Western Washington University
Computer Science Department

CSCI 141 Computer Programming I
Fall 2011

Assignment 2

Submitting Your Work

This assignment is worth 15% of the grade for the course. In this assignment you will write an Ada program to implement a calculator using roman numerals and a package body to provide the IO routines for numbers in roman numerals. Save your calculator program file and the package body file (the .adb files) in a single zipped tar file WnnnnnnnnAssg2.tar.gz (where Wnnnnnnnn is your WWU W-number) and submit the file via the **Assignment 2 Submission** item on the course web site. You must submit your assignment by 10:00am on Monday, November 14.

Roman Numerals

Roman numerals use the seven symbols I, V, X, L, C, D and M to represent the values 1, 5, 10, 50, 100, 500 and 1,000, respectively. The value of a number written in roman numerals is evaluated by considering the value of each numeral.

The following rules determine the reading and writing of roman numerals:

1. A repeated letter repeats its value that many times (XXX = 30, CC = 200, etc.).

2. A letter cannot be repeated more than three times. Therefore, since MMMM is an invalid sequence, the decimal value 4,000 cannot be expressed in roman numerals (using only these 7 characters).

3. If one or more letters are placed after another letter of greater value, add that amount.

   VI = 6 (5 + 1 = 6)

   LXX = 70 (50 + 10 + 10 = 70)

   MCC = 1200 (1000 + 100 + 100 = 1200)

4. If a letter is placed before another letter of greater value, subtract that amount.

   IV = 4 (5 – 1 = 4)

   XC = 90 (100 – 10 = 90)

   CM = 900 (1000 – 100 = 900)

5. Only subtract powers of ten (I, X, or C, but not V, L or D) from the following digit.

   For 95, do NOT write VC (100 – 5).
   DO write XCV (XC + V or 90 + 5)

6. Only subtract a single digit from the following digit.

   For 13, do NOT write IIXV (15 – 1 - 1).
   DO write XIII (X + I + I + I or 10 + 3)

7. Do not subtract a number from one that is more than 10 times greater (that is, you can subtract 1 from 10 [IX] but not 1 from 100—there is no such number as IC.)

   For 99, do NOT write IC (C – I or 100 - 1).
   DO write XCIX (XC + IX or 90 + 9)

8. Do not subtract a number from sequence of repeated digits of greater value. For example,

   For 19, do NOT write IXX (XX – I or 20 – 1)
   DO write XIX (X + IX or 10 + 9)

Under these rules, the smallest number expressible in roman numerals is 1. There is no concept of zero or negative numbers in roman numerals. Also, the largest number than can be expressed in roman numerals is MMMCMXCIX which is 3,999 in decimal notation.

The Calculator

The calculator must prompt for an expression to be calculated. The only operators to be implemented are '+' (addition), '-' (subtraction), '*' (multiplication) and '/' (integer division).

The calculator must allow for optional spaces to be input either side of the operator.

The numbers in the expression must be input in roman numerals and the result of the calculation must be output in roman numerals, but the calculations are performed on (decimal format) positive integer numbers.

For the input and output of numbers in roman numeral notation, the calculator program must use a separate package Roman_IO, described below.

Here are some example input-output sequences when the calculator program runs.

```
What do you want calculated? MCMLXXXVI + XXIV
MCMLXXXVI + XXIV = MMX

What do you want calculated? MMX – LXI
MMX - LXI = MCMXLIX

What do you want calculated? CXLIII * XXII
CXLIII * XXII = MMMCXLVI
```

```
What do you want calculated? MMMCXLVI / XXII
MMMCXLVI / XXII = CXLIII
```

If the result of the calculation goes outside the range of numbers that can be expressed in roman numerals, the calculator program must use exception handling to report the error, as shown in the following examples.

```
What do you want calculated? MMMCMXCIX + I
The result cannot be expressed in roman numerals

What do you want calculated? MMCXLVI – MMM
The result cannot be expressed in roman numerals

What do you want calculated? MMCXLVI * CCL
The result cannot be expressed in roman numerals
```

The calculator must use exception handling to report invalid operators, as shown in the following example.

```
What do you want calculated? MMCIV & XXIV
Illegal operator '&'
```

The package Roman_IO can raise a Roman_Numeral_Error exception if the input sequence of characters cannot be interpreted as a valid number in roman numerals. In that case, the calculator is to handle the exception and display the exception message, as shown in the following examples.

```
What do you want calculated? MXXXXIV + VII
An error has occurred: sequence of more than 3 repeated digits

What do you want calculated? VCII + III
An error has occurred: subtraction digit must be a power of ten

What do you want calculated? IIX + XII
An error has occurred: invalid sequence of digits

What do you want calculated? ICIV + XXI
An error has occurred: subtraction digit is too small

What do you want calculated? + XII
An error has occurred: empty roman numeral
```

Package Roman_IO

You are to write the body of package Roman_IO to implement the package specification roman_io.ads, which is provided on the course web site.

Procedure Get must read a sequence of characters from the keyboard, if possible interpret that character sequence as a decimal number and return that decimal number as the parameter Value.

Procedure Get must raise a Roman_Numeral_Error exception if the sequence of digits input from the keyboard cannot be interpreted as a valid number expressed in roman numerals. When it raises the exception it must set the exception message to some short description of the error. See the examples in the previous section for the type of messages that should be produced. (Note: those are only examples – your messages may be different, but should not be any less helpful to a user of calculator program).

Procedure Get must skip any leading white space characters before the number and must terminate its processing of the input and return the number value when it encounters either the end of line or a character which is not a valid roman numeral.

Procedure Get must allow for both lower case and upper case roman numerals.

Procedure Put must output the number supplied in the parameter Value in roman numerals, in accordance to the rules of roman numerals listed above.

What you must submit

1. You must submit the Ada source files (the .adb files) for the package body Roman_IO and your calculator program, which uses Roman_IO. These files must be included in a single zipped tar file.

2. Your calculator program and the package body for roman_io must be written in accordance with the coding standards listed below.

Coding Standards

1. Use meaningful names that give the reader a clue as to the purpose of the thing being named.

2. Use comments at the start of the program to identify the purpose of the program, the author and the date written.

3. Use comments at the start of each procedure to describe the purpose of the procedure and the purpose of each parameter to the procedure.

4. Use comments at the start of each section of the program to explain what that part of the program does.

5. Use consistent indentation:

   - The declarations within a procedure must be indented from the Procedure and begin reserved words. The body of the procedure must be indented from the begin and end reserved words. Example of procedure indentation:

```
procedure DoStuff is
     Count : integer;
     Valid : boolean;
begin
     Count := 0;
     Valid := false;
end DoStuff;
```

   - The statements within the then-part, each elsif-part and the else-part of an if statement must be indented from the reserved words if, elsif, else and end.

```
if Count > 4 and not Valid then
     Result := 0;
     Valid := true:
elsif Count > 0 then
     result := 4;
else
     Valid := false;
end if;
```

   - The statements within a loop must be indented from the loop and end loop.

```
loop
     Count := Count + 1;
     Get (Number);
     exit when Number < Count;
end loop;
```

   - The exception handlers and statements within each exception handler must be indented.

```
begin
     ...  -- normal processing statements
exception
     when Exception1 =>
          Put_Line (“An error has occurred”);
          Total := 0;
     when others =>
          Put_Line (“Something weird happened”);
end;
```