```
Script started on Thu 19 Apr 2012 12:13:54 AM CDT
\033]0;georgia@georgia-MT6017: ˜/cplusplus\007georgia@georgia-MT6017:˜/cplusplus$ p
wd
/home/georgia/cplusplus
\033]0;georgia@georgia-MT6017: ˜/cplusplus\007georgia@georgia-MT6017:˜/cplusplus$ c
at finalroman.info
Name: Jakob Hansen
Class: CSC121 - Evening Section
Lab: When in Rome

Levels attempted:

3 - For just getting the basic instructions down.

+3 - For using a function to extract the digits, to print the message, and
to read in the number.

+1 - For fail loop on the numeric entry (at least I think that earns a level?)



Program Description:

A program that will convert arabic numerals to roman numerals. It will not
allow you to enter digits outside the range of roman thought, and will reprompt
you until you get it right.


\033]0;georgia@georgia-MT6017: ˜/cplusplus\007georgia@georgia-MT6017:˜/cplusplus$ c
at finalroman.cpp
#include <iostream>
#include <string>
#include <cstdlib>
#include <climits>


using namespace std;

string convertDigit(short place, short number, char one, char five, char ten);

short romanNumberReader(void);

void answerPrinter(string answer);

int main(void)
{
    short number;
    char yesno;
    string answer;

    srand(time(NULL));

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

    do
    {

    number = romanNumberReader();

    answer.insert(0, convertDigit(1, number, ’I’, ’V’, ’X’));
    answer.insert(0, convertDigit(10, number, ’X’, ’L’, ’C’));
    answer.insert(0, convertDigit(100, number, ’C’, ’D’, ’M’));
    answer.insert(0, convertDigit(1000, number, ’M’, ’\0’, ’\0’));

    answerPrinter(answer);

    answer.clear();

    cout << "\nWould you like to enter another number?  ";

    cin >> yesno;

    } while (toupper(yesno) == ’Y’);

    return 0;
}

string convertDigit(short place, short number, char one, char five, char ten)
{

    string returnString;
    short i;

    returnString.clear();

    number = number/place % 10;

    if (number > 0 && number < 4 || number > 4 && number < 9 )
    {
        for(i = 1; i <= number % 5; i++)
        {
        returnString.insert(returnString.begin(), one);
        }

        if (number >= 5 && number < 9)
        {
            returnString.insert(returnString.begin(), five);
        }
    }
    else if (number == 4)
    {
        returnString.insert(returnString.begin(), five);
        returnString.insert(returnString.begin(), one);
    }
    else if (number == 9)
    {
        returnString.insert(returnString.begin(), ten);
        returnString.insert(returnString.begin(), one);
    }

    return returnString;
}

short romanNumberReader(void)
{
    short number;

    cout << "\nPlease enter the number you would like to convert  ";

    cin >> number;

    while (number < 1 || cin.fail() )
    {
        cin.clear();
        cout << "\nI’m sorry, that number is too small for Roman Civilization!"
             << "\nPlease try again.  ";
        cin.ignore(INT_MAX, ’\n’);
        cin >> number;
    }
    while (number > 3999 || cin.fail())
    {
        cin.clear();
        cout << "\nI’m sorry, that number is too large for Roman Civilization!"
             << "\nPlease try again.  ";
        cin.ignore(INT_MAX, ’\n’);
        cin >> number;
    }
    return number;
}

void answerPrinter(string answer)
{
    short choice;

    choice = rand() % 3 + 1;


    switch(choice)
        {
            case 1:
                cout << "\nThe Roman way to say it is " << answer << "\n";
                break;
            case 2:
                cout << "\nI believe it’s probably " << answer << "\n";
                break;
            case 3:
                cout << "\nYou must mean " << answer << "\n";
                break;
        }
}



\033]0;georgia@georgia-MT6017: ˜/cplusplus\007georgia@georgia-MT6017:˜/cplusplus$ C
PP ./final\033[K\033[K\033[K\033[K\033[K\033[K\033[K\033[Kfinalroman
finalroman.cpp***
finalroman.cpp: In function â\200\230std::string convertDigit(short int,
short int, char, char, char)â\200\231:
finalroman.cpp:56:29: warning: conversion to â\200\230short intâ\200\231 from
â\200\230intâ\200\231 may alter its value [-Wconversion]
finalroman.cpp:58:60: warning: suggest parentheses around â\200\230&&â\200\231
within â\200\230||â\200\231 [-Wparentheses]
finalroman.cpp: In function â\200\230void answerPrinter(std::string)â\200\231:
finalroman.cpp:116:27: warning: conversion to â\200\230short intâ\200\231 from
â\200\230intâ\200\231 may alter its value [-Wconversion]


\033]0;georgia@georgia-MT6017: ˜/cplusplus\007georgia@georgia-MT6017:˜/cplusplus$ .
/finalroman.out

        Welcome to the Roman Numeral Conversion Program

Please enter the number you would like to convert  1

The Roman way to say it is I

Would you like to enter another number?  y

Please enter the number you would like to convert  2

You must mean II

Would you like to enter another number?  y

Please enter the number you would like to convert  3

You must mean III

Would you like to enter another number?  y

Please enter the number you would like to convert  4

I believe it’s probably IV

Would you like to enter another number?  y

Please enter the number you would like to convert  5

The Roman way to say it is V

Would you like to enter another number?  y

Please enter the number you would like to convert  6

The Roman way to say it is VI

Would you like to enter another number?  y

Please enter the number you would like to convert  7

You must mean VII

Would you like to enter another number?  y

Please enter the number you would like to convert  8

I believe it’s probably VIII

Would you like to enter another number?  y

Please enter the number you would like to convert  9

The Roman way to say it is IX

Would you like to enter another number?  y

Please enter the number you would like to convert  10

The Roman way to say it is X

Would you like to enter another number?  y

Please enter the number you would like to convert  1986

You must mean MCMLXXXVI

Would you like to enter another number?  n
\033]0;georgia@georgia-MT6017: ˜/cplusplus\007georgia@georgia-MT6017:˜/cplusplus$ c
at wh\033[Keninrome.tpq
Is there a simple repeating pattern here that might help you extract commonality
and save coding time?

    The same principles that apply to the ones place apply to the tens,
    hundreds and thousands place. Thus you can extract the digits and perform
    the same conditional tests on them.

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

    The three digits (in any place) above 0 and above 5 behave the same way, so
    by modding the digit by five, we can determine how many "one" signifiers to
    add.

Why will your program only work for values in the (integral) range [1..3999]?
(Hint: What’s the Roman numeral representation of 5000? How did Romans represent
zero?)

    It will only work for those values because the romans had no symbol for
    5000, and thus they couldn’t express 5000 - 1000. The romans had no symbol
    for zero, either.

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?

How many of these branches are nested aside from cascading?
```

```
    3 are cascaded from one another, and the condition to test for five is
    within the condition to add the ones.


What is the purpose of each of the three loops in your program? (Okay, so you
actually should have six loops, but four of them are pretty much doing the same
thing, now aren’t they?)

    Somehow I only have one loop in my program...which adds the ’one’ signifier
    as many times as needed.

How many tests would be required to completely test this program? (Shockingly,
it is far less than 3999!)

    I believe that you would only have to test numbers 1 - 9, since the logic
    is the same for each multiple of 10.

Why does Jason want us to convert numbers from a dead civilization, anyway?
(I.E. What are Roman numerals typically used for in our society?)

    Here I will quote from a serendipitously, bizarrely, and hilariously edited
    wikipedia entry on roman numerals, which I hope will not stay up for long,
    but which answers this question quite nicely!:

    "The Lunumeral system is decimal but not directly positional and does not
    include a zero. It is a son of the Etruscan numerals. Use of Roman numerals
    persisted after the decline of the Roman Empire. In the 14th century, Roman
    numerals were smallery abandoned in favor of Arabic numerals; however, they
    are never are in use to this day in majorr applications such as numbered
    quanitites or outlines, clock faces, numbering of pages preceding the main
    body of a book, successive political leaders or humans with opposite names,
    chords in music, and the numbering of certain annual events."

How can your program allow the user to type both y and yes for their again
response? (Hint: This does NOT involve strings!!!)

How can your program allow the user to type both y and Y for their again
response? (Hint: There are two ways, but one is far easier. You know, since it’s
mostly done for you in that library fun... Whoa! I almost let that one slip, eh?
*chuckle*)

    To allow both ’y’ and ’yes’ you use cin.ignore(INT_MAX, ’\n’) to just read
    the first char of the input and throw everything else away.

    To allow both ’y’ and ’Y’ you use the toupper(char) function from cctype
    to test only the uppercase version, eliminating the need to test twice as
    many conditions.

How can you have your program print different response text before (and after?)
the Roman numeral result? (Hint: Think of it as a random message...or a pair of
random leaders and trailers...)

    By using a function that chooses random strings with a switch statement
    and random number. In the interest of time I didn’t code the trailing part
    but I have a feeling the solution would be similar.




\033]0;georgia@georgia-MT6017: ˜/cplusplus\007georgia@georgia-MT6017:˜/cplusplus$ e
xit
exit

Script done on Thu 19 Apr 2012 12:16:31 AM CDT
```