PIC 10A Section 2 - Homework # 5 (due Monday, February 7, by 11:59 pm)

You should upload each .cpp (and .h file) separately and submit them to CCLE before the due date/time! Your work will otherwise not be considered for grading. Do not submit a zipped up folder or any other type of file besides .h and .cpp.

Be sure you upload files with the precise name that you used in your editor environment otherwise there may be linker and other errors when your homeworks are compiled on a different machine. Also be sure your code compiles and runs on Visual Studio 2022.

COMPUTING THE PRIME FACTORIZATION OF INTEGERS

The aim with this homework is to practice nested loops and ifs. You will have the pleasure of writing a program that factors positive integers or determines if they are prime. Lots of people really like prime numbers; personally, I'm kind of indifferent to them...

In the end, you should submit a single file **prime.cpp**.

If you want to use **std::vector**s for this, go ahead. But they are not necessary for this homework.

Terminology:

We say that the positive integer b is a *divisor (or factor)* of the positive integer a if $\frac{a}{b}$ is an integer and there is no remainder. For example, 7 is a divisor of 28 because $28/7 = 4$. But 7 is not a divisor of 12, because $12/7$ is not an integer.

A *prime number* is a positive integer that has no divisors except for 1 and itself (a number like 5); a number is *composite* if it has divisors other than 1 and itself (like 9 = 3x3, 22 = 2 x 11, etc.).

The number 1 is not prime, nor is it composite, it is the "unit". The number 2 is prime, and it is the only even prime number. After 2, all prime numbers are odd because all even numbers bigger than 2 have 2 as a divisor.

The Fundamental Theorem of Arithmetic states that that all positive integers can be written uniquely as a product of prime numbers or they are prime themselves. For some examples, **12 = 2 x 2 x 3** is a product of prime numbers; **7** is prime; **23** is prime; **72 = 2 x 2 x 2 x 3 x 3**; and **700 = 2 x 2 x 5 x 5 x 7**...

The task:

You will need to write a program that will ask a user for a lower and upper bound of (positive) integers to factor. The program **must be able to handle integer inputs as large as 10 billion!!!**. The program will then...

- write out all the prime factors with x's between them if the number is composite.
- print "prime!" if the number is prime.
- print "unit!" if the number is 1.

Write a program that will prompt a user for a two integers, a lower and upper bound. It will then print out all the numbers in that range (inclusive) giving the factorization or stating the number is prime (or a unit).

Do not forget to comment your code and practice good coding and choose appropriate variable types, etc. The program should execute in the following manner. You must make sure the output matches the format below perfectly. **Be sure to read the hints!**

Enter a range of numbers to factorize. Enter lower bound and upper bound separated by a space: [LOWER NUMBER ENTERED] [UPPER NUMBER ENTERED]
[EACH NUMBER]: [FACTORIZATION WITH x's BETWEEN FACTORS OR "prime!" MESSAGE IF PRIME OR "unit!" IF THE NUMBER IS 1.

```
Enter a range of numbers to factorize. Enter lower bound and upper bound separated by a space: 33 57
33: 3 x 11
34: 2 x 17
35: 5 x 7
36: 2 x 2 x 3 x 3
37: prime!
38: 2 x 19
39: 3 x 13
40: 2 x 2 x 2 x 5
41: prime!
42: 2 x 3 x 7
43: prime!
44: 2 x 2 x 11
45: 3 x 3 x 5
46: 2 x 23
47: prime!
48: 2 x 2 x 2 x 2 x 3
49: 7 x 7
50: 2 x 5 x 5
51: 3 x 17
52: 2 x 2 x 13
53: prime!
54: 2 x 3 x 3 x 3
55: 5 x 11
56: 2 x 2 x 2 x 7
57: 3 x 19
```

```
Enter a range of numbers to factorize. Enter lower bound and upper bound separat
ed by a space: 5000000029 5000000033
5000000029: prime!
5000000030: 2 x 5 x 500000003
5000000031: 3 x 3 x 13 x 13 x 3287311
5000000032: 2 x 2 x 2 x 2 x 2 x 37 x 4222973
5000000033: 7 x 7 x 17 x 127 x 151 x 313
```

Hints:

1. Hint 6 provies you with nearly all the implementation details in pseudo-code. Be sure to read #6 and all of these hints!

2. You are dealing with some very long numbers here... pick a data type that will not result in overflow errors.

3. The example given that factors numbers from 5000000029 to 5000000033 takes a while - a minute or two. Don't expect those computations to be lightning fast.

4. Take this in stages: can your program even list the prime factors of a single number? Until you can do that, don't even concern yourself with having factors listed multiple times and managing the multiplication signs, etc.

5. Do not panic: this is not a very long assignment if you understand nested loops and

conditions. Not counting comments and adding white space for readability, it can all be done in about 30 lines of code!

6. The pseudo-code giving the implementation is below:

   - Set **lower** and **upper**, the lower and upper bounds of numbers to factorize.
   - For each **i** from **lower** to **upper**...
     - Set **current** to **i** and start with a divisor of **divisor = 2**. Until **divisor** exceeds **i/2** or **current** equals **1**...
       * If **divisor** divides evenly into **current**, set **current** = **current**/**divisor** and print **divisor** with an 'x' (or without an 'x' if **current** is 1).
       * Otherwise, replace **divisor** by the smallest odd number bigger than **divisor**.
     - If **divisor** has exceeded **i/2**, the number is prime.

7. In determining the factors as in the pseudo-code above, we are essentially using a divide-and-conquer approach. We start with small divisors and gradually increase.

   Consider 180...

   - Begin by considering a divisor 2. $180/2 = 90$, so **2** divides into 180.
   - Try 2 again, this time with 90. $90/2 = 45$, so **2** appears a second time in the factorization.
   - Try 2 again, this time with 45. Unfortunately 2 does not divide into 45, so we give up on 2 (and any other even number for that matter because 45 is odd).
   - Let's only consider odd numbers now, starting with 3. $45/3 = 15$, so **3** is a factor.
   - Try 3 again. $15/3 = 5$. Thus, **3** appears a second time.
   - Try 3 again, but 3 does not perfectly divide into 5, so we give up on 3.
   - We move to the next odd number, 5.
   - Well, $5/5 = 1$, so **5** is another factor.
   - And since we're down to 1, we're done.

   This method is safe and will allow us to only obtain the prime numbers in the factorization of a number: there's no danger of including composite numbers in this.

   Let's consider a prime example. Consider now 7.

   - Well, 2 fails to divide into 7,
   - as does 3,

- and we stop here, because the next factor we would consider, 5, is bigger than $7/2 = 3.5$.
- We conclude 7 is prime.

8. This can be made more efficient, but the focus of the course is not algorithm design... or prime numbers! Focus on writing good, working code, with proper documentation and management of variables.