**COMP1917 Computing 1**

**Session 2, 2014**

**Project 2 - Message Board**

Due: Sunday 19 October, 11:59 pm   
Marks: 12% of final assessment

**Introduction**

One of the most popular means of exchanging information these days is the Message Board. In this project, you will be writing a C program to implement a simple text-based Message Board which allows messages to be added, deleted, listed, sorted and organized into Threads.

**Stage 0 - Provided Code**

We have provided a program in the directory [hw2/src](http://www.cse.unsw.edu.au/~cs1917/14s2/hw2/src) as a base for you to begin the assignment. Notice that it consists of the source file hw2.cand the header file hw2.h. Copy these files to your own directory and compile them by typing

gcc -Wall -o hw2 hw2.c

Now run the program. You will be presented with these options:

Enter command (A,L,P,F,B,D,R,T,S,U,Q, H for Help):

If you type 'h' for Help, you will see a more detailed list of commands:

A - Add

L - List

P - Print

F - Forward

B - Back

<k>- jump to message k

D - Delete

R - Reply

T - Threads

S - Search

U - Undo

Q - Quit

H - Help

Now type 'a' to Add a message. You will be invited to enter the following data:

Name: (a one-line string)

Date: (in the format dd/mm/yyyy)

Time: (in the format hh:mm:ss)

Text: (free-form text which may extend over several lines)

You signal the end of the "Text" by typing a dot '.' on a line by itself. You will see that the program then prints the message you have entered, using the function printList().

When you have finished, type 'q' to Quit.

**Stage 1 - Writing a Makefile, checking dates**

1. Write a Makefile as discussed in Lab 8, so that the program will be re-compiled when you type "make", producing an executable calledhw2.
2. Write code for the function dateOK() to check whether the date entered actually exists (for example, 29/02/2014 does not exist). You should assume that all dates are later than 1582 and conform to the Gregorian Calendar.

**Stage 2 - Adding, listing and printing items**

For Stage 2, you will need to implement (or complete) these commands:

A - Add

L - List

P - Print

You will need to complete the coding for the A command so that it links the newly-created node into a linked list. The items should be listed in the order in which they were entered (i.e. in order of increasing messageNum), which means that each newly added node will go to the end of the list.

The L and P commands should toggle the program between two different printing "modes" - "Brief" and "Full". In "Brief" mode, the program should print the list of all messages, in the following format:

1 John: Twist and Shout!

2 Paul: I read the news today, oh boy!

3 George: Here comes the sun!

-> 4 Ringo: I'd like to be under the sea

Each message should occupy one line. The "current item" (in this case, the most recently added one) should be distinguished by the characters "->" at the beginning of the line; the other items should begin their line with two blank spaces. The messageNum should then be printed in the format "%2d", followed by a blank space. Your program should then call the function printBrief() which prints the Name and the first line of the Text. In "Full" mode, only the current item should be printed, using the function printFull(), which prints the message in this format:

Message 4

Date: 26/09/1969

Time: 10:08:42

Name: Ringo

Text: I'd like to be under the sea

in an Octopus's garden in the shade.

After each successful A,L,P,F,B,<k>,D,R,T or U command, your program should either print the entire list of items or the current message, depending on the current printing mode.

**Stage 3 - Navigating the list**

For Stage 3, you will need to implement these commands:

F - Forward

B - Back

<k>- jump to message k

D - Delete

The F and B commands allow the user to move Forward and Back in the list by changing which item is the "current" item. The F command finds the next item after the current item and makes it the current item. The B command finds the previous item to the current item, and makes it the current item. The F command fails (has no effect) if the current item is the last item in the list; B has no effect if the current item is the first item in the list.

When a number is typed, your program should find the item with that messageNum and make it the current item. If there is no item with thatmessageNum, the command has no effect.

The D command should mark the current item as "Deleted" by setting msg->deleted to TRUE. In "Brief" mode, deleted items should still be included in the list but with the word "[deleted]" in place of the Name and Text, for example:

1 John: Twist and Shout!

2 Paul: I read the news today, oh boy!

-> 3 [deleted]

4 Ringo: I'd like to be under the sea

In "Full" mode, the function printFull() will print the deleted message like this:

Message 3 has been deleted.

When applied to a message that has already been deleted, the D command should have no effect.

**Stage 4 - Replying and Threads**

For Stage 4 you are to implement the commands:

R - Reply

T - Threads

The R command should have the same effect as the A command, except that the inReplyTo field in the new message should be set equal to themessageNum of the current message.

The T command should switch the printing mode to "Brief" and also change the order of the messages in the list so they are listed *by Thread*, as follows:

The message with the lowest MessageNum should remain the first item in the list. After that, any replies to that message should be listed, and any replies to those replies, etc., recursively (these messages constitute the "Thread" of the original message). After the messages in the thread of the original message have been listed, the earliest (lowest) of the remaining messages should come next in the list, followed by the messages in the Thread of that message, then the earliest of the remaining messages, etc. until all messages have been listed.

When the items are printed, each reply should be indented three spaces more than the message to which it is replying. For example, if Message 3 had been posted in reply to Message 1, and Message 4 had been posted in reply to Message 3, then the list should print like this:

1 John: Twist and Shout!

3 George: Here comes the sun!

-> 4 Ringo: I'd like to be under the sea

2 Paul: I read the news today, oh boy!

When the items are ordered by Threads, each newly entered Reply will not necessarily go to the end of the list but should instead be inserted into its proper place, according to Threads.

Any subsequent L command, as well as switching to "Brief" mode, should change the order of the messages in the list so that they are listed strictly in order of increasing messageNum, with no indentation (as described in Stage 2 above).

**Stage 5 - Searching the list**

For Stage 5 you are to implement a search command:

S - Search items

The S command should:

1. prompt the user like this:
2. Search text:
3. scan one line of text into an array of char (until the first newline character is encountered, or the maximum of 256 characters is exceeded)
4. search for all messages containing the search text within their Name or Text. Note that the search should NOT be case-sensitive (i.e. the strings "text" and "Text" **do** match). For each message where text appears in these fields, print the entire message, in the same format as the printFull() function described in Stage 2. The search text should be highlighted, wherever it occurs in the Name or Text field, by converting all lowercase letters to uppercase. Note that the actual contents of these fields should not be modified in any way. If the message is subsequently printed, it should appear as normal.

**Stage 6 - Undo**

For Stage 6, you need to implement one final command:

U - Undo

If the previous command was A,L,P,F,B,<k>,D,R or T, the U command should undo the previous command and return the list to its prior state. If the previous command was H,S or U, or if it had no effect, the U command should have no effect. (You do not have to Undo the Undo!)

**Sample files and tools**

Examples of input files are provided in the directory [hw2/sample](http://www.cse.unsw.edu.au/~cs1917/14s2/hw2/sample).

A compiled solution to the assignment has be provided for you in the [tools](http://www.cse.unsw.edu.au/~cs1917/14s2/hw2/tools) directory.  
Your program should produce EXACTLY the same output as this executable.

Feel free to copy chunks of code from the [list.c](http://www.cse.unsw.edu.au/~cs1917/14s2/lab/lab9/list.c) program, if you find it helpful.

**Groups**

This assignment may be done individually, or in groups of two students. Each group should send an email to cs1917.hw2@cse.unsw.edu.au by 30 September, indicating the names of the group member(s). We will then assign you an hw2group number. If you intend to do the assignment on your own, please send us an email for that as well. If you are working in a group, you are strongly advised to make use of file sharing tools like Apache Subversion or DropBox (which are safer and more convenient that emailing code back and forth to each other).

**Submission**

For this project you will need to submit a Makefile as well as program files (.c and .h files).

Once you have been assigned an hw2group number, you should submit by typing

give cs1917 hw2 Makefile \*.[ch]

Please ensure that you submit the source files and NOT any binary file (i.e., compiled hw2 or .o files). There is no need to submit the binary file as we will compile your program using your Makefile before testing it via the automarking program. It is therefore VERY IMPORTANT that you include a Makefile in your submission and that, when run with the make command, it will produce a compiled binary file called hw2 which performs exactly as described in the specification.

Only one member of the group needs to submit. You can submit as many times as you like - later submissions (by either member of the group) will overwrite earlier ones. You can check that your submission has been received by using the following command:

1917 classrun -check

The submission deadline is Sunday 19 October, 11:59 pm  
15% penalty will be applied to the (maximum) mark for every 24 hours late after the deadline.

Additional information may be found in the [FAQ](http://www.cse.unsw.edu.au/~cs1917/14s2/hw2/faq.shtml) and will be considered as part of the specification for the project. Questions relating to the project can also be posted to the MessageBoard on the course Web page.

If you have a question that has not already been answered on the FAQ or the MessageBoard, you can email it to your tutor, or tocs1917.hw2@cse.unsw.edu.au

**Marking**

This project will be marked on functionality in the first instance, so it is very important that the output of your program be EXACTLY correct. Submissions which score very low on the automarking will be looked at by a human and may receive a few marks, provided the code is well-structured and commented.

Programs that generate compilation errors will receive a very low mark, no matter what other virtues they may have. In general, a program that attempts a substantial part of the job but does that part correctly will receive more marks than one attempting to do the entire job but with many errors.

**Plagiarism Policy**

Your program must be entirely your own work. Plagiarism detection software will be used to compare all submissions pairwise (as well as submissions for similar assignments from previous years, if appropriate) and serious penalties will be applied, particularly in the case of repeat offences.

**DO NOT COPY FROM OTHERS; DO NOT ALLOW ANYONE TO SEE YOUR CODE**

Please refer to the [Yellow Form](http://www.cse.unsw.edu.au/people/studentoffice/policies/yellowform.html), as well as the [CSE Addendum to the UNSW Plagiarism Policy](http://www.cse.unsw.edu.au/~chak/plagiarism) if you require further clarification on this matter.

Good luck!