CSC2402 Assignment 2

Full mark = 100% and is equivalent to 20 % of course mark.

Submit your assignment on-line using the course web page.

Assignment consists of two tasks. Task 2 is divided into 3 parts.

You are required to submit:

- Task 1: task1.cpp, number.cpp, number.h, task1.txt
- Task 2a: task2a.cpp, roman.cpp, roman.h, task2a.txt
  You will be using the same number.cpp and number.h as in task 1.
- Task 2b: task2a.cpp, binary.cpp, binary.h, task2b .txt
  You will be using the same number.cpp and number.h as in task 1 and same roman.cpp and roman.h as in task 2a.
- Task 2c: task2c.cpp, task2c.txt
  You will be using the same number.cpp and number.h as in task 1, same roman.cpp and roman.h as in task 2a, and same binary.cpp and binary.h as in task 2b.

The .cpp file should have the program code and comments about the algorithm and the coding. We may compile the .cpp files if required. The documentation files should contain the compilation messages and sample runs for each application program. All files must be in pure text format. No PDF, HTML, Word files, Open Office files, RTF etc. and no compression or archiving such as zip, rar, tar etc.

If you are using Codelite, the compilation messages can be copied and pasted with the usual crtl-c and crtl-v. For output of sample runs, right click the title bar of the command window, select EDIT/MARK and then highlight the output. With the output selected, right click the title bar of the command window, select EDIT/COPY. Position your cursor to a (new) text file opened in a text editor such as Notepad, and crtl-v will paste the copied output to the text file.

If you are using MinGW, right click the title bar of the command window; select EDIT/MARK and then high light the output. With the output selected, right click the title bar of the command window, select EDIT/COPY. Position your cursor to a (new) text file opened in a text editor such as Notepad, and crtl-v will paste the copied output to the text file.

If you are using Linux, you can cut and paste the output on the terminal window into a text such as vim.

Background information

From decimal to binary

We can change a decimal number to binary format by repeatedly dividing the decimal number and the reminders form the binary number from right to left.

Example: Convert 2402 to binary:

```
2 |2402       0    quotient is 1201 and remainder is 0 (right most digit)
2 |1201       1    quotient is  600 and remainder is 1
 2 |600       0    quotient is  300 and remainder is 0
 2 |300       0
 2 |150       0
   2|75       1
   2|37       1                 . . .
   2|18       0
   2|9        1
    2|4       0    quotient is    2 and remainder is 0
    2|2       0    quotient is    1 and remainder is 0
    2|1       1    quotient is    0 and remainder is 1 (left most digit)
      0
```

Resulting binary for decimal number 2402 is `100101100010`

US style decimal number

The US style uses a comma to separate every three digits of a decimal number.

Example: 2345 in US style is 2,345

Euro style decimal number

The Euro style uses a dot to separate every three digits of a decimal number.

Example: 2345 in Euro style is 2.345

Converting integers to string

We will learn <sstream> in chapter 9. One common use of <sstream> is to convert numbers to strings. When a number is pushed to a string stream, you can get back the content as a string or as a number.

```
#include <iostream>
#include <sstream>   // using string stream
using namespace std;

// Converting int to string
string int_to_string(int n){
    ostringstream outstr;
    outstr << n;                  // push an int to string stream
    return outstr.str();          // get a string
}

// Converting string to int
int string_to_int(string s){
    istringstream instr(s);      // form a string stream with the string
    int n;
    instr >> n;                  // get int from string stream
    return n;
}

int main(){
    cout << "Integer 2402 to string: ";
    cout << int_to_string(2402) << endl;

    cout << "String \"2402\" to int: ";
    cout << string_to_int(2402) << endl;

    return 0;
}
```

You may also use <sstream> to covert strings to decimal numbers. You can simply replace the int used in the functions to double. See textbook section 9.4 for more details.

Always use <sstream> instead of atoi(), … , sscanf(), …, sprintf() … functions.

Task 1 (20%)

Given the incomplete header file number.h:

```
class Number {
    public:
        // Default constructor without any initial value.
        // Initial value will be set to 0.
        // Value of integer is 0.
        // Value of US is "0".
        // Value of EURO is "0".
        Number();

        // Constructor with initial value.
        Number(int);

        // Returns int as a string in US style.
        string get_US_style()const;

        // Returns int as a string in Euro style.
        string get_EURO_style()const;

        // Returns the int.
        int get_number()const;

        // Sets the value of int, US, and EURO.
        void set_number(int);

    private:
        // A decimal number less than 4000.
        int number;

        // A string represent number in US style.
        string US;

       // A string representing number in EURO style.
       sting EURO;

       // converting int to string in US style
       void int_to_US();

       // converting int to string in EURO style
       void int_to_EURO();
};
```

Complete the header file by adding included libraries and directives. You are NOT allowed to modify the content between the pair of curly braces.

Make sure that you put in measures to prevent multiple inclusion of the header file.

Implement the Number class in number.cpp. Test your implementation using task1.cpp.

Task1.cpp should:

- Create Number *one*
  - Without initial value;
  - Display *one* as an integer;
  - Display *one* in US style;
  - Display *one* in EURO style;
- Set *one* to 1111
  - Display *one* as an integer;
  - Display *one* in US style;
  - Display *one* in EURO style;
- Set *one* to 99
  - Display *one* as an integer;
  - Display *one* in US style;
  - Display *one* in EURO style;
- Create Number *two*
  - With initial value 22;
  - Display *two* as an integer;
  - Display *two* in US style;
  - Display *two* in EURO style;
- Set *two* to 2022
  - Display *two* as an integer;
  - Display *two* in US style;
  - Display *two* in EURO style;
- Create Number *three*
  - With initial value 3333;
  - Display *three* as an integer;
  - Display *three* in US style;
  - Display *three* in EURO style;

Note: We will use MinGW 4.7.1 to compile programs if needed; executables will be run in Windows.

```
/*
Expected output:
Number one:
0
0
0
Number one:
1111
1,111
1.111
Number one:
99
99
99
Number two:
22
22
22
Number two:
2022
2,022
2.022
Number three:
3333
3,333
3.333
*/
```

Marking Scheme:

Criteria	Marks
Code to prevent multiple inclusions of header file	2%
Header file works properly	2%
Member functions properly implemented	8%
Program compiles and runs correctly	4%
Program layout for program which runs correctly	2%
Comments for program which runs correctly	1%
Sample run for program which runs correctly	1%

Task 2a (20%)

Using the Number class as the base class, derive a Roman class:

- The Roman class is the same as Number class except it also stores the integer as a Roman numeral.
- If a Roman is created without an initial value, then the Roman numeral is "INVALID".
- You are **not** allowed to use arrays, atoi(), atol(), atof(), printf(), sprintf(), fprintf(), vfprintf(), vprintf(), vsprintf(), scanf(), sscanf(), fscanf(). Feel free to use any other library functions provided by MinGW.

Create the roman.h and implement the Roman class in roman.cpp. Make sure that you put in measures to prevent multiple inclusion of the header file. Test your implementation using task2a.cpp.

Task2a.cpp should:

- Create Roman *one*
  - Without initial value;
  - Display *one* as an integer;
  - Display *one* in US style;
  - Display *one* in EURO style;
  - Display *one* as a Roman numeral;
- Set *one* to 2014
  - Display *one* as an integer;
  - Display *one* in US style;
  - Display *one* in EURO style;
  - Display *one* as a Roman numeral;
- Create Roman *two*
  - With initial value 22;
  - Display *two* as an integer;
  - Display *two* in US style;
  - Display *two* in EURO style;
  - Display *two* as a Roman numeral;
- Create Roman *three*
  - With initial value 2402;
  - Display *three* as an integer;
  - Display *three* in US style;
  - Display *three* in EURO style;
  - Display *three* as a Roman numeral;

Note: We will use MinGW 4.7.1 to compile programs if needed; executables will be run in Windows.

```
/*
Expected output:
Roman one:
0
0
0
INVALID
Roman one:
2014
2,014
2.014
MMXIV
Roman two:
22
22
22
XXII
Roman three:
2402
2,402
2.402
MMCDII
*/
```

Marking Scheme:

Criteria	Marks
Proper implementation of header file	2%
Member functions properly implemented	10%
Program compiles and runs correctly	4%
Program layout for program which runs correctly	2%
Comments for program which runs correctly	1%
Sample run for program which runs correctly	1%

Task 2b (30%)

Using the Roman class as the base class, derive a Binary class:

- The Binary class is the same as Roman class except it also stores the integer as a binary string.
- If a Binary is created without an initial value, then BINARY "0".
- You are **not** allowed to use arrays, atoi(), atol(), atof(), printf(), sprintf(), fprintf(), vfprintf(), vprintf(), vsprintf(), scanf(), sscanf(), fscanf(). Feel free to use any other library functions provided by MinGW.

Create the binary.h and implement the Binary class in binary.cpp. Make sure that you put in measures to prevent multiple inclusion of the header file. Test your implementation using task2b.cpp.

Task2b.cpp should:

- Create Binary *one*
  - Without initial value;
  - Display *one* as an integer;
  - Display *one* in US style;
  - Display *one* in EURO style;
  - Display *one* as a Roman numeral;
  - Display one as a binary string;
- Set *one* to 1024
  - Display *one* as an integer;
  - Display *one* in US style;
  - Display *one* in EURO style;
  - Display *one* as a Roman numeral;
- Create Roman *two*
  - With initial value 22;
  - Display *two* as an integer;
  - Display *two* in US style;
  - Display *two* in EURO style;
  - Display *two* as a Roman numeral;
- Create Number *three*
  - With initial value 2402;
  - Display *three* as an integer;
  - Display *three* in US style;
  - Display *three* in EURO style;

Note: We will use MinGW 4.7.1 to compile programs if needed; executables will be run in Windows.

```
/*
Expected output:
Binary one:
0
0
0
INVALID
0
Binary one:
1024
1,024
1.024
MXXIV
10000000000
Roman two:
22
22
22
XXII
Number three:
2402
2,402
2.402
*/
```

Marking Scheme:

Criteria	Marks
Proper implementation of header file	2%
Member functions properly implemented	14%
Program compiles and runs correctly	10%
Program layout for program which runs correctly	2%
Comments for program which runs correctly	1%
Sample run for program which runs correctly	1%

Task 2c (30%)

It is a handy feature if a word processor can convert and display numbers in various formats. To mimic that feature, you are asked to write an application task2c.app that:

- displays the following prompt:

```
Enter 1 if you want to change from decimal to Roman numeral
Enter 2 if you want to change from decimal to Binary numeral
Enter 3 if you want to display decimal in US style
Enter 4 if you want to display decimal in Euro style
Enter a non-digit to quit
```

- When user enters <1>, <2>, <3> or <4>, the program display the prompt:

```
Enter a positive integer smaller than 4000:
```

- After entering a positive integer less than 4000, the program will create a Binary class object and display the requested format of the integer. The application will display prompt for another input.
- If the user enters any character other than a digit, then the program will display a history of all the results and exit.

Note 1: You must use cin.fail() to check if non-digit is entered or not.

Note 2: You must use assert() to check if the menu choice is from 1 to 4.

Note 3: Use of arrays is not allowed.

Note 4: Run the program, use the number 2402 and 1748 for all the 4 options and finally quit with <Q>.

Marking Scheme:

Criteria	Marks
Prompts	4%
Proper use of cin.fail()	4%
Proper use of assert()	4%
Display of result history	4%
Program compiles and runs correctly	10%
Sample runs for program which runs correctly	1%
Comments for programs which runs correctly	1%
Program layout for program which runs correctly	2%

-- End of assignment 2 --