Assignment 3: Cobol Re-engineering (20%)

ROMAN NUMERAL CONVERSION

This assignment involves a program to convert a number expressed using Roman numerals to its decimal equivalent. The algorithm uses the Roman numerals I, V, X, L, C, D, and M - all others are considered invalid. The decimal equivalents of the above numerals are 1, 5, 10, 50, 100, 500, and 1000.

Figure 1 shown in the Appendix provides an algorithm flowchart for the problem.

1. Inputting the symbols in a Roman number.
2. Invoking a function sub-algorithm to convert the Roman number into its decimal equivalent.
3. Outputting the Roman number and its decimal equivalent.

The function sub-algorithm for performing the conversion is shown in Figure 2 (see Appendix). Note that the value of the variable sum is the one that is returned to the point of invocation. In addition the variable **err** has its value set within the sub-algorithm to indicate whether the return is normal (conversion was successful) or abnormal (an illegal Roman digital was encountered).

Note that the Roman number is input into a 15 character string and then moved to a record that contains an array of 15 one-character strings. Using this technique we are able to input the value as one string and operate on it one character at a time. Also note that the flowchart variable **sum** had to renamed **sum1** in the program since **sum** is a reserved word in Cobol. Finally, note that the input value and the error message are output together in the subroutine rather than separately as in the flowcharts.

The resulting program is provided in the file `roman2dec.cob`. This program is not complete, i.e. it is missing some information which would allow it to compile. The program processes a series of Roman numerals in an ASCII input file, `romnum.txt`, and writes the output to a file `decinum.txt`.

The ASCII input file has the following specification:

```
MDVC...........................................................................
DM.............................................................................
MDRI...........................................................................
```

The periods represent spaces in the file (filling out to 80 columns per line). The program allows input Roman numbers up to 15 characters in length.

TASKS

Given a partial Cobol program to convert Roman numerals to their decimal equivalents, get it working, and then re-engineer it. To do this, perform the following tasks:

1. The program is missing specification information related to file I/O. Add the appropriate file information that will allow the program to compile and run.

2. Migrate the program to a modern rendition of Cobol, making whatever changes are deemed necessary.
   - Modify any legacy features, e.g. old decision and loop structures etc.
   - Remove any archaic features, e.g. `next sentence`.
   - Update arithmetic statements to use `compute`.

3. Make improvements to the file I/O:
   - Allow the user to interactively specify the name of the input file during run-time.
   - Modify the input file contents, so it does not rely on having to add padding.
   - Output the decimal numbers to standard output (i.e. screen) instead of to a file.

4. Make improvements to usability:
   - Clean-up the output to make it more aesthetically pleasing. For example, printing a key to the Roman numerals might be helpful.

TESTING

A sample input for the **original** program is:

```
MDVC
CXIV
DM
MDRI
```

The corresponding output is:

```
   ROMAN NUMBER EQUIVALENTS
 ---------------------------
  ROMAN NUMBER     DEC. EQUIV.
 ---------------------------
 MDVC                   1595
 CXIV                    114
 DM                      500
 MDRI              ILLEGAL ROMAN NUMERAL
```

COMPILING

Please do not include a Makefile, and make sure your program compiles in the following manner:

```
> cobc -free -x -Wall roman2dec.cob
```

ASSIGNMENT INFORMATION

REFLECTION REPORT

Discuss your re-designed program in 1 (one) page (or more if you like) **reflection report** (single-spaced), explaining decisions you made in the re-engineering process. Consider this document a synopsis of your experience with your Cobol re-engineering process. You could do this by answering the following questions:

- What was the biggest challenge?
- Was it hard to find and fix the bug in the code?
- What parts of the Cobol program were the most challenging to comprehend?
- Could you imagine re-engineering a 10,000 line program?
- Why do you think Cobol has survived as long as it has?

DELIVERABLES

The submission should consist of the following items:

- The reflection report (PDF).
- The code (well documented and styled appropriately of course): **`roman2dec.cob`**
- Both the code and the reflection report should be submitted as a ZIP, TAR, or GZIP file.

STYLING & COMMENTS

Style consists of mnemonic variable names, indentation, and the use of whitespaces and paragraphing. The purpose of good style is to make the meaning of your program clear to someone who has never seen it before, cannot run it, and cannot talk with you. Documentation consists of in-code documentation. Examples of qualities to look for include:

- Are variable names well chosen?
- Are comments relevant rather than simple repetitions of the code?
- Do comments point out key sections of code, indicate special cases, or make assertions?
- Are the indents 3 or 4 spaces? Do not use tabs or 2 space indenting (please check "convert tabs to spaces" in your editor)
- Is whitespacing used to separate parts of the program to provide clarity?
- Is the program all in lowercase?

SKILLS

- This is an exercise in migration re-engineering, converting a program from one dialect of Cobol to another. It requires some research into the algorithm, and documentation of the re-engineered code.

APPENDIX: FLOW CHARTS

Figure 1:

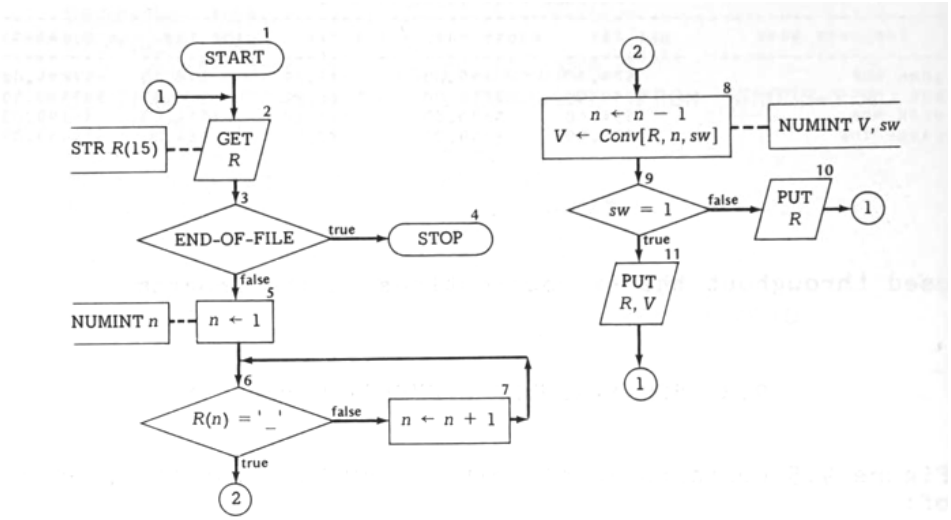
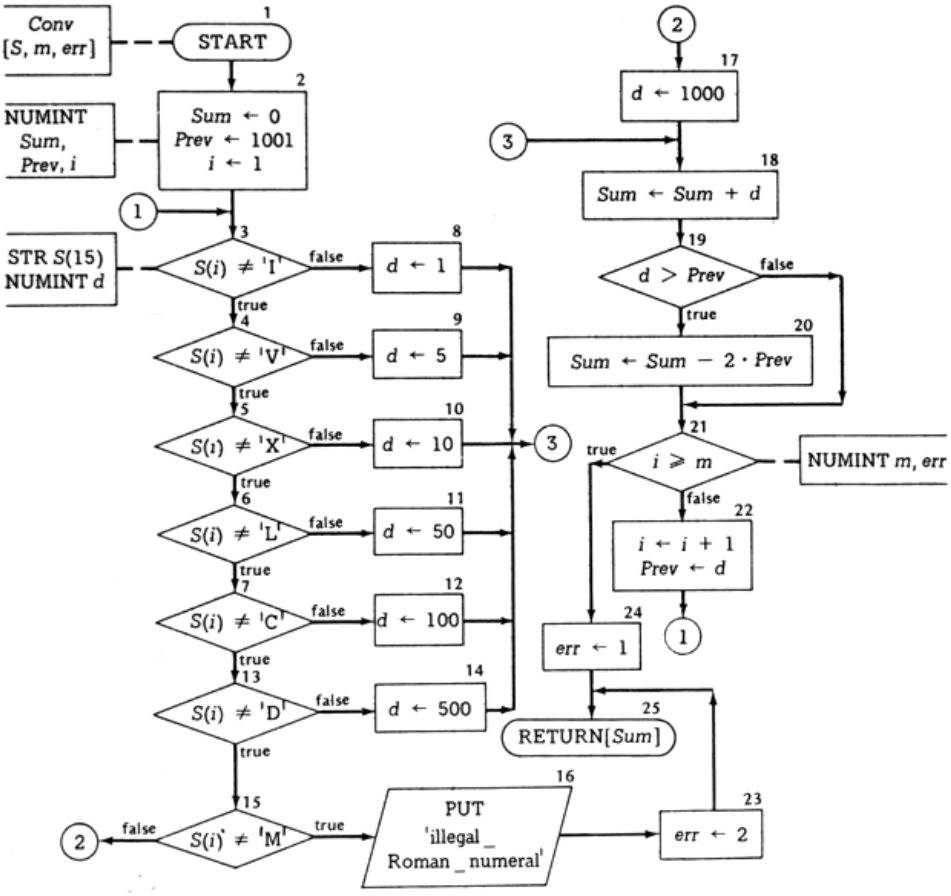


Figure 2:



```
IDENTIFICATION DIVISION.
PROGRAM-ID. ROMAN-NUMERAL.
*> PROGRAM TO CONVERT ROMAN NUMERALS TO THEIR DECIMAL
…EQUIVALENT

ENVIRONMENT DIVISION.

INPUT-OUTPUT DIVISION.

DATA DIVISION.

WORKING-STORAGE SECTION.
77 EOF-SWITCH   PIC 9    VALUE 1.
77 SWITCH       PIC 9.
77 N          PIC S9(2)  COMP.
77 SUM1         PIC S9(8)  COMP.
77 I            PIC S9(2)  COMP.
77 PREV         PIC S9(4)  COMP.
77 D            PIC S9(4)  COMP.
01 ARRAY-AREA.
   02 R         PIC X(1)   OCCURS 15 TIMES.
01 INPUT-DATA-RECORD.
   02 IN-R      PIC X(15).
   02 FILLER    PIC X(65).
01 OUTPUT-TITLE-LINE.
   02 FILLER    PIC X(28)  VALUE
                   "  ROMAN NUMBER EQUIVALENTS  ".
01 OUTPUT-UNDERLINE-1.
   02 FILLER    PIC X(30)  VALUE
                   "------------------------------".
01 OUTPUT-COLUMN-HEADINGS.
   02 FILLER    PIC X(14)  VALUE
                   "  ROMAN NUMBER".
   02 FILLER    PIC X(16)  VALUE
                   "     DEC. EQUIV.".
01 OUTPUT-UNDERLINE-2.
   02 FILLER    PIC X(30)  VALUE
                   " --------------------------- ".
01 OUTPUT-TABLE-RECORD.
   02 FILLER    PIC X      VALUE SPACE.
   02 OUT-R     PIC X(15).
   02 FILLER    PIC X(3)   VALUE SPACES.
   02 V         PIC Z(9).
```

```
01 OUTPUT-ERROR-MESS.
   02 FILLER    PIC X      VALUE SPACE.
   02 OUT-ER-R  PIC X(15).
   02 FILLER    PIC X(24)  VALUE
                    "   ILLEGAL ROMAN NUMERAL".

PROCEDURE DIVISION.
   OPEN INPUT INPUT-FILE, OUTPUT OUTPUT-FILE.
   WRITE OUTPUT-LINE FROM OUTPUT-TITLE-LINE
      AFTER ADVANCING 0 LINES.
   WRITE OUTPUT-LINE FROM OUTPUT-UNDERLINE-1
      AFTER ADVANCING 1 LINE.
   WRITE OUTPUT-LINE FROM OUTPUT-COLUMN-HEADINGS
      AFTER ADVANCING 1 LINE.
   WRITE OUTPUT-LINE FROM OUTPUT-UNDERLINE-2
      AFTER ADVANCING 1 LINE.
   READ INPUT-FILE INTO INPUT-DATA-RECORD
      AT END MOVE ZERO TO EOF-SWITCH.
   PERFORM PROC-BODY
      UNTIL EOF-SWITCH IS EQUAL TO ZERO.
   CLOSE INPUT-FILE, OUTPUT-FILE.
   STOP RUN.

PROC-BODY.
   MOVE IN-R TO ARRAY-AREA.
   MOVE 1 TO N.
   PERFORM SEARCH-LOOP
      UNTIL R(N) IS EQUAL TO SPACE.
   SUBTRACT 1 FROM N.
   PERFORM CONV.
   IF SWITCH IS EQUAL TO 1
      MOVE SUM1 TO V
      MOVE ARRAY-AREA TO OUT-R
      WRITE OUTPUT-LINE FROM OUTPUT-TABLE-RECORD
         AFTER ADVANCING 1 LINE
   ELSE NEXT SENTENCE.
   READ INPUT-FILE INTO INPUT-DATA-RECORD
     AT END MOVE ZERO TO EOF-SWITCH.

SEARCH-LOOP.
   ADD 1 TO N.

CONV.
```

```
   MOVE ZERO TO SUM1.
   MOVE 1001 TO PREV.
   MOVE 1 TO SWITCH.
   PERFORM CONVERSION-LOOP
      VARYING I FROM 1 BY 1
      UNTIL I IS GREATER THAN N OR
         SWITCH IS EQUAL TO 2.

CONVERSION-LOOP.
   IF R(I) IS EQUAL TO "I"
      MOVE 1 TO D
   ELSE IF R(I) IS EQUAL TO "V"
      MOVE 5 TO D
   ELSE IF R(I) IS EQUAL TO "X"
      MOVE 10 TO D
   ELSE IF R(I) IS EQUAL TO "L"
      MOVE 50 TO D
   ELSE IF R(I) IS EQUAL TO "C"
      MOVE 100 TO D
   ELSE IF R(I) IS EQUAL TO "D"
      MOVE 500 TO D
   ELSE IF R(I) IS EQUAL TO "M"
      MOVE 1000 TO D
   ELSE MOVE 2 TO SWITCH
        MOVE ARRAY-AREA TO OUT-ER-R
        WRITE OUTPUT-LINE FROM OUTPUT-ERROR-MESS
           AFTER ADVANCING 1 LINE.
   ADD D TO SUM1.
   IF D IS GREATER THAN PREV
      COMPUTE SUM1 = SUM1 - 2 * PREV
   ELSE NEXT SENTENCE.
   MOVE D TO PREV.
```