ECOR 1606 Lab #5

<u>Note:</u> To prove their attendance at this lab, students are to submit file: "*lab5.cpp*" as Lab #5 for their Lab Section before the end of the lab period.

This lab has two purposes. It is intended to introduce students to working in the Dev C++ environment and to provide an example of how programs should be developed. It is best to develop programs by "progressive refinement" and to test each part of the program as it is completed.

This lab will involve writing a program that converts decimal (base 10) numbers between 1 and 3999 into Roman Numerals. Below is a reminder of how Roman Numerals work, followed by the problem statement and the nine steps you should follow to complete your program. **Step 0 (on the next page) will lead you through starting up Dev-C++.**

Introduction to Roman Numerals:

The Romans used letters to represent numbers. The letters used and their values are given below:

M = 1,000
D = 500
C = 100
L = 50
X = 10
V = 5
I = 1

In the simplest cases, interpreting a value expressed in Roman Numerals is a simple as adding up the digits:

MDCCCLXVII = 1867

This rule applies as long as all of the letters involved represent progressively smaller values (e.g. we start with M's and run down to I's). "Out of order" letters indicate that the value of the first letter is to be subtracted from the second.

CM = 900
CD = 400
XC = 90
XL = 40
IX = 9
IV = 4

Only the possibilities listed above are permitted. The concept does NOT extend to, for example, representing 999 as IM.

Problem Statement:

In this lab you will be producing a program that converts values between 1 and 3999 into Roman Numerals. Your program is to repeatedly read in integer values until zero is entered. For each value entered your program should either i) output an error message (if the value is not between 1 and 3999 inclusive) or ii) output the value in Roman Numerals.

Step 0:

Get the Dev-C++ environment and your C++ file ready to go. Here are the steps to follow:

- To Run Dev-C++: Under "start", select "All Programs; Bloodshed; Dev-C++
- In the Dev-C++ window:
    - Under "Tools" select "Compiler Options"; tick the box next to "Add the following commands when calling compiler" and, in the space below, enter **–Wall** (i.e. a minus sign, capital W, lower case "all") and then click "OK".
    - Under "Tools" select "Editor Options"; untick the "Smart Tabs" box; and tick the "Highlight matching braces/parentheses" box; and click "OK".
    - Under "File" select "Open" then go to G:/1606 and select "framework.cpp"
    - Under "File" select "Save as" and go to the Desktop, enter filename "lab5" or "lab5.cpp" and click "Save".
    - In the program comments, put "Lab 5: This program converts integers between 1 and 3999 into roman numerals", and your name and student number, and click the "Save" icon.
- Open CppSyntax.pdf so that you can refer to the C++ syntax as you work through the lab.

Step 1:

As a first step in solving the program we might come up with the pseudo code below *Process the value* is of course going to need some refining, but we can come back to this later.

```
read value
while the value is not zero do
   Process the value
   read value
endwhile
```

Implement the above pseudo code in C++.

Notes and hints:

- Because we will only be dealing with integer values, variable *value* should be an "int" variable.
- Remember that reading a value always involves outputting a message asking for input, and then reading it in.
- Don't forget the end "}"s. It's best to input them at the same time as the "{"s.
- Also remember the ";"s. Check the C++ Syntax document if you are unsure.
- Ensure that you indent the code inside each "while" and "if" by 4 spaces (1 tab).
- Put an output statement that outputs "Processing not yet implemented" (with an endl) in place of *Process the value*.
- Run your code and verify that it runs correctly (i.e. that it repeatedly accepts values until zero is entered).
- Save your code regularly!

Step 2:

The first step in refining *Process the value* is to replace it with

```
if the value is not between 1 and 3999 then
   output "Invalid value ignored" (with an endl)
else
   Convert value to Roman Numerals
endif
```

Replace your temporary output statement (the one that outputted "Processing not yet implemented") with C++ code that implements the pseudo code above.

Use an output statement that outputs "Conversion not yet implemented" (with an endl) in place of *Convert value to Roman Numerals*.

Run your code again and verify that it now rejects invalid values. Be sure to test that it works correctly for 1 and 3999. "Borderline" values often cause problems and should be included in any testing strategy.

Step 3:

The conversion process is simplified by first breaking the number down into thousands, hundreds, tens, and ones. This can be accomplished as shown below:

```
thousands = value / 1000
remainder = value % 1000
hundreds = remainder / 100
remainder = remainder % 100
tens = remainder / 10
ones = remainder % 10
```

All variables used should be "int" variables because all of the quantities involved are integer quantities. Also we **want** integer division here, and the "modulus" operator (**%**), which can only be used with "int" operands, is convenient.

Replace your temporary output statement (the one that outputted "Conversion not yet implemented") with C++ code that implements the pseudo code above followed by an output statement that outputs the values of *thousands*, *hundreds*, *tens*, and *ones*.

Run your program and verify that it correctly breaks values into thousands, hundreds, tens, and ones.

Step 4:

All we need to do now is to replace the output statement that outputs the values of *thousands*, *hundreds*, *tens*, and *ones* with code that instead outputs the required Roman Numeral. The pseudo code below gives the steps required.

```
output  value and  " in Roman numerals is " (no endl)
convert thousands into letters
convert hundreds into letters
convert tens  into letters
convert ones  into letters
output an endl
```

Replace the output statement that outputs the values of *thousands*, *hundreds*, *tens*, and *ones* with the two output statements called for by in this pseudo code.

In between these output statements, add four comments representing the work that remains to be done. The first comment should be "convert thousands into letters" and so on.

Test your program. For each valid input, you should now see:
"xxxx in Roman numerals is ", where xxxx is the value input

Step 5:

Dealing with the thousands is easy. The concept *convert thousands into letters* can be refined to:

```
while thousands is not zero do
   output an "M"
   thousands = thousands - 1
endwhile
```

Insert the C++ implementation of this pseudo code after the "convert thousands into letters" comment.

Run your code and confirm that the correct number of M's is output. For numbers less than 1000 you should get no M's at all.

Step 6:

Converting the hundreds into letters is a bit trickier because we must deal with both the possibility that a "D" can be used and the special cases of 900 and 400. The pseudo code below illustrates what is required.

```
if hundreds is 9 then
   output "CM"
elseif hundreds is 4
   output "CD"
else
   if hundreds is greater than or equal to 5 then
      output "D"
      hundreds = hundreds - 5
   endif
   while hundreds is not zero do
      output "C"
      hundreds = hundreds - 1
   endwhile
endif
```

Add the C++ implementation of this pseudo code to your program (after the "convert hundreds to letters" comment). Run your program and verify that it now deals with hundreds as well as thousands.

Step 7:

Add the code required to deal with tens to your program. This code is very similar to the code that deals with the hundreds, except that the special cases are now 90 and 40 instead of 900 and 400. (See page 1 for the roman numerals that you will need to use.)

Run your program and verify that it now deals with tens as well as with hundreds and thousands.

Step 8:

Add the code required to deal with ones to your program. This code is very similar to the code that deals with the hundreds (and tens), except that the special cases are now 9 and 4 instead of 900 and 400 (90 and 40). (See page 1 for the roman numerals that you will need to use.)

Run your program and verify that it now deals with ones as well as with tens, hundreds and thousands.

Step 9:
Go back and double check that the program is well-formatted (i.e. nicely indented) and easy to read (good comments, etc.). Add indentation, blank lines, and additional comments as needed.

Now that the program is complete, test your program with a variety of inputs, both valid and invalid, to ensure that it works as expected for **all** integer inputs (negative, zero, and positive).

Once you are happy with your program, submit it as "*lab5.cpp*" to prove your attendance at the lab.