



## CMSHN1201 Programming Workshop

### Date and Number Conversion

#### Preface

This is a piece of work in which you will complete a partially-written program. The program is much larger than the ones you will be tackling in the weekly assessments. It should begin to give you a better idea of what 'real' programs look like. It should take you all semester to complete as it makes use of programming techniques taken from the entire module.

Where techniques are required that are not covered in the module, then you will generally find that that we have already written those bits of code. Your task is to 'fill in the blanks' (although there are more blanks than completed bits).

The program is an interactive menu-driven system that allows you to convert numbers and dates into different numbering systems and back again. It also contains a 'game' which will use the various components that you will write. Writing the program will involve solving several different types of programming problem, some of which are quite relevant in the context of the Y2K-problem (millennium bug).

#### Introduction

During the past couple of millennia, civilisations have used a number of different systems for measuring time and dates. This semester began on Monday 13<sup>th</sup> September, 1999 using the Gregorian calendar that we all take for granted today. However, the Gregorian calendar is a reasonably recent invention only being introduced in the sixteenth century (and not adopted by some countries until this century). Prior to that, Europe used the Julian calendar (named after its inventor, Julius Ceasar). Prior to the Roman Empire still other calendrical systems were used, some of which are still in concurrent use today (e.g. the Judaic and Islamic calendars).

The millennium bug (or Y2K) problem manifests itself most commonly through imprecise Gregorian date coding whereby the Gregorian year has been truncated to the its last two digits (e.g. 99 for 1999). One solution to the problem (and one that some programmers have always used) is to store dates as an offset from some fixed point in time. This offset can then be used to translate the date into a Gregorian date for output purposes. The system that is most popular is to store dates as Julian Day Numbers (JDN). A JDN is the number of days that have elapsed since the start of the Julian Calendar (midday on January 1<sup>st</sup>, 4713 B.C. on the Gregorian calendar). So, midday on the start of this semester would be Julian Day Number 2451435. Note that Julian Days start at noon, not midnight; thus, midnight on 13/9/199 which would be the start of the day according to international conventions would actually be JDN 2451434.5

By converting all dates to JDNs, the major part of the Y2K problem is easily solved. All that is needed is a means of converting dates into JDNs and back again (bank statements would not be popular if all transaction dates were printed as JDNs!).

A second aspect of our dates that we take for granted is the numbering system used. We express numbers in a base 10 system using digits derived from an Arabic system. By using such a system arithmetic is straightforward. However, consider the Roman Empire that had an altogether different numbering system. In the Roman system numbers were represented by combinations of the following primitives:

<b>I</b>	1
<b>V</b>	5
<b>X</b>	10
<b>L</b>	50
<b>C</b>	100
<b>D</b>	500
<b>M</b>	1000

So, 51 would be written as **LI**, 1500 as **MD** etc. Further, the numbers 4, 9, 40, 90, 400, and 900 are written as **IV**, **IX**, **XL**, **XC**, **CD**, and **CM** respectively. Thus 14 is **XIV**, 99 is **XCIX** etc. (Question: what is common to the numbers 9, 40, 90 and 900?).

In this system, the year 1999 would be written as **MCMXCIX**.

As you can imagine, arithmetic was not so simple using such numbers!

### Malformed numerals

You should be aware that just as English words can be misspelt, so can Roman numeral strings be malformed. For example, the Roman numeral string **MCMC** is malformed as it contradicts the syntax of the system. The rules for forming Roman numeral strings are<sup>1</sup>:

1. Generally speaking, smaller numerals follow larger numerals (see rule 3 below). In such cases, add up the values of the numerals to determine the quantity represented.
2. Numerals which are powers of ten (**I**, **X**, **C** & **M**) can be repeated up to three times in a row; other numerals cannot be repeated.
3. In certain cases, a smaller numeral may precede a larger one. Evaluate these expressions by subtracting the smaller numeral from the larger one. A smaller numeral can be placed before a larger one only if **all** the following conditions are met:
  - The smaller numeral must be a power of ten.
  - The smaller numeral must be either one-fifth or one-tenth the value of the larger one.
  - The smaller numeral must either be the first numeral in the expression, or be preceded by a numeral of at least ten times its value.
  - If another numeral follows the larger numeral, it must be smaller than the one that precedes the larger numeral.

### Requirements Specification

Your task is to complete the partially-written program (see Appendix A). The program will allow the user to convert Gregorian dates to Julian Day Numbers (and back again) and to convert decimal numbers to Roman numerals (and vice versa). A 'game' facility is also to be included that will quiz the user on the decimal value of random Roman numerals. Finally, a 'guess the day' function will allow the user to try and guess what day of the week a particular date fell on.

The main menu for the program, then, will look like:

```
Conversion Program
-----
1 : Decimal to Roman
2 : Roman to Decimal
3 : Gregorian to Julian Day Number
4 : Julian Day Number to Gregorian
5 : Roman number quiz
6 : Guess the day
7 : Quit

Select an action :
```

Selecting the first six options will cause the relevant **function** to be called. Option 7 causes the program to terminate.

#### 1. Decimal to Roman

Selecting menu option 1 should cause the screen to be cleared and the following dialogue to appear:

```
Enter a year (yyyy) : 1999

The year 1999 is written as MCMXCIX

Hit a key to continue
```

<sup>1</sup> Rules taken from Edward R. Hobbs' *Compvter Romanvs* resource (<http://www.naturalmath.com/tool2.html>).

Because of various complexities surrounding this operation, you are not required to write this function and we have provided it for you. However, you still need to write the dialogue and call the function appropriately.

The decimal year value is converted into Roman numerals by calling the function *DecimalToRoman*. The prototype for this function is:

```
char * DecimalToRoman (int) ;
```

The function takes an integer argument (the decimal year value) and returns a string (the Roman numerals). After the Roman numeral equivalent has been displayed, the user is asked to hit a key to return to the menu.

## 2. Roman to Decimal

The second menu option does the reverse of option 1; that is, the user enters a number in Roman numerals and the computer displays the decimal equivalent. The basic screen dialogue should look like:

```
Enter a Roman number : MCMLXVI
MCMLXVI. In decimal : 1966.
Hit a key to continue
```

The conversion is carried out by the procedural function *RomanToDecimal*. The function prototype is:

```
void RomanToDecimal (void) ;
```

Notice that the function takes and returns no values. In the above example, the Roman value entered by the user was syntactically valid. However, it is possible that the user may enter an invalid number and the program should account for this as follows:

```
Enter a Roman number : mcmc
MCMC. Sorry, your Roman number was malformed. Invalid portion : MCMC
Hit a key to continue
```

Lastly, it is possible to write Roman numerals that are inefficiently expressed whilst still being interpretable. For example, the value **XXXXX** obviously represents that value 50, but should be properly written as **L**. So, the program should be able to correct such sub-optimal user input as follows:

```
Enter a Roman number : xxxxx
XXXXX. In decimal : 50.
By the way, the number should have been written as L
Hit a key to continue
```

### 3. Gregorian to Julian

Menu option 3 allows the user to enter a Gregorian date in the format dd/mm/yyyy which the program will convert into the corresponding Julian Day Number. The screen dialogue should look like:

```
Enter a date (dd/mm/yyyy) : 25/12/1999

25/12/1999 = JD 2451538

Hit a key to continue
```

There is no requirement to validate the Gregorian date (for instance, one could enter 45/12/1999). The conversion is done by the function *GregorianToJulian*, prototype:

**long GregorianToJulian (int, int, int) ;**

The function returns the Julian Day Number as a long integer and takes as input three integers, one each for the day, month and year of the Gregorian date.

### 4. Julian to Gregorian

This menu option simply causes the program to accept a Julian Day Number from the user and presents the corresponding Gregorian date. Because of various complexities surrounding this operation, you are not required to write this function and we have provided it for you. However, you still need to write the dialogue and call the function appropriately. The dialogue should appear as:

```
Enter a Julian Day Number : 2451538

JD 2451538 = 25/12/1999

Hit a key to continue
```

The function you need to call is *JulianToGregorian*, prototype:

**void JulianToGreg (long, int\*, int\*, int\*) ;**

The function takes four arguments. The first is a long integer and should hold the Julian Day Number. The three remaining parameters are call-by-reference arguments and correspond to the day, month and year of the Gregorian date. This means that these three arguments will have their values **changed** by the function.

### 5. Roman number quiz

Menu option 5 uses the *DecimalToRoman* function in a game. The program selects a random number, and writes it in Roman numerals on the screen by using *DecimalToRoman*. If the user then types in the same number as represented by the Roman numerals then a congratulatory message is given:

```
What is the decimal equivalent of DIX ? : 509

Well done!

Hit a key to continue
```

If a wrong answer is given, then the screen would look like:

```
What is the decimal equivalent of MCCCVIII ? : 2001
Wrong! The answer is 1308
Hit a key to continue
```

## 6. Guess the day

Menu option 6 gets the user to enter a date and to guess the day of the week on which that date fell. Days of the week are to be entered as integers in the range 0-6 where 0 corresponds to Sunday and 6 to Saturday. The screen should look like:

```
Enter a date (dd/mm/yyyy) : 8/9/1999
What day of the week was 08/09/1999 (0-6)? : 3
Correct! It was a Wednesday!
Hit a key to continue
```

If the user guesses the day wrongly then, the screen would look like:

```
Enter a date (dd/mm/yyyy) : 8/9/1999
What day of the week was 08/09/1999 (0-6)? : 6
Sorry, it was a Wednesday (3)
Hit a key to continue
```

The game uses the function *GetDay* which takes a date and returns the day of the week for that date as an integer between 0 and 6. The prototype for the function is:

```
int GetDay (int, int, int) ;
```

The return value is the day-of-the-week and the three arguments will hold the day, month, and year of the date respectively. The function uses an additional function, *Lookup*, that returns a string corresponding to the day integer; that is, if 0 is passed to *Lookup* then the string "Sunday" would be returned. We have provided this function for you, so all you have to do is call it.

## Editing and Compiling the Program

Unlike the other workbook programs, this program uses 'separately compiled subprograms' (more of which in chapter 18 of the book). To help you get started, we have provided you with working versions of the functions listed above in the form of compiled *object code* files (files with an extension .OBJ). In other words, we coded the functions and compiled each of them into separate files. So, you can write the code for the main menu of the program, insert the appropriate calls to the various functions and test your menu by linking to these external .OBJ files during compilation. This means that you can get the menu working properly before worrying about the different functions. It also means that having got the menu working, you can just write each of the functions, one at a time, testing as you go. Once you have completed one function, you can then move on to the next.

So, you should tackle this coursework incrementally (in stages). The program you will eventually compile is called *dates.exe*. The program is made up by linking several files:

- **menu.c**— The C source file that you will be editing

- **RomToDec.obj**— The compiled file that contains a working version of the function *RomanToDecimal*
- **GregToJul.obj**—the working version of *GregorianToJulian*
- **GetDay.obj**— working version of *GetDay*

We link code from different files by using something called a **project**. On the module web page you will find a file called *dates.ide* which is the **project file** for the program.

You can find the project file, *menu.c* and the five OBJ files on the module's web site (go to [www.cms.livjm.ac.uk/paulvickers/hn1201/resource.htm](http://www.cms.livjm.ac.uk/paulvickers/hn1201/resource.htm)) You will notice that the OBJ files **do not** contain C source code.

Because this program (at least during the development stages) makes use of functions stored in separately compiled "object" files (.OBJ) compiling it requires a different set of actions You can compile the program by doing the following:

1. Copy the project file (*dates.ide*), the three object files (*RomToDec.obj*, *GregToJul.obj*, and *GetDay.obj*) and the main program file (*menu.c*) into the same folder on the 'M' drive.
2. Run the Borland C++ v5.02 compiler
3. Select Project|Open from the menu and use the Open Project File dialogue box to locate and open the project file *dates.ide*.
4. On the screen you should see two windows. The left-hand window should contain the file *menu.c* and the right-hand window should show a hierarchical structure with *dates.exe* at the top and *menu.c* and the five .OBJ files listed beneath it.
5. Make the necessary changes to *menu.c*
6. When you're ready to try compiling the program, right-mouse-click on *dates.exe* in the Project window on the right. From the menu that appears on the screen select Build Node. The compiler will then try to compile the program. If all goes well, it should eventually give you a 'Success' message.
7. You can run the program (*dates.exe*) by double-clicking on *dates.exe* in the project window.

## Writing the functions

So, you've written the code for the main menu and tested it and now you're ready to tackle writing the function *RomanToDecimal*. How do you do it? Well, if you examine *menu.c* you will see towards the bottom, sections of code that have been enclosed by comment brackets. If you look carefully you will observe that these commented-out sections contain skeletons for the various functions. So, to write your own version of *RomanToDecimal*, here's what you do:

1. Remove the comment brackets from around the function definition.
2. Enter the necessary lines of C code to complete the function.
3. Remove *romtodec.obj* from the project window (the one on the right) by right-mouse-clicking on *romtodec* and selecting Delete node from the pop-up menu. This ensures that the version of the function that you have just written will be used instead of the version we provided.
4. Compile the project as before.

You should repeat this process for the other two functions.

## Extra work

If you have enjoyed this piece of work and would like an additional challenge, then why not add another menu option to write Gregorian dates in the notation used by the Romans? For instance, September 25, 1999 (Gregorian) would be written in the Roman system as **pridie idvs september MMDCXXLII**. For full instructions on how to do this and for an explanation as to why the year in this example is given as 2752, rather than 1999, see Edward Hobbs' *Compvter Romanvs* on-line at <http://www.naturalmath.com/tool2.html>.

## Guidance

The program is interactive and involves lots of prompts and user data entry. A problem with data entered at the keyboard is that one must press the ENTER/RETURN key to submit the values. The problem is that that causes a special end-of-line character also to be inserted into the data-read-buffer which, if not dealt with, will screw up any following input. The following example shows how

use of an extra **char** variable solves the problem. Use this technique for accepting user input in the main menu:

```
printf ("Enter a year (yyyy) : ") ;  
scanf ("%d%c", &year, &dummy) ;
```

This puts the end-of-line marker into the char variable *dummy* hence removing it from the keyboard buffer.

All the files associated with the coursework can be downloaded from <http://www.cms.livjm.ac.uk/paulvickers/hn1201/resource.htm>. There you will also find two fully working versions of the program:

- *Dates.exe*— the MS-DOS version
- *DatesW.exe*— A Windows version.

Running the fully-working version will give you a feel for what the specifications really mean. The outline program you have been supplied mirrors the style of the exercises in the course textbook. However, you will find that we have already coded some of the design statements and that these pre-coded statements use features of the C language that have not been covered on the course up to this point. **DO NOT WORRY** if you do not follow the code we have provided. If you are interested in finding out how this code works, then read ahead in the book (relevant chapter/page numbers are given). If you have any problems, then make full use of the staff in the laboratory sessions and the tutorial.

Above all, try to enjoy it. Programming is fun! It can also be mind-numbingly frustrating at times, but hey, you didn't come to university just to let your brain atrophy.

## Appendix A Program Outline

```
/* Program to perform date conversions */
/* CMSCD1003 Programming Assignment, 1999 */

/* The include files conio.h, ctype.h, stdlib.h, string.h, and time.h */
/* are needed by some of the functions contained in this program */
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

/* Program Outline */
/*
do
{
Clear the screen
Display six blank lines
Display main menu and user prompt
Get user's choice
switch user choice of
1 : clear the screen and display six blank lines
    Prompt user to enter a year
    Accept year
    Display year in roman numerals
    Prompt user to hit a key to continue
    Accept keystroke
2: clear the screen and display six blank lines
    Call the RomanToDecimal function
    Prompt user to hit a key to continue
    Accept keystroke
3: clear the screen and display six blank lines
    Prompt for and accept a date (dd/mm/yyyy)
    Display date
    IF month Jan, Feb or March
        Display Gregorian_to_julian conversion of dd,mm,yyyy-1
    ELSE
        Display Gregorian_to_julian conversion of dd,mm,yyyy
    Prompt user to hit a key to continue
    Accept keystroke
4: clear the screen and display six blank lines
    Prompt for and accept a Julian day number
    Display Julian_to_gregorian conversion of JDN
    Prompt user to hit a key to continue
    Accept keystroke
5: clear the screen and display six blank lines
    select a random number, n
        Do this with the following lines of code-
        randomize ( ) ;
        n = rand ( ) % 4000 ;
    Display user prompt and Roman version of chosen number
    Accept user's guess
    IF guess equals chosen number
        Display congratulatory message
    ELSE
        Display comiserations and correct answer
    Prompt user to hit a key to continue
    Accept keystroke
6: clear the screen and display six blank lines
    Prompt for and accept date (dd/mm/yyyy)
    calculate day of week for the date
    Prompt for and accept user's guess of day of week
```

```

        IF guess equals day of week
            Display congratulatory message
        ELSE
            Display comiserations and correct answer
            Prompt user to hit a key to continue
            Accept keystroke
    }
    while user not choose to exit.
*/

void main ()
{
    /* Prototypes for the program functions you will write          */
    /* Functions are covered in detail in Chapter 10 of the book    */
    char * DecimalToRoman (int) ;
    void JulianToGregorian (long, int*, int*, int*) ;
    int GetDay (int, int, int) ;

    /* Prototypes for functions we have written for you */
    void RomanToDecimal (void) ;
    int translate (char) ;
    long GregorianToJulian (int, int, int) ;
    char * lookup (int day) ;

    /* Variable declarations. We have declared all the variables you need */
    /* Think carefully before introducing any new ones                    */
    int day,
        month,
        year,
        choice,
        cntr ;
    int n,
        guess,
        dayofweek ;
    char dummy ;

    long julian ; /* Why have we made this a long integer?          */

    /* Insert your code here                                         */

} /* End of function main                                          */

```