UNIVERSITY OF PARMA

Department of Engineering and Architecture
Degree course in Computer, Electronic and Communications Engineering
PRACTICAL TEST IN COMPUTER SCIENCE & PROGRAMMING LABORATORY
April 4th, 2024

Nome: _____ Cognome:_____ Matr:_____ Postazione_____

Write a program using the C language (name the project with your student ID) that behaves as described below. The available time is 120 minutes. At the end of the time, the saved files on **U:** are going to be automatically collected. Additional documents, files... are available in **T:\Bertozzi**, and it is recommended to use WordPad to read text files.

We want to develop a C program to calculate all prime numbers within an interval *[1, n]*.

To do this:

1. Define a dynamically allocated array of unsigned int initially empty.
2. Ask the user for an integer $n > 1$.
3. For all numbers x from *1* to n, invoke the function *ferm(x)* which returns the number of prime divisors of the number passed as an argument. If this function returns *2*, increase the size of the array defined in step #1 and add the number x to that array.
4. Implement the recursive function **unsigned int ferm(unsigned int n)** which performs the following steps:
   a) If n equals 0, return *0*
   b) If n equals 1, return *1*
   c) If n is even, return *1 + ferm(n/2)*
   d) Alternatively, define three integer support variables: a, b, and $b2$.
   a is initialized with the ceiling approximation of the square root of n.
   $b2$ is initialized with $a \times a - n$.
   b is initialized with the integer part of the square root of $b2$.
   Until $b2$ is not a perfect square, increase a and recalculate the values of b and $b2$ as indicated.
   at the end of the loop, if $a + b$ equals n, the function returns 2, otherwise returns *1 + ferm(a + b)*.
5. The program ends by printing how many prime numbers are present in the interval *[1, n]* and their respective values.

The code should be developed following the proposed order. Correction ends at the first point not correctly implemented.

In order to develop point #3, initially define an empty stub for the function ferm() which returns a preset value so that point #3 can be tested.

Take into account the risk of overflow for variables a, b, and especially b2.

Refer to the manual to understand what the function double ceil(double x) does.