COSC2138 / CPT220: Programming Principles 2A
Study Period 4, 2014

Assignment 2 (20%) 100 Marks + Bonus 15 marks

Due date – Monday August 24th 2015, 11:59pm

Introduction

Please note that you must read the supplementary document "cpt220_general_info_2015.pdf " in the Assignments General Information section of Blackboard prior to reading this assignment. This document presents important general information pertaining to all assignments.

Scenario: In this assignment you are asked to implement a program simulating a "Word Link" game to demonstrate more advanced knowledge of C programming principles.

The concepts covered include:

- Command line arguments.
- Structures.
- File handling.
- Dynamic memory allocation and linked lists.
- Modularisation and multi-file programs.
- Makefiles.
- All concepts covered in assignment #1.

Your assignment must compile and execute cleanly on **jupiter, saturn or titan**. A **makefile** must be used to compile your program: **Of course, most of the development will be implemented in your own environment and testing performed on the Linux servers.**

Compile:

```
%make
```

Execute:

```
%./wordlink [command line args]
```

Start-up code will be provided on Blackboard in the Assignments section.

You are expected to have a modularized solution with multiple source files for this assignment. You are expected to use all start up code provided.

Permission to change the start-up code is not normally given, unless there is very good reason (i.e. a bug). You will need to build upon the code that is provided, instead. You can add extra functions and #defines. If you have any concerns about the start up code, please post your query to the Blackboard discussion board.

Functional Requirements

This section describes in detail all functional requirements of assignment #2. A rough indication of the number of marks for each requirement is provided. You should make a conscientious effort to write your program in a way that is capable of replicating the examples given in these requirements.

Base Requirements

"Word Link" is a game where two players alternate entering words with the requirement that the first letter of a word must match the last letter of the previous word thus making a "word chain" such as the one below:

```
apple
    elephant
            taxi
                igloo
                    oxygen
                        ......
```

The game is over when one player cannot give another word or an incorrect word is given. The same word cannot be used twice. Your implementation will use a dictionary file to determine which words are considered valid/invalid. The game can be played between two human alternating players, or a human player playing against a computer selecting words automatically.

Requirement #1 – Command line arguments - 5 marks

The user of your program must be able to execute it by passing in the names of one data file that the system is to use. You need to check that exactly 2 command-line arguments are entered and that the specified file is valid (that is, it exists). You need to abort the program and provide a suitable error message if this is not the case.

Your program will be run using these command line arguments:

```
%./wordlink  <dictionary-file>
```

For example:

```
./wordlink  words-small.txt
```

Requirement #2 – Load Data - 15 marks

For this requirement you will be extracting all of the words from the dictionary file to be loaded into the linked-list data structure provided with the start up code. The dictionary file is any plain text file containing any number of words with one word per line.

Your data structure stores your words in 26 linked-lists with each linked list representing words beginning with one letter of the alphabet. The linked lists will remain unsorted by way of simply inserting each new word in a ***random*** position of the correct linked list. Duplicate words are not allowed. It is critical for your program to be able to load the smallest supplied data files as a minimum.
The data files were obtain from http://wordlist.aspell.net/scowl-readme/

Requirement #3 - Main menu - 5 marks

Your program must display an interactive menu displaying seven options like the one below:

```
Main Menu:
(1) 1 Player Game
(2) 2 Player Game
(3) Dictionary Summary
(4) Search Dictionary
(5) Add Dictionary Word
(6) Save Dictionary
(7) Exit

Select your option (1-7):
```

Your program must allow the user to select these options by typing the number and hitting enter. Upon completion of all options except "Exit", the user is returned to the main menu. The behavior of these options is described in requirements #4 – #10.

Requirement #4 – 1 Player Game – 5 marks

This option runs the game between the user and a computer controlled opponent. Each word that is used is "marked" in the dictionary so that it cannot be used again in the current game. Hence you should not forget to "reset" the dictionary at the start of each game.

The player will lose if an incorrect word is entered or he/she chooses to give up by entering an empty line. The computer opponent will lose when it is exhausted of guesses. Example:

```
1 Player Game
-------------

Player: Enter a word (1-20 char, blank line to quit): apple
Computer selects word "elephant".
Player: Enter a word (1-20 char, blank line to quit): taxi
Computer selects word "igloo".
Player: Enter a word (1-20 char, blank line to quit): zzzzz
"zzzzz" is not in the dictionary. Computer wins!
```

Other terminating messages:

```
Computer could not make a guess. You win!
You entered a blank line to give up. Better luck next time.
```

Note: In requirement 2, it was mentioned that the linked list is ordered ***randomly***. This is to make sure that the 1 player game is always different when the computer player makes guesses by selecting words from the start of the relevant linked list.

.

Requirement #5 – 2 Player Game – 5 marks

This option runs the game between two alternating human controlled players. Other rules as per the one player game apply. Example:

```
2 Player Game
-------------

Player 1: Enter a word (1-20 char, blank line to quit): apple
Player 2: Enter a word (1-20 char, blank line to quit): elephant
Player 1: Enter a word (1-20 char, blank line to quit): taxi
Player 2: Enter a word (1-20 char, blank line to quit): igloo
Player 1: Enter a word (1-20 char, blank line to quit): zzzzz
"zzzzz" is not in the dictionary. Player 2 wins!
```

Requirement #6 – Dictionary Summary – 5 marks

This option reports some statistics concerning the contents of the dictionary. This option reports the number of words beginning with each letter and the total number of words:

```
Dictionary Summary
------------------

+--------+--------+
| a: 109 | n:  25 |
| b:  23 | o:   4 |
| c:   9 | p: 127 |
.................  <- (Add other rows here)
| m:  10 | z:   1 |
+--------+--------+

Total Words: 567
```

Requirement #7 – Search Dictionary – 5 marks

This option prompts the user to enter a word and the program reports on whether or not the word is in the dictionary. Example:

```
Search Dictionary
-----------------

Enter a word (1-20 char): abcde
"abcde" is NOT in the dictionary.
```

Requirement #8 – Add Dictionary Word – 5 marks

This option prompts the user to enter a new dictionary word to be stored in the dictionary data structure. The new word is added to the end of the relevant linked list. You must make sure that duplicate words are not inserted into the data structure.

```
Add Dictionary Word
-------------------

Enter a word (1-20 char): hello
"hello" has been added to the dictionary.
```

Requirement #9 – Save Dictionary – 5 marks

This option writes all words in your dictionary back to the original data file. This file is overwritten. Each word is output on a separate line.

Requirement #10 – Exit – 5 mark

This option terminates your program. Memory deallocation is performed before the program terminates.

Requirement #11 – "Abort" – 5 marks

This option should terminate your program. All program data will be lost. You should be freeing memory at this point as well.

Requirement #12 – Return to Menu Functionality – 3 marks

Your program should allow the user to return to the main menu at any point during data input from the user. The user can do this by simply hitting enter. i.e. an empty line was entered or pressing ctrl-d on an empty line.

Requirement #13 – Functional Abstraction -5 Marks

We encourage the use of functional abstraction throughout your code. It is considered to be a good practice when developing software with many benefits. Abstracting code into separate functions reduces the possibility of bugs in your project, simplifies programming logic and eases the need for debugging. We are looking for some evidence of functional abstraction throughout your code. As a rule, if you notice that you have the same or similar block of code in two different locations of your source, you can abstract this into a separate function.

Requirement #14: Proper Use of an ADT – 4 marks.

In this assignment, you are implementing an Abstract data type . For this requirement you will need to propose a list of interface functions for a list and implement these. All

reference to these types should be via these interface functions.

Requirement #15 – Makefile - 4 marks

Your project must be compiled using a Linux makefile. All compiler commands must include the "`-ansi -Wall -pedantic`" compile options and compile cleanly with these options.

Your makefile needs to compile your program incrementally. i.e.: use object files as an intermediate form of compilation.

Also include a directive called "clean" that deletes unnecessary files from your working directory such as object files, executable files, core dump files, etc. This directive should only be executed when the user types "make clean" at the command prompt.

Have a look at the courseware, Week 7 Laboratory for examples of makefiles.

Requirement #16 – Memory leaks - 4 marks

The start up code requires the use of dynamic memory allocation. Therefore, you will need to check that your program does not contain memory leaks. Therefore, you will need to check that your program does not contain memory leaks. Use the:
"valgrind --leak-check=full --show-reachable=yes <command> <arguments>"
to check for memory leaks.
Marks will only be awarded for this requirement if the feedback valgrind provides reports zero memory leaks and no other memory related problems.
Another common problem in is memory abuses. These are inappropriate accesses such as reading from uninitialized memory, writing to memory addresses you should not have access to and conditional statements that depend on uninitialized values. You can test for these again by using valgrind:
"valgrind --track-origins=yes <command><arguments>"

Requirements #17-19 – General Requirements – 11 marks

You must implement Buffer Handling, Input Validation, and Coding Conventions as per the requirements listed on the "Assignment General Information" sheet (available on Blackboard. These requirements are going to be weighted as:

- Buffer Handling: 4 marks
- Input Validation: 4 marks
- Coding Conventions: 3 marks

Optional Requirements

These requirements outline some interesting extensions to your assignment that show you some ways that you can extend what you have learnt in this course. These requirements will only be assessed if you get 80% or more on the rest of the assignment so if you are struggling with this assignment, you are not encouraged to try these out during the pressure to get the work for assignment 2 done.

If you are going to attempt these sections you MUST submit one zip file containing two directories. The first directory must be named BASIC and contain the code for the core specifications listed above. The second directory must be named OPTIONAL and contain the code for the the optional requirements. You can and should include a second readme.txt in the OPTIONAL directory with details of your changes to the skeleton code and listing which of the optional requirements you have implemented. If you are not attempting any of the optional requirements then do not submit any directories in your zip file as the standard penalties will still apply.
However, these requirements are intended to both give students who want more of a challenge than is already provided a chance to test out more ideas and to further your C programming skills.
If you get over 100% in this assignment due to these bonus marks, these marks may be applied to your mark in other parts of this course. Also note that these requirements involve more effort than would normally be allocated for this amount of marks. They are meant to be challenge tasks for those interested in furthering their learning.

Binary I/O (Bonus 5 marks)

For this requirement you will be required (on top of the other requirements in this assignment) to use fread() and fwrite() to be able to read data for your linked list from a binary file and write it back to a binary file. So, instead of storing the struct data in its ascii representation, you will write functions which will allow the data stored in each customerNode and stockNode to be written to disk directly as binary.
If you are looking for a way to start on this requirement, have a look at the fwrite() example in week 6's material. For this requirement, you would also need to provide a way that the use can specify from the command line to use binary I/O for either saving or loading rather than ascii I/O.
You will only get marks for this section for a successful implementation. If you attempt this section and it does not work there will be no marks allocated. The marks will be allocated as follows:
Successful writing of the binary file: 2.5 marks.
Successful reading of the binary file: 2.5 marks.

Generic Programming (Bonus 5 marks)

The linked lists described in this assignment is for a particular type – a customerNode or stockNode node. The problem
with this is that if we want a different kind of linked list, we would need to make many modifications to allow for this. The task in this requirement is to use void pointers instead for the data pointers and provide functions for comparison of the type. Note that this is a fairly advanced task so you should only attempt this if you are a fairly comfortable C programmer.
Please note that you will not get the marks for this are only attainable if you submit a correct implementation. There are no half marks for this requirement. With this in mind, we encourage you to submit two versions of your assignment if you are aiming for these marks – a version that uses generics and a version that does not in case the generic version does not work as expected, we do not want you to fail the assignment.

Pointer to Pointer Implementation (Bonus 5 marks)

The linked list used in the normal startup code version of the assignment passes around a pointer to the head of a list. But you will find that an alternative is to pass around instead a pointer to a pointer to the head of the list so we can change the headnode. To get these marks, you need to alter the startup code so that you are passing around a pointer to the head node pointer rather than a pointer to the head.

Deliverables and Submission Details

Submission date/time:

Submission details for assignment #1 are as follows. Note that late submissions attract a marking deduction of 10% per day for the first 5 university days. After this time, a 100% deduction is applied.

CATEGORY	DUE DATE/TIME	PENALTY
On time	**Monday 24/8/2015 11:59pm**	**N/A**
1 day late	Tuesday 25/8/2015 11:59pm	10% of total available marks
2 days late	Wednesday 26/8/2015 11:59pm	20% of total available marks
3 days late	Thursday 27/8/2015 11:59pm	30% of total available marks
4 days late	Friday 28/8/2015 11:59pm	40% of total available marks
5 days late	Saturday 29/8/2015 11:59pm	50% of total available marks
6 or more days late	Not accepted	100% of total available marks

We recommend that you avoid submitting late where possible because it is difficult to make up the marks lost due to late submissions.

Submission content:

For assignment #2, you need to submit at least your implementations of these 8 files. However, you can add your own .c and .h if required to these 8 files.

`wordlink.c`
This file will include your main function.

`wordlink.h`
Header file for wordlink.c.

`list.c`
This file will include your list interface functions.

`list.h`
Header file for **`list.c`**

`wordlink_options.c`
This file contains functions for each of the six major menu options.

`wordlink_options.h`
Header file for wordlink_options.c.

`wordlink_utility.c`

This file will contain additional code for the running of your program. For example, this can include your own functions that help you collect and validate user input. It may also contain functions to load the data files and initialize your system to a safe state.

`wordlink _utility.h`
Header file for wordlink_utility.c.

`makefile`
This file will compile your program in an incremental fashion.

`README`

This file will contain important additional information about your program that you wish your marker to see. Examples include incomplete functionality, bugs, assumptions, and so on. It is recommended that you ask about any assumptions you want to make in the Assignment 2 Discussion Forum on Blackboard first before you make them.

Assignment Regulations

- You may refer to textbooks, notes, work in study groups etc. to discover approaches to problems, however the assignment should be your own individual work.
- Where you do make use of other references, please cite them in your work. Students are reminded to refer and adhere to plagiarism policies and guidelines: RMIT CS&IT Academic Integrity: http://www.cs.rmit.edu.au/students/integrity/

How to Submit

This assessment task is due at 11.59pm on Monday 24th August 2015

Late submissions must first be approved by the Online Programs Administrator by completing the "Assignment Extension Form" at this URL:

http://oua.cs.rmit.edu.au/procedures/forms.html

Assignments may be accepted up to 6 days after the deadline has passed; a late penalty of 10% will apply for each day late. Submissions over 4 days late will not be marked.

Emailed submissions will not be accepted without exceptional circumstances.

Submission will take place through Weblearn. As your submission will comprise of multiple-files, please submit a compressed archive assignment2.zip of your work with the files named wordlink.c, wordlink.h, list.c, list.h, wordlink _options.c, wordlink _options.h, wordlink _utility.c, wordlink _utility.h, makefile and README) in one of the following formats:

- ZIP compressed file (.zip; created by Windows XP, WinZIP Legacy, MacOS X, etc.)
- tar/gzip compressed file (.tar.gz; created by the Unix tar and gzip tools)

Important Note: **other compression formats, such as (but not limited to) RAR/WinRAR (.rar) and proprietary WinZip (.zipx) will result in mark deductions.**

Submit a README text file as described above. The README should inform the marker of the development environment used for completing the assignment,. Remember the marker will be using gcc with the flags -ansi -pedantic -Wall and possibly –lm

.

Also, your README should state any problems you encountered that may help the marker.