Tel-Aviv University, Spring Semester, 2020

C programming - Exercise 2

Functions, Recursion

Instructor: Andrew Moshkin

IMPORTANT NOTE: For all questions, first read the instructions written in here, then also check the instructions that appear in the source code files attached to this document.
There is no need to write the whole code by yourselves, we gave you a little head start by writing those source code files - use them!
All the instructions for submitting the homework can be found on the course's Moodle, and MUST be followed exactly.

GRADING: There are two questions in this exercise, each of them is divided into sections.
Beside each question, you will find the percentage of the question in the final exercise grade.
Beside each section, you will find the percentage of the section in the final question grade.
You are also graded for documenting your code, for naming your variables and functions properly, and for keeping your code clean and readable:
For comments – 5%.
For readability (proper spacing, etc.) – 5%.
Overall, the maximum grade of this exercise is 100.

Questions about the exercise are to be asked on the designated forum, so that everyone can use the answer. Please do not repeat questions – before posting a question, check if an answer is already there.
But as always, before posting on the forum, search the web!

General notes for recursion:

- Remember to think about the base case(s).
- Note that a function can always receive more than one input, even if it is recursive. Those inputs can be of different types.

Throughout the whole exercise, there is no need to use external libraries (other than those that are already included), and it is forbidden!

Notes for submission: (replace 123456789 by your ID number)
You need to upload a single .zip file named: ex2_123456789.zip. This file will include all of the 6 attached .c files, one file for the solution of each section, except for Q1, section C, for which there are two files. All the files must be renamed in the following fashion:
You should add "ex2_" before each filename, and add "_123456789" at the end, before the .c.
For example, upon submission, the file of question 1, section A, should look like this:
ex2_Q1_SecA_123456789.c

Notes for all files: In this exercise, there is no input-checking inside "main", every integer input is considered valid (i.e. will return 0 from "main", since there are no errors), and handling each case must be done inside the functions you implement. You are not asked to handle inputs that will cause errors or undefined scenarios (e.g. characters).
For both questions, beside what was already implemented it the attached files, "main" should only call the function you implemented, and store the result inside the variable "res", nothing else.

(50%) Question 1

In this question, you will learn about the double factorial, and about tail recursion.
NOTE: Throughout the question, for n<0, return 0.
Test case for this question are all the same and can be found across the web. We will only test your solutions for n<20. Think why (try larger numbers and see what happens).

(25%) Section A:
Using the definition of the double factorial:

$$n!! = n * (n-2) * (n-4) * \dots$$

By filling in the needed lines of code in the file Q1_SecA.c, implement a **recursive** C program that will calculate the double factorial of an integer n.
Note, a non-recursive implementation will not get any credit.

(25%) Section B:
There is another recursive relation for the double factorial, which also depends on the factorial, i.e.

$$n!! = f(n!, g(n)!!)$$

where $g(n)$ is a function that returns a smaller number than n. For example,

$$g(n) = \frac{n}{2}, or\ g(n) = n - 10, etc.$$

Find both $f(\cdot)$ and $g(\cdot)$ and implement this recursive relation by completing the code in Q1_SecB.c. Note that you will also need to implement the factorial function – implement it recursively.
Note, a non-recursive implementation will not get any credit. Implementing the same solution as in the previous section will also not get any credit.

Hint: Use the web :)

(50%) Section C:
A tail-recursive function is a recursive function in which the recursive call is the last thing executed by the function.
Most probably, in both sections A, B you've implemented non-tail-recursive functions. Check how you've implemented your recursive functions in the last sections, and repeat the solution using the other way of implementation (i.e. if you solved section a using tail-recursion, and section be without it, in this section you need to implement section a using a non-tail-recursive function, and implement section b using tail-recursion). Note that in section B, you don't have to implement the factorial function in a tail-recursive way.

Complete the missing lines of code in the files:
Q1_SecC_A.c - for the implementation of the code from section A (GRADING: 15% of Q1).
Q1_SecC_B.c - for the implementation of the code from section B (GRADING: 35% of Q1).

You can read about tail-recursion, and why it can be useful, here.

Hint: Start by implementing the regular factorial in a tail-recursive way.

Another hint: Use the web!

Notes for the files Q1_SecB.c and Q1_SecC_B.c: Overall, there should be only two functions (excluding "main") in those files. One function for the (regular) factorial, and one for the double-factorial.

(40%) Question 2

In this question, you will compute the sum of different series. You will also learn a little bit about stack-overflow and what can cause it.
NOTE: Throughout the question, for n<0, return 0.

(35%) Section A:

Using recursion, find the value of the following expression:

$$\sqrt{n+\sqrt{(n-1)+\sqrt{(n-2)+\sqrt{\ldots+\sqrt{1}}}}}$$

Note, a non-recursive implementation will not get any credit.

Test cases:

n = 10:	3.675980
n = 100:	10.509991
n = 1000:	32.126479

(65%) Section B:

Using recursion, find the value of the following expression:

$$\sqrt{1+\sqrt{2+\sqrt{3+\sqrt{\ldots+\sqrt{n}}}}}$$

Note, a non-recursive implementation will not get any credit.

Test cases (this sum is actually converging, and it does so pretty fast!):

n = 3:	1.712265
n = 10:	1.757933
n = 100:	1.757933
n = 1000:	1.757933

IMPORTANT: For both sections, if you put in large enough values of n, you will get an error. This is okay, and we will learn about it in the course. Read about this type of error here.
Now think – would this error appear if we use an iterative approach to calculate the expressions?