Selection Logic, Loops, and Arrays

Assigned: Monday, September 21
Due: Thursday October 1, before midnight
Value: 20 points (for successfully submitting a functionally correct program to your Assembla repo before the deadline). Late submissions will receive a 0.

Executive Summary

You will develop functions for manipulating the Gaussian primes.

Purpose

- Learn how to use nested loops, conditionals and arrays in a sophisticated setting
- Learn to implement an API
- Sharpen your problem solving skills and general programming skills

Deliverables

The `assignment-3.c` program source file implementing the functionality described below.

You will receive two c files (`main.c` and `assignment-3.c`) for this lab. You are to implement the Application Programming Interface (API) defined in `assignment-3.c`.

`main.c` is for testing your design. You can make changes on test cases to make sure that your program works correctly for each possible input. We will test your program with a different `main.c` file, which will include different test cases.

Warning: You are not allowed to change the API defined in `assignment-3.c`. That is to say you cannot change the function names, the arguments passed to the functions, or the return type of the function.

Introduction

The *Gaussian integers* are complex numbers of the form $a + bi$, where a and b are integers. The numbers $1, -1, i, -i$ are known as the *units*. A Gaussian integer (a+bi) is called a *Gaussian prime* if and only if (a+bi) = (p+qi)(r+si) implies that (p+qi) or (r+si) is a unit. The convention is that units are NOT primes, and this is the convention we've followed throughout the document.

If a real integer x is not a prime, e.g., $x = y \times z$ where neither *y* nor *z* are 1 or -1, then it is not a Gaussian prime. (For example, 6, 15, 289 are not Gaussian primes.)

What makes the Gaussian primes interesting is that the converse may not be true: for example *2* cannot be factored over the real integers, but $2 = (1 + i)(1 - i)$, neither of which is a unit so *2* is not a Gaussian prime. The integer 3 is a Gaussian prime, for reasons that will be apparent later.

The API you are to implement consists of 3 functions:

1. Take a Gaussian integer *z* as an argument, and determine if it is a Gaussian prime. If it is not a Gaussian prime, factor it into Gaussian integers. Write the result to stdout.
2. Take a magnitude *M* and compute all Gaussian primes *a* + *bi* such that 0 <= a <= M and 0<= b <= M. Write the primes to stdout.
3. Take a magnitude *M* and draw a plot showing all the Gaussian primes that would be computed in Part 2. The plot should be in the form of text that you print to stdout.

Requirements

Setup

1. Create a Project named *Lab3* in Visual Studio for this lab.
2. From the Solution Explorer, add the file `assignment-3.c` to your *Lab3* project, so your file structure will be `<repo>/Lab3/assignment-3.c`

Input format

Inputs for your functions will come from tests cases written in `main.c` file. You can assume that input format will always be correct. That is, you don't need to check the validity of input in your functions.

Output Format

1. For Mode 1, your output should be for the form:

    -1 + 3i is not a Gaussian prime. Factorization = (1+2i)(1+1i)

    1 + 4i is a Gaussian prime.

    Warning: Your output should be exactly in the form described above. Otherwise, you may lose points. Print **only one** factorization of the input if it is not a Gaussian Prime.

2. For Mode 2, your output should be of the form:

    Primes with real and imaginary parts having magnitude less than or equal to 4 = {0 + 3i, 1 + 1i, 1 + 2i, 1 + 4i, 2 + 1i, 2 + 3i, 3 + 0i, 3 + 2i, 4 + 1i}

    Warning: Your output should be exactly in the form described above. Otherwise, you may lose points. Pay attention to the order of sequence (ordered first by real component and then by imaginary component) and the commas! You should not put a comma at the end of the list. You will need to find a way to handle it. Print Gaussian primes only in the **positive-positive** quadrant.

3. For Mode 3, your output should be text, e.g., if M = 4

    For M = 4;

```
4 0X000
3 X0X00
2 0X0X0
1 0XX0X
0 000X0
  01234
```

    For $z = a + bi$, y-axis represents b values, and x-axis represents a values.
    Warning: Your output should be exactly in the form described above. Otherwise, you may lose points. Plotting Gaussian primes in the **positive-positive** quadrant is enough for correct result.

Design and implementation constraints

1. For Mode 1: you will need to write nested loops to use brute force in order to find if the input is a Gaussian prime or not. If not, you should print one of the factorization of the input.

To find that the input is not a Gaussian prime, you need to find 2 non-unit complex numbers where multiplication of these two complex numbers should be equal to your input (check the definition described above for more detail about Gaussian primes). That is, for input z where $z = (x + yi)$, you need to iterate over variables a, b, c, and d to find values that satisfy $(x + yi) = (a + bi)(c + di)$. Learn more in the FAQ.

2. For Mode 2: you should iterate over [0,M] for both *a* value and *b* values where $z = a + bi$. For each *a*, *b* values, you need to check if (*a*+*bi*) is a Gaussian prime or not. If so, you should print the result. For better performance, you may want to use a 2D array. You can iterate over cell indexes, and put 1 if it is a Gaussian prime, 0 if it is not. Then, you can print your 2D array as described in previous section.

3. For Mode 3; you should do the same thing as you do for Mode 2. However, this time you will plot your result as described in previous section. If it is a Gaussian prime, you should print X, otherwise 0 (zero). Your output should be exactly in the form described above.

Grading

Your score will be solely a function of your program's functionality. In particular, efficiency is not a consideration.

Lab Specific FAQ

- What part of the text is this based on?

  Chapters 5, 6, 7, and some elements of *printf* and *scanf*

- Where can I learn more about the Gaussian primes?

  The Wikipedia page has a wealth of information about the Gaussian primes.

- How do I multiply and divide Gaussian primes?

  Multiplication is straightforward $(a + bi)(c + di) = (ac - bd) + (ad + bc)i$. Division is more complicated, and you do not need to know how to perform division for this lab. The quotient *q* and remainder *r* of $(a + bi)/(c + di)$ are

defined by $(a + bi) = (c + di)q + r$ with *r* having the minimum possible modulus satisfying this equation.

The fact that this uniquely defines division is not completely obvious. There is also an algorithm for performing division described here (but you do not need to perform division to for this lab!).

- When finding whether a complex number is a Gaussian prime, what range of variables should I iterate over?

  You are trying to find values of a, b, c, d that satisfy $(x + iy) = (a + bi)(c + di)$.

  The absolute values of these four integers can be proven to always be less than or equal to the norm of the complex number z, which is $|z| = \sqrt{x^2 + y^2}$.

  Since you cannot use the math library in this lab, a sufficient range to consider would be $|a|, |b|, |c|, |d|$ all $\leq (x^2 + y^2)$. Thus, you need to brute force over values in the range $-(x^2 + y^2)$ to $+(x^2 + y^2)$

- Is there any faster algorithm to solve that problem?

  Yes, you can use sieving approach. It is not required for that lab. However, if you want to try, have fun :)

  Sieving: One approach to computing all the (conventional) primes from 2 to N is to sieve, that is start with all numbers from 2 to N in the set S of candidate primes, and set the set of primes P to the empty set. Now iteratively add the smallest number x in S to the set P of primes, and remove all multiples of x from S.

  The same approach works for computing Gaussian primes. The key, for any candidate Gaussian number, is to enumerate all Gaussian integers up to its modulus :for $z = x + iy$, the modulus of *z* is $\sqrt{(x \times x + y \times y)}$ and is denoted by |*z*|.

  It is straightforward to see that |z1 ×z2| = |z1| ×|z2|, so to see if *z* is a prime, we only need to consider numbers whose modulus is less than that of *z.*

- Can we use math library or any other libraries for this Lab?

No. You cannot. We try to evaluate your ability to write program, not your ability to use existing libraries.

Important Notes

1. To make expected output format clear, please check the following outputs examples;

   1 - 1i ⇒ wrong ⇒ 1 + -1i ⇒ correct
   -1 - 1i ⇒ wrong ⇒ -1 + -1i ⇒ correct
   2i ⇒ wrong ⇒ 0 + 2i ⇒ correct
   i ⇒ wrong ⇒ 0 + 1i ⇒ correct
   1 ⇒ wrong ⇒ 1 + 0i ⇒ correct
   1 + 1i ⇒ correct
   -1 + 1i ⇒ correct
   0 + 0i ⇒ correct