Main Project 2020

Instructions

This is an individual project for 25% of the marks available in the SD2 module: Discrete Mathematics. All work is to be submitted as a single zipped folder by 11:55pm on the due date Friday 27th November 2020, using Moodle. The folder name must be your K-number.

Plagiarism shall result in all associated parties receiving 0 marks for the project.

Projects submitted after the above deadline shall be penalised 5 marks (out of the 25 available) for each day (or part of a day) it is late. So if you are 4 days late, the *maximum* mark you can receive is 25 – 20 = 5 marks.

Introduction

Use C# to code programs 1 to 4, as specified on this document, and a main program that presents the user with a menu leading to these programs, plus an exit menu option. Upon selecting menu option 1 through 4, the user should be prompted for input data for that program, the appropriate program should then run with the inputted data, and the corresponding program output displayed. Finally, a prompt should allow the user to return to the menu in the main program.

Requirements

You are required to upload a zipped folder to Moodle, before the project deadline, containing all components of the project:

- ReadMe file (containing instructions on how to *run* and *use* your software).
- All program files.
- All executable files.
- Report file, giving all answers generated by your software to all of the test data, and to any questions posed, for each of the four program modules. No screenshots are required – I shall run each program and check the generated output versus the output stated in your report.

NB If at any stage of the project, you are unsure as to what the customer requires, then ask the customer for clarification.

Program 1: Program for the prime factorization of a natural number.

Write a program ***FactorList* = PrimeFactors (*n*)** that inputs a natural number $n > 1$, and outputs a vector *FactorList* that lists all of the prime factors, from smallest to largest, and with repetition for multiple factors. For example, since the prime factorization of 24 is $2^3 \times 3$, the output of *PrimeFactors*(24) should be the vector [2 2 2 3]. Starting with $k = 2$, and running k through successive integers, the method should check to see whether n has k as a factor. If it does, then k should be added to the *FactorList* output vector. We then replace n with n/k, and continue to check whether k divides into (the new) n until it no longer does, and then we move on to update k to $k + 1$. With this scheme, only prime factors will be found. Also, we may stop as soon as k reaches the current value of floor($\sqrt{n}$).

i. Run your program with each of the following input values: n = 30, 31, 487, 8893, 987654323, 131317171919.

Program 2: Program for the extended Euclidean algorithm.

Write a program with syntax ***outVec* = ExtEucAlg (*a, b*)** that takes two inputs: a and b, which are positive integers such that $a \geq b$. The output will consist of a length 3 vector *outVec* that has three integer components, d, x, and y, where $d = \gcd(a, b)$, and x and y satisfy $d = ax + by$. The program should follow the extended Euclidean algorithm, given in the Appendix.

i. For each pair of integers a and b that are listed here, use your program to compute $\gcd(a, b)$, and two integers x, and y such that $d = ax + by$.
(a) a = 8,359, b = 4,962; (b) a = 95,243, b = 24,138.
ii. Use your program to solve the equation $88243x + 16947y = 1$, for integers x and y (or to determine that such a solution does not exist).

Program 3: Program for the RSA encryption.

Write a program with syntax ***C* = RSAEncrypt (*P, e, n*)** that will perform the RSA encryption given the inputs: P = the plaintext (an integer mod n), n = the RSA modulus, and e = the (public) encryption exponent. The output, C, is a mod n integer representing the ciphertext.

i. Run your program with the plaintext message $P = 44$, $n = 1{,}517$, and $e = 49$ and confirm the results.
ii. Suppose that Bob adopts the RSA cryptosystem with primes $p = 153{,}817$ and $q = 1{,}542{,}689$, and public key encryption exponent $e = 202,404{,}606$. If Alice uses this system to send Bob the plaintext message $P = 888{,}999{,}000$, apply your program to determine the ciphertext.

Program 4: Program for the RSA decryption.

Write a program with syntax ***P* = RSADecrypt (*C, d, n*)** that will perform the RSA decryption given the inputs: C = the ciphertext (an integer mod n), n = the RSA modulus, and d = the (private) decryption exponent. The output, P, is a mod n integer representing the plaintext.

i. Run your program with the ciphertext message $C = 1069$, $n = 1{,}517$, and $d = 529$ and confirm the results.
ii. For the RSA system described in part (ii) of the previous program, compute Bob's corresponding decryption exponent d. Apply your program to decrypt Alice's ciphertext computed in part (ii) of the previous program.
iii. Explain why the RSA algorithm does not *appear to work* in part (ii), i.e. in decrypting Alice's ciphertext.

Appendix

The Extended Euclidean Algorithm

Input: A pair of positive integers, a and b, with $a \geq b$.
Output: Three integers $d = \gcd(a, b)$, x, and y, that satisfy the equation $d = ax + by$.

Step 1: SET $U = [a, 1, 0]$, $V = [b, 0, 1]$ (Initialize recordkeeping vectors)

Step 2: WHILE $V(1) > 0$ (Tasks below will be repeated while first component of V is positive)

$W = U - \text{floor}(U(1)/V(1))V$
UPDATE $U = V$
UPDATE $V = W$
END WHILE

Step 3: OUTPUT $d = U(1)$, $x = U(2)$, $y = U(3)$.