```
Script started on 2023-10-25 13:06:26-05:00 [TERM="xterm" TTY="/dev/pts/6" COLUMNS=
ee43254@ares:~$ pwd
/home/students/ee43254
ee43254@ares:~$ cat roman.info
Name: Kyle Enkhzul
Class: CSC121-001

Activity: When in Rome...
Level: 6, 3 (base program), 3 (function writing)

Description:

The user inputs an integer they want converted to Roman numerals. Due to the
limits of Roman numerals, the user can only input integers from (0, 4000]. The
program will continue to reprompt if they want to continue the program with
a simple yes or no.

ee43254@ares:~$ show-code roman.cpp


roman.cpp:
```

```cpp
#include <iostream>
#include <string>
#include <limits>

using namespace std;

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

/*
    This method is intended to take a number input and check if it is
    a valid number. If it is not a valid number, the function reprompts
    the user until a valid number is passed through. It then stores the
    valid number into a variable and returns the variable.
    @return newNumber within bounds
*/
short checkValidNum(short number) {
    short newNumber;
    while( number > 3999 || number < 1) {
        if(number > 3999) {
            cout << "\nOh I'm sorry, that value is too large for Roman";
            cout << " civilization. \n";
            cout << "\nPlease try again with a number [strictly] smaller";
            cout << " than 4000, thank you. \n";
            cout << "\n Enter your number: ";

            cin >> number;
            }
        else {
            cout << "\nOh I'm sorry, that value is too small for Roman";
            cout << " civilization. \n";
            cout << "\nPlease try again with a number [strictly] bigger";
            cout << " than 0, thank you. \n";
            cout << "\n Enter your number: ";

            cin >> number;
            }
        }
    newNumber = number;
    return newNumber;
}
/*
    This method simply prints out the display result of the number
    translated to roman numerals.
    @return void
*/
void displayResult(string roman) {
    cout << "\nAh I believe that would be " << roman << ", right?\n";
}


int main() {

    char yes_no;
    string roman;
    short number;
    short temp;

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

    cout << "Will you converting numbers to Roman form with us today? ";

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

    while ( toupper(yes_no) != 'N')
    {
        cout << "\nExcellent! Glad to have you along! \n";
        cout << "\nEnter your number: ";

        cin >> number;

        number = checkValidNum(number);

    //thousands
    if(number >= 1000) {
        temp = (number / 1000);

        for(short i = 0; i < temp; i++) {
            roman += 'M';
        }
        number %= 1000;
    }

    // hundreds
    if( number >= 100) {
```

```
        temp = (number / 100) ;

        if(temp == 9) {
            roman += "CM";
        }
        else if ( temp >= 5) {
            roman += 'D';

            for(short i = 0; i < temp; i++) {
                roman += 'C';
            }
        }
        else if( temp == 4) {
            roman += "CD";
        }
        else if(temp >= 1) {
            for(short i = 0; i < temp; i++) {
                roman += 'C';
            }
         }
        number %= 100;
    }

    // tens
    if(number >= 10) {
        temp = (number / 10);

        if(temp == 9) {
            roman += "XC";
        }
        else if(temp >= 5) {
            roman += 'L';

            for( short i = 0; i < temp; i++) {
                roman += 'X';
            }
        }
        else if(temp == 4) {
            roman += "XL";
        }
        else if (temp >= 1) {
            for(short i = 0; i < temp; i++) {
                roman += 'X';
            }
        }
        number %= 10;
    }

    // ones
    if(number >= 1) {
        temp = number;

        if(temp == 9) {
            roman += "IX";
        }
        else if (temp >= 5) {
            roman += "V";

            for(short i = 0; i < number - 5; i++) {
                roman += 'I';
            }
        }
        else if (temp == 4) {
            roman += "IV";
        }
        else if (temp >= 1) {

            for (short i = 0; i < temp; i++) {
                roman += 'I';
            }
        }
    }


      displayResult(roman);

      cout << "\nWould you like to convert another number? ";
      cin >> yes_no;
      cin.ignore(INF_FLAG, '\n');
      roman.clear();
    }

    cout << "\nThank you for using the RNP!!\n";
    cout << "\nHave a good day\n";
}
ee43254@ares:~$ CPP roman
roman.cpp***


ee43254@ares:~$ ./roman.out

                Welcome to the Roman Numeral Program!!!

Will you converting numbers to Roman form with us today? yes

Excellent! Glad to have you along!

Enter your number: 4330

Oh I'm sorry, that value is too large for Roman civilization.

Please try again with a number [strictly] smaller than 4000, thank you.

 Enter your number: 0

Oh I'm sorry, that value is too small for Roman civilization.

Please try again with a number [strictly] bigger than 0, thank you.
```

```
 Enter your number: 1330

Ah I believe that would be MCCCXXX, right?

Would you like to convert another number? y

Excellent! Glad to have you along!

Enter your number: 132

Ah I believe that would be CXXXII, right?

Would you like to convert another number? nah...

Thank you for using the RNP!!

Have a good day
ee43254@ares:~$ cat roman.tpq
1. Is there a simple repeating pattern here that might help you extract
commonality and save coding time?

The simple repeating pattern for converting an integer to roman numerals is
extracting the single digit out of the thousands, hundreds, tens, or ones
place. By doing this, one can assign different roman numerals to the
respective values of 9, values greater than 5, and iterate in a for loop
for how many values below 5.

2. How does modulo fit into this scheme? (Hint: Look, for instance, at 2 and
7... or 1 and 6... or 3 and 8...)

Modulo allows the programmer to extract a single digit out of the thousands,
hundreds, tens by simply modulus dividing by a thousand, hundred, or ten
respectively.

3. Why will your program only work for values in the (integral) range
[1..3999]?

Roman civilization's organization of numbers in their Roman numeral system do
not have a value representing negatives and 0. For values greater than 3999,
Roman civilization instead used vinculum which is a sort of line over the ones
representation of thousands. Our program doesn't have the ability to represent
vinculum.

4. For the conversion of each digit to Roman form (except maybe the thousands
digit), you should have four branches. How many are cascaded from one another?

Thirteen branches are cascaded from one another.

5. How many of these branches are nested aside from cascading?

Seven branches are nested aside from cascading.

6. What is the purpose of each of the three loops in your program?

The purpose of each of the three loops in the program is to evaluate the digit
at the thousandth, hundreth, tenth, or ones place. Depending on if the digit is
a 9, greater than or equal to 5, greater than or equal to 4, and finally the
amount of ones. They all add Roman numerals accordingly to the digit in any one
of these digit places.

7. How many tests would be required to completely test this program?

Three tests would be required to completely test this program. One should be
less than or equal to 0. One should be greater than or equal to 4000. Lastly,
the final one should be a number within in the range (0, 3999].

8. Why does Jason want us to convert numbers from a dead civilization, anyway?

Even though Roman civilization is dead, the Roman numeral system still
has a lot of practical uses in modern civilization. This includes
denotations of book chapter titles, the Super Bowl, time clocks, and
titles of the Winter and Summer Olympics.

9. How can your program allow the user to type both y and yes for their
again response?

The program only tests if the first char of the input is not a 'N'. The user can
type in a word that is as long as they want but the program will only test the
first char. If the user types either a 'y' or a 'yes', the program evaluates it
all the same.

10. How can your program allow the user to type both y and Y for their again
response?

The program allows the user to type both 'y' and 'Y' for their response because
the program forces any char to an uppercase through .toupper(). This means that
regardless of what they input, it will always evaluate the uppercase allowing
both lowercase and uppercase input.

11. How can you have your program print different response text before
the Roman numeral result?

The program can print different response text before the Roman numeral result
by using a while loop to check if the number is greater than 3999 or less than
1. If the number is greater than 3999, then it will cout statements that tell
the user that the number is too large. It will then reprompt for another number.
If the number is less than 1, then it will cout statements that tell the user
that the number is too small. It will then reprompt for another number. ee43254@are
exit

Script done on 2023-10-25 13:06:55-05:00 [COMMAND_EXIT_CODE="0"]
```