Programming Homework 1: Converting Integer to Roman Numeral System

Topics and References

- Arithmetic and relational expressions.
- Selection structures. See this page for details of the `if` statement and the `else` clause. Chapter 5 of the text contains information about selection statements.
- Input/output. See this page and this page for details about function `print` and its format specifications, respectively. See this page and this page for details about function `scanf` and its format specifications, respectively. Chapter 3 of the text contains information about functions `printf` and `scanf`.
- Programming Tutorial 4.

Problem Statement

Write a function `decimal_to_roman` in source file `q.c` that takes a positive decimal integer of type `int` and returns nothing. As its name indicates, the purpose of the function is to print to standard output the Roman numerals equivalent to the positive decimal integer that is passed to the function.

You do not require anything more than the problem-solving techniques that have been discussed the past four weeks: understanding decimal numbers and Roman numerals and an algorithm to convert decimal numbers to Roman numerals; sequence statements involving integer arithmetic and assignment expressions; relational expressions and selection structures involving `if-else` or `switch` statements; and the ability to write a single character to the standard output stream using either function `printf` or function `putchar`. And, of course, if you know more complex techniques, you're welcome to showcase them in your definition of the function.

Roman Numeral System

Numbers in Roman numeral system are formed according to the following rules:

1. Numbers are formed by combining 7 different numerals: I, V, X, L, C, D, and M. The decimal values represented by these Roman numerals is illustrated in the following picture:

Roman numeral	I	V	X	L	C	D	M
Decimal value	1	5	10	50	100	500	1000

2. These 7 Roman numerals are *usually* written largest to smallest from left to right to make up thousands of numbers using an *additive interpretation*. Here are some examples:
    - Number 2 (2×1) is written by mashing together numeral I like this: II.
    - Number 13 ($10 + 3 \times 1$) is obtained by mashing together numerals X and I like this: XIII.
    - Number 37 ($3 \times 10 + 5 + 2 \times 1$) is obtained by mashing together numerals X, V, and I like this: XXXVII.
    - Number 268 ($2 \times 100 + 50 + 10 + 5 + 3 \times 1$) is represented by mashing together numerals C, L, X, V, and I like this: CCLXVIII.

    - Number 3726 ($3 \times 1000 + 500 + 2 \times 100 + 2 \times 10 + 5 + 1$) is represented by mashing together numerals M, D, C, X, V, and I like this: $\mathrm{MMMDCCXXVI}$.
3. However, Romans did not like writing the same letter more than *three* times. For example, they did not want to write number 4 as IIII. Instead, they developed a method of *subtraction* to avoid repeating the same numeral four times. In this subtractive system, number 4 is represented as IV by placing smaller numeral I (1) before larger numeral V (5). This means that when you see a smaller numeral before a larger numeral, the subtractive interpretation must be employed. Therefore, when you see Roman numerals IX, since numeral I (1) is smaller than the next numeral X (10), you must employ subtractive interpretation to deduce that IX represents 9. On the other hand, if a larger numeral is before a smaller numeral, additive interpretation is employed. Using additive interpretation, Roman numerals VI represents decimal number $5 + 1 = 6$.
4. Romans used subtraction in six specific instances:
    1. Numeral I (1) is placed before numerals V (5) and X (10) to make values 4 (IV) and 9 (IX), respectively.
    2. Numeral X (10) is placed before numerals L (50) and C (100) to make values 40 (XL) and 90 (XC), respectively.
    3. Numeral C (100) is placed before numerals D (500) and M (1000) to make values 400 (CD) and 900 (CM), respectively.
5. Only numbers up to $3{,}999$ are represented.

Strategy for Converting Decimal Numbers to Roman Numerals

Begin with the complete list of Roman numerals formed by the subtractive rule:

Roman numeral	I	IV	V	IX	X	XL	L	XC	C	CD	D	CM	M
Decimal value	1	4	5	9	10	40	50	90	100	400	500	900	1000

and the representation of numbers 1 to 9 in Roman numeral system:

Roman numeral	I	II	III	IV	V	VI	VII	VIII	IX
Decimal value	1	2	3	4	5	6	7	8	9

Suppose you want to convert 3456 to Roman numerals. Remembering that numbers only up to 3999 can be represented in Roman numerals, write 3456 as $3 \times 1000 \ + \ 4 \times 100 \ + 5 \times 10 \ + 6$. Since Roman numeral M is 1000, 3×1000 can be written with Roman numerals MMM. The $4 \times 100 = 400$ can be represented from the above table with Roman numerals CD. The $5 \times 10 = 50$ is represented with Roman numeral L. Finally, digit 6 is represented by Roman numerals VI. Concatenating these Roman numerals provides the Roman numeral representation of 3456 as $\mathrm{MMMCDLVI}$.

Suppose you want to convert 987 to Roman numerals. Write 987 as $0 \times 1000 + 9 \times 100 + 8 \times 10 + 7$. Skipping 0×1000, $9 \times 100 = 900$ is represented from the above table with Roman numerals CM; $8 \times 10 = 80$ is represented as LXXX; and 7 is represented as VII. Concatenating these Roman numerals provides the Roman numeral representation of 987 as $\mathrm{CMLXXXVII}$.

Suppose you want to convert 46 to Roman numerals. Write 46 as $0 \times 1000 + 0 \times 100 + 4 \times 10 + 6$. Skipping 0×1000 and 0×100, $4 \times 10 = 40$ is represented from the above table with Roman numerals XL; 6 is represented as VI. Concatenating these Roman numerals provides the Roman numeral representation of 46 as XLVI.

Suppose you want to convert 9 to Roman numerals. Write 9 as $0 \times 1000 + 0 \times 100 + 0 \times 10 + 9$. Ignoring 0×1000, 0×100, and 0×10, 9 is represented from the above table with Roman numerals IX. Thus, the Roman numeral representation of 9 is IX.

Finally, year 2020 (written as $2 \times 1000 + 0 \times 100 + 2 \times 10 + 0$) will have Roman numeral representation MMXX. Next year 2021 will have Roman numeral representation MMXXI while the previous year 2019 will have Roman numeral representation MMXIX.

Function Behavior

If the function is given input value `1978`, the function must *exactly* print the following text to standard output:

```
MCMLXXVIII

```

Notice that function `decimal_to_roman` will insert a newline to standard output after writing the numerals representing the number in the Roman system.

If the function is given input value `555`, the function must *exactly* print the following text to standard output:

```
DLV

```

If the function is given the input `994`, the function must *exactly* print the following text to standard output:

```
CMXCIV

```

Driver Program

To test your definition of function `decimal_to_roman` in source file `q.c`, you'll require header file `q.h` that will provide declarations for C standard library function `printf` and the function you're defining `decimal_to_roman`:

```c
// header file q.h
#include <stdio.h>
void decimal_to_roman(int);

```

You can author your own driver that defines function `main` or use the following code in a source file `qdriver.c`:

```c
#include <stdio.h>
#include "q.h"

```

```c
int main(void) {
  printf("Enter a number (CTRL-D to quit): ");
  int value;
  while (1 == scanf("%d", &value)) {
    if (value <= 0 || value >= 4000) {
      printf("Enter a number (CTRL-D to quit): ");
      continue;
    }
    printf("%d: ", value);
    decimal_to_roman(value);
    printf("Enter a number (CTRL-D to quit): ");
  }
  printf("\nQuitting ...\n");
  return 0;
}

```

Compile (only) your source file `q.c` (which must include `q.h`) using the full suite of `gcc` options:

```
digipen@dit0701sg:/mnt/c/sandbox$ gcc -std=c11 -pedantic-errors -Wstrict-prototypes -Wall -Wextra -Werror -c q.c -o q.o
```

Separately compile (only) the driver source file `qdriver.c`:

```
digipen@dit0701sg:/mnt/c/sandbox$ gcc -std=c11 -pedantic-errors -Wstrict-prototypes -Wall -Wextra -Werror -c qdriver.c -o qdriver.o
```

Link both these object files plus C standard library functions (such as `printf` and `scanf`) into an executable file:

```
digipen@dit0701sg:/mnt/c/sandbox$ gcc q.o qdriver.o -o q.out
```

Test your code with a variety of input. Make sure to test what are called edge cases such as the smallest number 1 and the largest number 3999 and numbers such as 4, 9, 40, 90, and so on.

If you wish, you can collect all the various numbers you wish to test in a text file `your-input.txt`. You can then redirect the contents of `my-input.txt` to standard input stream and redirect the standard output stream to a text file `your-output.txt` like this:

```
digipen@dit0701sg:/mnt/c/sandbox$ ./q.out < your-input.txt > your-output.txt
```

However, note that to get pretty output, you must comment lines 5, 9, 14, and 16 of the driver source file `qdriver.c` presented above. You can download a sample executable `qsample.out`, a sample input file `qinput.txt`, and the corresponding output file `qoutput.txt` from the assignment web page. Remember that you can compare your output file `your-output.txt` and the correct output file `qoutput.txt` using `diff`:

```
digipen@dit0701sg:/mnt/c/sandbox$ diff -y --strip-trailing-cr --suppress-common-lines your-output.txt qoutput.txt
```

If `diff` is not silent, then your definition of function `decimal_to_roman` is incorrect and will require further work. Also remember to exhaustively test your implementation since text file `qinput.txt` does not contain an exhaustive sample of the 4000 possible numbers that can be provided as input to the function.

Submission and automatic evaluation

1. In the course web page, click on *Assignment 1 Submission Page* to submit `q.c`.
2. Please read the following rubrics to maximize your grade. Your submission will receive:
    - F grade if your `q.c` doesn't compile with the full suite of `gcc` options.
    - F grade if your `q.c` doesn't link to create an executable.
    - F grade if output of function doesn't match correct output of the grader (you can see the inputs and outputs of the auto grader's tests).
    - $A+$ grade if output of function matches correct output of auto grader.
    - The auto grader will provide a proportional grade based on how many incorrect results were generated by your submission.
    - A deduction of one letter grade for each missing documentation block. Your submission `q.c` must have one file-level documentation block and one function-level documentation block for function `decimal_to_roman`. A teaching assistant will physically read submitted source files to ensure that these documentation blocks are authored correctly. Each missing block will result in a deduction of a letter grade. For example, if the automatic grader gave your submission an $A+$ grade and one documentation blocks are missing, your grade will be later reduced from $A+$ to $B+$. Another example: if the automatic grade gave your submission a C grade and the two documentation blocks are missing, your grade will be later reduced from C to E.