 1 – Recursion

1. a) Download the program **Multiplication_by_Recursive_Addition.py** which we find under the Example/Files/Feedback column of the Labs and Tutorials page on our course web site.

   i. Let's figure out how its recursive function works by executing the program and observing the stack frames produced by this execution and the program's output. What are the base cases? What is the recursive case?

   ii. We may also want to have a look at **Multiplication_by_Iterative_Addition.py** and compare and contrast both ways (recursive and iterative) of implementing multiplication.

   b) Download the program **ReverseString.py** (**or ReverseStringV.py)**, which we find under the Example/Files/Feedback column of the Labs and Tutorials page on our course web site.

   i. Let's figure out how its recursive function works by executing the program and observing the stack frames produced by this execution and the program's output. What is the base case? What is the recursive case?

   ii. Write a second function called **ReverseStringIteratively( )** and add it to this program. This function must reverse a string in an iterative fashion, i.e., using a loop as opposed to using recursion. Let's make sure we test it completely. This means that we use a sufficient number of test cases such that all its statements are executed (i.e., tested).

2. Consider the following Python program:

```
def magic(aNumber, aSymbol):
   print(aSymbol * aNumber)
   if aNumber > 1 :
     magic(aNumber-1, aSymbol)
     print(aSymbol * aNumber)
   return

# Main part of the program
```

```
magic( 6, "*")
print("Wow!")
```

a) What is its base case? What is its recursive case?

b) How many times is the function `magic( )` called? In order to answer this question, box trace the execution of this program (i.e., hand trace the execution of this recursive function by drawing boxes, where each box represent the stack frame assigned to each execution of the function – as we have done in class).

3. Modify the above Python program such that its function is as general as possible. To figure how to do this, have a look at the following two Sample Runs, which represent output our function, once modified, will be able to produce. The function must remain recursive:

Sample Run 1:

```
******
*****
****
***
**
*
     <- this is a blank line (a printed line with no characters)
*
**
***
****
*****
******
Wow!
```

Sample Run 2:

```
@@@@@@@@
@@@@@@@
@@@@@@
@@@@@
@@@@
@@@
@@
@@@
@@@@
@@@@@
```

```
@@@@@@
@@@@@@@
@@@@@@@@
Wow!
```

4. Consider the following Python program:

```
def magic(aNumber, aSymbol, end):
 print(aSymbol * aNumber)
 if aNumber <= end :
    magic(aNumber+1, aSymbol, end)
    print(aSymbol * aNumber)
 return

# Main part of the program
magic( 3, "*", 7)
print("Wow!")
```

   a) What is its base case? What is its recursive case? What is the difference between the base case and recursive case of this function `magic()` and the base case and recursive case of the function `magic()` in Problem 2 above?

   b) How many times is the function `magic( )` in the above program called? In order to answer this question, box trace its execution.

   c) What happens when we call `magic()` with the argument `10` instead of `3`.

5. a) Modify the above Python program such that when we execute it, it now produces the following output and its function is still recursive:

```
*
**
***
****
*****
******
*******
********
*********
**********
***********
************
Wow!
```

5. b) Modify the above Python program once again such that when we execute it, it now produces the following output and its function is still recursive:

```
***********
**********
*********
********
*******
******
*****
****
***
**
*
Wow!
```

6. Write a Python program, containing a recursive function, that produces the following output:

```
*!
**!!
***!!!
****!!!!
*****!!!!!
******!!!!!!
*******!!!!!!!
********!!!!!!!!
*********!!!!!!!!!
********!!!!!!!!
*******!!!!!!!
******!!!!!!
*****!!!!!
****!!!!
***!!!
**!!
*!
Wow!
```

Let's make this function as general as we can.

7. What does the following Python function do?

```
def fun(aString):
    if len(aString) == 1 :
```

```
        if aString.isdigit():
            result = int(aString)
        else:
            result = 0
    else:
        if aString[0].isdigit():
            result = int(aString [0]) + fun(aString[1:])
        else:
            result = fun(aString[1:])
    return result
```

In order to answer this question:

a) Box trace the execution of the function when we call it with `aString = "a3c2$"`.

b) Create a Python program by copying and pasting the function above and adding a `# Main part of the program` section to it. This `# Main part of the program` section must contain statements that test the above function by calling it using the following three **test data**:

1. `aString` is assigned the value `"358"`,
2. `aString` is assigned the value `"abc"` and
3. `aString` is assigned the value `"a3c2$"`.

8. Let's write a recursive function that reverses a list. It will be helpful if we first figure out how to define a list recursively.

   Also, let's write the main part of the program from which we will be calling (i.e., testing) our functions.

9. Write a recursive function that counts how many even numbers there are in a list. Feel free to use the recursive function `fun( )` in the above exercise as an inspiration. Also, let's write the main part of the program from which we will be calling (i.e., testing) our function.

10. Do Exercise 6.3 in Section 6.11 - Exercises from our online textbook.

11. Write a recursive function that finds a character in a string. More specifically, our function is to find the first occurrence of a character in a string and if it does find it, it returns True, otherwise, it returns False. It will be helpful if we first figure out how to define a string recursively. Also, let's write the main part of the program from which we will be calling (i.e., testing) our function.

Have fun!

Anne Lavergne – July 2017