```
Script started on 2023-11-29 18:23:52-06:00 [TERM="xterm" TTY="/dev/pts/0" COLUMNS=
mf98604@ares:~$ pwd
/home/students/mf98604
mf98604@ares:~$ cat roman.info

Name:        Philip May'r
Class:       CSC121-001

Activity:    Roman Numeral Conversion Program

Options:     Add (Level 2) to use a switch to good effect in your conversion code.
             Add (Level 2.5) to translate fractions.

Level:       7.5 (3 + 2 + 2.5)

Description: Converts Indo-Arabic numerals to Roman numerals.

mf98604@ares:~$ show-code roman.cpp


roman.cpp:
```

```cpp
#include <iostream>
#include <cmath>

using namespace std;

constexpr streamsize INF_FLAG{numeric_limits<streamsize>::max()};

short count_digits(short number)
{
    short digit_count = 0;

    while (number > 0)
    {
        number /= 10;
        digit_count++;
    }

    return digit_count;
}

int main()
{
    short number, numerator, denominator, digit_count;
    char y_n, slash;
    bool has_fraction;
    string roman;

    cout << "\n\t\tWelcome to the Roman Numeral Program!!!\n";

    while (tolower(y_n) != 'n')
    {
        roman = "";
        has_fraction = false;
        cout << "\nEnter a number: ";

        cin >> number;

        while (number < 1 || number > 3999 || !cin)
        {
            if (!cin)
            {
                cin.clear();
                cin.ignore(INF_FLAG, '\n');
                cout << "\nPlease enter a number "
                        "greater than 0 and less than 4000: ";
            }
            else if (number < 1)
            {
                cout << "\nPlease enter a number greater than 0: ";
            }
            else if (number > 3999)
            {
                cout << "\nPlease enter a number less than 4000: ";
            }
            cin >> number;
        }

        if (cin.peek() != '\n')
        {
            cin >> numerator >> slash >> denominator;
            has_fraction = true;
        }

        digit_count = count_digits(number);

        cout << "\n";

        for (short i = digit_count; i > 0; i--)
        {
            short digit, multiplier;
            multiplier = static_cast<short>(pow(10, i - 1));
            digit = static_cast<short>(number / multiplier % 10);

            string letters,
                   hundreds = "MDC",
                   tens = "CLX",
                   ones = "XVI";

            switch(i)
            {
                case 4:
                    for (short j = 1; j <= digit; j++)
                    {
                        roman += 'M';
                    }
                    continue;
                case 3:
                    letters = hundreds;
                    break;
                case 2:
                    letters = tens;
                    break;
                case 1:
                    letters = ones;
                    break;
            }

            if (digit == 9)
            {
                roman += letters[2];
                roman += letters[0];
            }
            else if (digit == 4)
            {
                roman += letters[2];
                roman += letters[1];
            }
            else if (digit > 4 || digit < 4)
            {
                if (digit > 4)
                {
                    roman += letters[1];
                    digit -= 5;
                }
                for (short j = 1; j <= digit; j++)
                {
                    roman += letters[2];
                }
            }
        }

        if (has_fraction)
        {
            short multiplier;

            multiplier = 12 / denominator;

            numerator *= multiplier;

            if (12 % denominator != 0)
            {
                cout << "Invalid fraction entered.\n"
                        "Only duodecimal (base twelve) fractions "
                        "are supported.\n";
                continue;
            }

            if (numerator > 5)
            {
                roman += 'S';
                numerator -= 6;
            }

            for (short i = 0; i < numerator; i++)
            {
                roman += '.';
            }
        }

        cout << roman << "\n\n";

        cout << "Would you like to convert another number? Enter yes/no: "

        cin >> y_n;
        cin.ignore(INF_FLAG, '\n');
    }

    cout << "\n";
}
```

```
mf98604@ares:~$ CPP roman
roman.cpp***


mf98604@ares:~$ ./roman.out

                Welcome to the Roman Numeral Program!!!

Enter a number: 2

II

Would you like to convert another number? Enter yes/no: yes

Enter a number: 88

LXXXVIII

Would you like to convert another number? Enter yes/no: Y

Enter a number: 555

DLV

Would you like to convert another number? Enter yes/no: y

Enter a number: 21347

Please enter a number less than 4000: 888

DCCCLXXXVIII

Would you like to convert another number? Enter yes/no: y
```

```
Enter a number: 2030

MMXXX

Would you like to convert another number? Enter yes/no: y

Enter a number: 12 1/4

XII...

Would you like to convert another number? Enter yes/no: y

Enter a number: 12 3/7

Invalid fraction entered.
Only duodecimal (base twelve) fractions are supported.

Enter a number: 24 5/6

XXIVS....

Would you like to convert another number? Enter yes/no: y

Enter a number: 2028

MMXXVIII

Would you like to convert another number? Enter yes/no: y

Enter a number: 97

XCVII

Would you like to convert another number? Enter yes/no: n

mf98604@ares:~$ cat roman.tpq

Thought Provoking Questions - Lab I -
Roman Numerals

1.)
There is a pattern behind Roman numerals. The units, tens, and hundreds
each have three different Roman numerals associated therewith.
The three numerals lie in order from least to greatest. The greatest
of the associated numerals for the units is the least
of the associated numerals for the tens. Likewise,
the greatest of the associated numerals associated with the tens is the
least of associated numerals for the hundreds. To represent numbers
multiplied by a factor of three or fewer, the base unit is simply repeated.
For a factor of four times the base unit, two numerals are used: the least
associated numeral, followed by the middle associated numeral. For a factor
of five, only the middle numeral is used. For factors greater than 5, but
fewer than 9, the middle associated numeral is used followed by an
appropriate number of least associated numerals, up to 3 times. For factors
of 9, the least associated numeral is followed by the greatest associated
numeral. For factors of ten, the least numeral of the next unit is used.

For instance, if Roman numerals were letters of the Latin alphabet,
starting from 'A' for the least (standing for 1), unto 'G' for the greatest
(standing for 1000):

The associated numerals for units would be 'A', 'B', and 'C'.
The associated numerals for tens would be 'C', 'D', and 'E'.
The associated numerals for hundreds would be 'E', 'F', and 'G'.

Then, the numbers from 1 to 9 would be represented by
'A', 'AA', 'AAA', 'AB', 'B', 'BA', 'BAA', 'BAAA', and 'AC'.

For a number of the base unit times a factor of 10, the next set of
numerals comes into play, and the least from among those is chosen.
Therefore, the number 10 would be represented by 'C'.

The same pattern then repeats in like fashion
for the multiples of tens and hundreds.


2.)
The main algorithm in the program for converting a representation
of a number written with Indo-Arabic numerals into a representation
using Roman numerals does not use modulus operations.
However, the block dealing with fractions does. A modular operation is used
to check whether or not the Indo-Arabic input fraction is valid,
as the Romans used base 12 fractions only.


3.)
The program will work only for values from 1 to 3999 because
there are not Roman numerals for zero or negative numbers, and
because for values greater than 3999, a bar (vinculum) is placed
over a numeral to multiply it by a factor of 1000. Implementing
such rare characters in a console program would likely be feasible,
albeit less than straightforward.

4.)
For the conversion of each digit, the algorithm uses three main branches:
one for a value of 9, one for a value of 4, and one
for values lesser than or greater than 4.


5.)
There are four 'for' loops in the conversion algorithm.
The first loop goes over each digit in the Indo-Arabic number one by one.
Within the scope of the first loop, there are three more loops.
The first loop takes charge of printing the Roman numeral equivalents of
the digits in the thousands' place of the original number. The
second one takes cares of printing the numerals associated with digits
greater than or lesser than four. The third and last loop prints dots for
fractions from seven to eleven twelfths.
```

```
6.)
Twenty-seven tests or so would likely be
required in order to completely test the program.
Nine tests for each base unit, whether ones, tens, or hundreds.
All numbers are but combinations of the twenty-seven unique combinations.


7.)
Roman numerals are still used today, mostly for stylistic, aesthetic,
emphatic, gravitational, or traditional reasons. A few examples:
book chapters, clock faces, and musical and mathematical notation.

8.)
The program can allow the user to type both 'y' and 'yes'
for their 'again' response by reading in only the first character
of the response and ignoring the rest.

9.)
The program can allow the user to type both 'y' and 'Y'
for their 'again' response by converting the first character
of the response to its lower-case equivalent.
If the character is already in lower-case, no change is effected.

10.)
Not applicable: associated option not implemented.

mf98604@ares:~$ exit
exit

Script done on 2023-11-29 18:29:55-06:00 [COMMAND_EXIT_CODE="0"]
```