

OBJECTIVES : Review Questions

Instructors : Serpil TIN

Assistants : Berk ÖNDER, Efe M. ŞAHİNKOÇ, Hatice Zehra YILMAZ

Q1. Two positive integers are considered to be *relatively prime* if there exists no integer greater than 1 that divides them both.

Write a C program that reads the pairs of numbers from an input file named “**nums1.txt**” or “**nums2.txt**” and stores them in two parallel arrays, checks whether the pairs are relatively prime or not, and displays the ones that are relatively prime on the screen. If there are no relatively prime numbers in the arrays, display an appropriate message.

Write the following functions;

- **readFromFile:** takes the input file pointer and two integer arrays as parameters, reads the pairs of numbers from the file, and stores them in two parallel arrays. The function returns the actual number of elements in the arrays.
- **isRelPrime:** takes two numbers as its parameters, and returns 1 if the given numbers are relatively prime; otherwise returns 0.
- **findRelPrimes:** takes two parallel arrays and their size as input parameters, finds the relatively prime numbers, and stores their indexes in another integer array. The function returns the new array and the number of relatively primes.

Project Name: LG19_Q1

File Name: Q1.cpp

Example Run(using nums1.txt) :

There are 3 relatively prime numbers in the arrays

```

7      17
13     7
15     8
    
```

nums1.txt

```

9 3
7 17
24 15
13 7
8 18
15 8
    
```

Example Run(using nums2.txt) :

There are no relatively prime numbers in the arrays!

nums2.txt

```

8 18
25 35
14 21
45 9
24 66
    
```

Q2. Write a C program that takes the number of steps in the range [2-10] and the step length in the range [3-10] from the user, and draws stairs on the console. Write the following function;

- **drawStairs** that takes the number of steps and the step length as parameters and draws stairs on the console.

Project Name: LG19_Q2

File Name: Q2.cpp

Example Run#1:

```

Enter the number of steps (2 - 10): -1
Enter the number of steps (2 - 10): 4
Enter the step length: (3 - 10): 5
    
```

```

*****
 *
*****
 *
*****
 *
*****
 *
*****
 *
    
```

Example Run#2:

```

Enter the number of steps (2 - 10): 2
Enter the step length (3 - 10): 7
    
```

```

*****
 *
*****
 *
    
```

Q3. Write a C program that reads ordered pairs of a relation on the set {1, 2, 3, 4} from each line of a text file named "relation.txt". The program decides whether the relation is symmetric, reflexive, and/or not-reflexive.

- A relation R on a set A is
- Reflexive; if every element in the major diagonal is 1.
- Not-Reflexive; if every element in the major diagonal isn't 1.
- Symmetric; to find out whether the matrix is symmetric or not, compare the lower triangle that is below the major diagonal to the upper triangle that is above the major diagonal. If any of these comparisons is unequal, then the matrix is not symmetric.

Write the following functions;

- **check_dia** that checks whether all major diagonal elements of a square matrix have a certain value.
- **isSymmetric** that decides whether a square matrix is symmetric or not.
- **display** that takes the square matrix

Hint: Set 1 to the given positions and 0 for the others.

Project Name: LG19_Q3

File Name: Q3.cpp

relation.txt

```
1 1
1 2
1 4
2 2
2 3
3 1
3 3
4 2
4 3
4 4
```

Example Run:

Matrix of the Relation

```
1 1 0 1
0 1 1 0
1 0 1 0
0 1 1 1
```

Relation is reflexive

Relation is NOT symmetric

Additional Question

A row is called "row-dominant" when the **sum** of its elements is strictly larger than the sum of the elements in any column of the matrix.

Write a **modular** C program that reads and validates the dimension **n** for a matrix ($1 \leq n \leq 10$) and fills the $n \times n$ matrix with the values given by the user. The program computes the sum of each row and the sum of each column. Then finds and displays all row indexes that are row-dominant, together with their row sums. If no row is row-dominant, displays an appropriate message.

Try to write the following functions;

- **findRowSums**: computes the sum of each row.
- **findColSums**: computes the sum of each column.
- **display**: display the content of the given one-dimensional array
- **rowDominants**: Finds and displays all row indexes that are row-dominant, together with their row sums.

Project Name: LG19_AQ

File Name: AQ.cpp

Example Run#1:

```
Enter n: 15
Invalid value for n.

Enter n: 0
Invalid value for n.

Enter n: 3
Enter the elements of the 3x3 matrix:
1 2 3
4 5 6
7 8 9

Row sums   : [ 6 15 24]
Column sums: [ 12 15 18]

Row 3 is row-dominant (sum = 24)
```

Example Run#2:

```
Enter n: 5
Enter the elements of the 5x5 matrix:
10 32 20 30 48
69 58 41 23 74
36 21 45 66 23
25 74 63 96 45
10 20 58 47 36

Row sums   : [140 265 191 303 171]
Column sums: [150 205 227 262 226]

Row 2 is row-dominant (sum = 265)
Row 4 is row-dominant (sum = 303)
```