ELEMENTS OF PROGRAMMING (CSC 1100), Sem 1 17/18

ASSIGNMENT 2: PAIR PROGRAMMING (Section 4)
DUE DATE: MON, 4 DEC 2017 (11.55 PM)

QUESTION 1 (COMMON PRIME FACTORS)

A **prime number** is a number that can be divided evenly **only** by 1 or itself and it must be a whole number greater than 1. For example :

The prime numbers between 2 to 12 = { 2, 3, 5, 7, 11 }

The prime numbers between 5 to 20 = { 5, 7, 11, 13, 17, 19 }

Prime Factorization is finding which prime numbers multiply together to make the original number. As an example :

Prime factors for 12 = 2 x 2 x 3 = { 2, 2, 3 } : 2 and 3 are prime numbers
Prim factors for 20 = 2 x 2 x 5 = { 2, 2, 5 } : 2 and 5 are prime numbers

Thus, the common prime factors for 12 and 20 is { **2** } as shown in Figure 1.

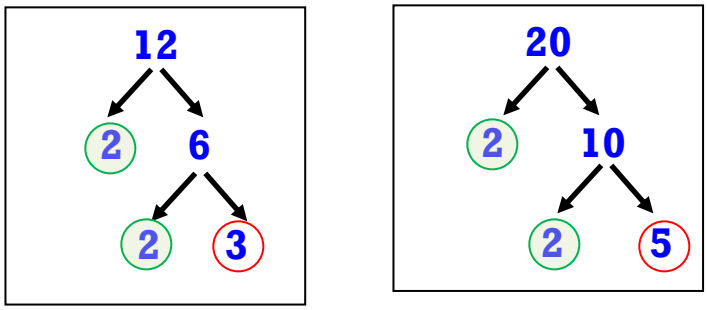

Figure 1: Common Prime Factors of 12 and 20 (in green)

Write a C++ program that creates a list of prime numbers for 2 input values and return the **highest** Common Prime Factor (CPF) for the two values.

a) In your `main()` function, define two empty single dimensional arrays of type integer. Create your arrays using `typedef`. Request the user to enter 2 integer values. Use the user inputs as the size of the array (define typedef only after user input).

b) Create a function named `generate_primes()` that auto-generates the prime factors (starts from the value 2 up to the input itself) for each user input in (a) in increasing order using repetition. Generate the prime factors only if the number can be divided by 1 or itself.

c) Create a function named `commonPrimes()` that compares each of the element in the first and second array generated in (b). Find the common prime factors of the two inputs and return the **highest** CPF of the two values (*Hint: You need a nested loop to compare the 2 arrays and take the last common prime factor as the highest common prime factor).

Sample Output :

```
Enter two integer values separated by a space: 15 24

Prime factors of 15:
3 5

Prime factors of 24:
2 2 2 3

Highest Common Prime Factor for 15 and 24 is 3
```

QUESTION 2 (MATRIX MANIPULATION)

A matrix is an array of numbers of size m by n (i.e., `m x n`). A **square matrix** is a matrix that has the same number of rows and columns. For example, Figure 1 shows a matrix of 2 by 2 with 2 rows and 2 columns:

$$\begin{bmatrix} 14 & 8 \\ 3 & -6 \end{bmatrix}$$

Figure 1: Matrix of size 2 by 2 (square matrix)

The **determinant** of a matrix represented by two vertical lines on either side **|A|** is a special number that can be calculated from a square matrix. Figure 2 shows an example of how a **determinant** for matrix A and matrix B are calculated :

$$\text{Matrix A} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$|A| = ad - bc$$

$$\text{Matrix B} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

$$|B| = a(ei - hf) - b(di - gf) + c(dh - ge)$$

Figure 2: Determinant of Matrix 2 x 2 and 3 x 3

A transpose of a matrix is a matrix that swaps its rows and columns. An **example** of a transposed 2 by 2 matrix A^T is shown below and the elements highlighted in green is its **diagonal** values:

$$\begin{bmatrix} p & q \\ r & s \end{bmatrix}^T = \begin{bmatrix} p & r \\ q & s \end{bmatrix}$$

Figure 3: Transposition and diagonal values of Matrix 2 by 2

Write a C++ program that returns the determinant of a 3 x 3 matrix, transpose the matrix and extract the diagonal values of the transposed matrix. Your program shall include the following functions:

a) Define **a 2 dimensional array, X** that represents a **square matrix** of size **3 x 3**. Request the user to enter the values for the array using **advanced pointer notations** (refer to your slides for the list of array pointer notations).

b) Create a function named **`findDeterm()`** that passes as argument, the values of array ***X*** and return its determinant from the function. Use **advanced pointer notations** to perform your calculation. Display your result in **`main()`**.

c) Create a second function named **`transposeMat()`** that passes as arguments, the value of **array X**. Transpose the array as X^T and store the results in a new 2d array (2 x 2) named ***XT***. Use **advanced pointer notations** to find the transpose. Display your transposed matrix.

d) Create a third function named **`xtractDiag()`** that passes as argument, the values of array ***XT*** and extract its diagonal values and store it in a 1 dimensional array named **DX**. Use **advanced pointer notations** to find and store the diagonal values. Display your 1d array containing the diagonal values.

The basic program and function stubs are given below as a guideline.

```
#include <iostream>
using namespace std;

//declare rows and columns sizes
//declare all empty arrays here as global

int findDeterm();//complete your prototype
void transposeMat();//complete your prototype
void xtractDiag(); //complete your prototype
```

```
int main()
{
   //get user input for array value using nested loop
   //call findDeterm(): pass 2d array
   //display determinant
   //call transposeMat() : pass 2d array
   //call xtractDiag() : pass transposed array
}

int findDeterm()//pass 2d array
{
   //more codes here
   return determinant;
}

void transposeMat()//pass 2d array
{
   //create if-else condition in nested loop to determine index position
    //find transpose
    //display array
}

void xtractDiag()//pass transposed array
{
   //create if-else condition in nested loop to determine index position
   //extract and store diagonal values
   //display array
}
```

Sample Output :

```
Enter values for array X:

6  1  1
4 -2  5
2  8  7

|X| or the determinant of X is : -306

XT or the transposed value of matrix X is:

6  4  2
1 -2  8
1  5  7

The values of matrix DT (diagonal values) is:

| 6 -2 7 |
```