EE 5103 - Assignment 3

==========

Total points: 10

Due: Feb 24 by 11:59 PM

Submission instructions:

Name your source files as <LastNameFirstName-q1>.cpp , <LastNameFirstName-q2>.cpp, etc. (e.g. KrishnanRam-q1.cpp). Place them in a folder called <LastNameFirstName> and zip the folder to get <LastNameFirstName>.zip. Upload the file to blackboard.

==========

Q1 (5 points)

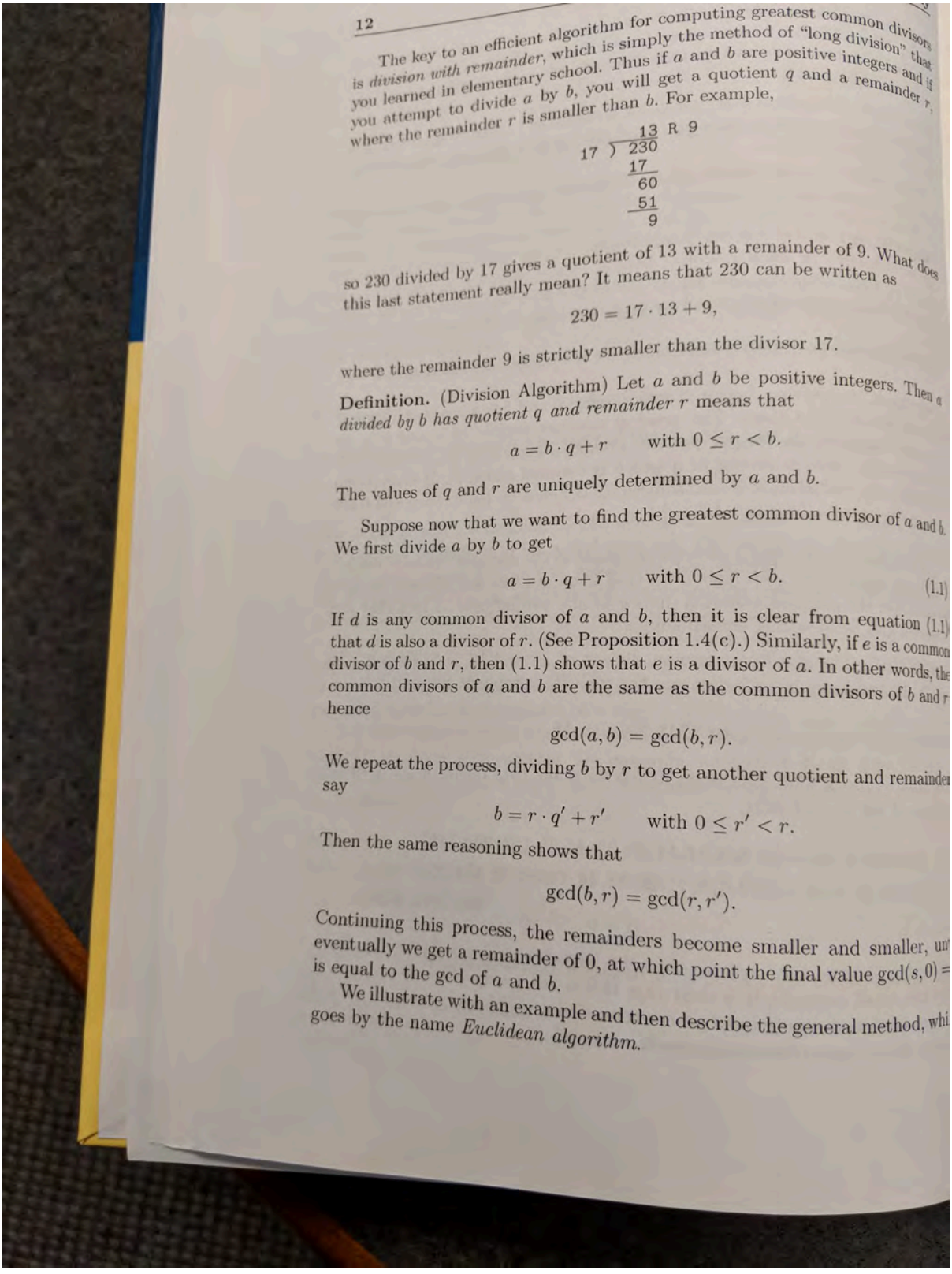
Write a program that implements the Euclidean Algorithm for finding the greatest common divisor of two integers a and b. You can read more about it online if you would like or use other sources. I happen to have a math textbook that has a nice explanation of this algorithm. Please see the screenshots in the next two pages for how the algorithm works. The algorithm is stated in steps 1-5 under Theorem 1.7.

Q2 (5 points)

Consider the scenario where the user wishes to input positive numbers in the form of columns. However, we do not know ahead of time (a) how many columns of inputs that the user wishes to input and (b) within each column, how many positive numbers that the user wishes to input. After the user is done inputting all the columns (indicated by a negative number as the first input of a column), display all the numbers in the form of a two dimensional matrix. Also, support displaying only the numbers from a particular column of the user's choosing.

We discussed dynamic single dimensional arrays on Feb 13. First, read about two-dimensional arrays. Then read about how to handle dynamic memory in the context of two-dimensional arrays. Here is a helpful link: https://www.techiedelight.com/dynamic-memory-allocation-in-c-for-2d-3d-array/. I have copy-pasted the figure and the code at the bottom for your reference. Now, implement a "dynamic" two-dimensional array following the idea discussed on Feb 13. In the link above, you **MUST** use the "Using Array of Pointers" approach under the discussion on "2-Dimensional Array".

Reference for Q1:



The key to an efficient algorithm for computing greatest common divisors is *division with remainder*, which is simply the method of "long division" that you learned in elementary school. Thus if a and b are positive integers and if you attempt to divide a by b, you will get a quotient q and a remainder r, where the remainder r is smaller than b. For example,

$$\begin{array}{r} 13 \text{ R } 9 \\ 17 \overline{)\,230} \\ \underline{17} \\ 60 \\ \underline{51} \\ 9 \end{array}$$

so 230 divided by 17 gives a quotient of 13 with a remainder of 9. What does this last statement really mean? It means that 230 can be written as

$$230 = 17 \cdot 13 + 9,$$

where the remainder 9 is strictly smaller than the divisor 17.

Definition. (Division Algorithm) Let a and b be positive integers. Then a *divided by* b *has quotient* q *and remainder* r means that

$$a = b \cdot q + r \qquad \text{with } 0 \le r < b.$$

The values of q and r are uniquely determined by a and b.

Suppose now that we want to find the greatest common divisor of a and b. We first divide a by b to get

$$a = b \cdot q + r \qquad \text{with } 0 \le r < b. \tag{1.1}$$

If d is any common divisor of a and b, then it is clear from equation (1.1) that d is also a divisor of r. (See Proposition 1.4(c).) Similarly, if e is a common divisor of b and r, then (1.1) shows that e is a divisor of a. In other words, the common divisors of a and b are the same as the common divisors of b and r, hence

$$\gcd(a, b) = \gcd(b, r).$$

We repeat the process, dividing b by r to get another quotient and remainder, say

$$b = r \cdot q' + r' \qquad \text{with } 0 \le r' < r.$$

Then the same reasoning shows that

$$\gcd(b, r) = \gcd(r, r').$$

Continuing this process, the remainders become smaller and smaller, until eventually we get a remainder of 0, at which point the final value $\gcd(s, 0) = s$ is equal to the gcd of a and b.

We illustrate with an example and then describe the general method, which goes by the name *Euclidean algorithm*.

Example 1.6. We compute $\gcd(2024, 748)$ using the Euclidean algorithm, which is nothing more than repeated division with remainder. Notice how the quotient and remainder on each line become the new a and b on the subsequent line:

$$\begin{aligned} 2024 &= 748 \cdot 2 + 528 \\ 748 &= 528 \cdot 1 + 220 \\ 528 &= 220 \cdot 2 + 88 \\ 220 &= 88 \cdot 2 + 44 \quad \leftarrow \boxed{\gcd = 44} \\ 88 &= 44 \cdot 2 + 0 \end{aligned}$$

Theorem 1.7 (The Euclidean Algorithm). *Let a and b be positive integers with $a \geq b$. The following algorithm computes $\gcd(a, b)$ in a finite number of steps.*

(1) *Let $r_0 = a$ and $r_1 = b$.*

(2) *Set $i = 1$.*

(3) *Divide r_{i-1} by r_i to get a quotient q_i and remainder r_{i+1},*

$$r_{i-1} = r_i \cdot q_i + r_{i+1} \qquad \text{with} \quad 0 \leq r_{i+1} < r_i.$$

(4) *If the remainder $r_{i+1} = 0$, then $r_i = \gcd(a, b)$ and the algorithm terminates.*

(5) *Otherwise, $r_{i+1} > 0$, so set $i = i + 1$ and go to Step 3.*

The division step (Step 3) is executed at most

$$2\log_2(b) + 1 \quad \textit{times.}$$

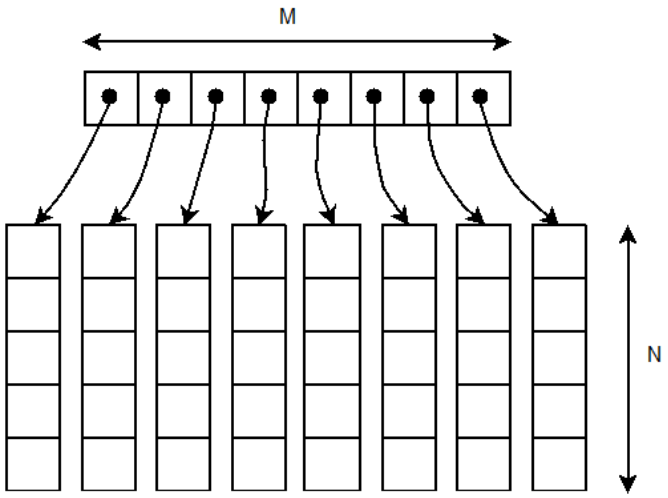
Proof. The Euclidean algorithm consists of a sequence of divisions with remainder as illustrated in Figure 1.2 (remember that we set $r_0 = a$ and $r_1 = b$).

$$\boxed{\begin{array}{rcl} a = b \cdot q_1 + r_2 & & \text{with } 0 \leq r_2 < b, \\ b = r_2 \cdot q_2 + r_3 & & \text{with } 0 \leq r_3 < r_2, \\ r_2 = r_3 \cdot q_3 + r_4 & & \text{with } 0 \leq r_4 < r_3, \\ r_3 = r_4 \cdot q_4 + r_5 & & \text{with } 0 \leq r_5 < r_4, \\ \vdots & \vdots & \vdots \\ r_{t-2} = r_{t-1} \cdot q_{t-1} + r_t & & \text{with } 0 \leq r_t < r_{t-1}, \\ r_{t-1} = r_t \cdot q_t & & \end{array} \quad \text{Then } r_t = \gcd(a, b).}$$

Figure 1.2: The Euclidean algorithm step by step

The r_i values are strictly decreasing, and as soon as they reach zero the algorithm terminates, which proves that the algorithm does finish in a finite

Reference for Q2:




```
#include <iostream>
// M x N matrix
#define M 4
#define N 5
// Dynamic Memory Allocation in C++ for 2D Array
int main(){
	// dynamically create array of pointers of size M
	int** A = new int*[M];
	// dynamically allocate memory of size N for each row
	for (int i = 0; i < M; i++)
		A[i] = new int[N];
	// assign values to allocated memory
	for (int i = 0; i < M; i++)
		for (int j = 0; j < N; j++)
			A[i][j] = rand() % 100;
	// print the 2D array
	for (int i = 0; i < M; i++){
		for (int j = 0; j < N; j++)
			std::cout << A[i][j] << " ";
		std::cout << std::endl;
	}
	// deallocate memory using delete[] operator
	for (int i = 0; i < M; i++)
		delete[] A[i];

	delete[] A;

	return 0;
}
```