COSC2451 Programming Techniques

Assignment 2

Deadline: 11:59 pm Monday of week 12 (Sep 5, 2011)

1. General Information

COSC2451 Programming Techniques has as one of its requirements the completion of a practical work component worth 40% of the overall assessment. This practical work component forms one of two major hurdles, the other being the final examination.

The 40% weighting translates into approximately 60 hours of hands-on programming work for both assignments, and which must be treated as an average commitment. The broad learning objectives are:

- To gain proficiency in C programming to the extent that students gain more than a passing familiarity with the most commonly used and moderately complex features of the language.
- To develop through this exercise, a good understanding and competency of programming techniques such as coding style, presentation, modularity, source code management, and debugging.
- To relate algorithmic design with code implementation and important factors in good software development practice.

In line with such objectives there are two distinct assignments, so that different elements of software development such as programming in C and overall design and implementation are exercised incrementally. Assignment #1 is worth 15% and Assignment #2 is worth 25%.

2. Plagiarism Notice

Plagiarism is a very serious offence. The minimum first time penalty for assignments deemed to be plagiarised is zero marks for that assignment. In the event that a hurdle requirement is not met, this will result in the failure of the course.

Please keep in mind that RMIT uses plagiarism detection software for plagiarism detection and you can expect that all assignments will be tested using this software.

More information about plagiarism can be found here:
http://www.cs.rmit.edu.au/students/integrity/

3. General Requirements

These are general requirements that are applicable for both assignments.

General Requirement #1 – Buffer handling
This requirement relates to how well your programs handle "extra input". To aid you with this requirement, we want you to use the `readRestOfLine()` function provided in the start

up code. Use of this function with `fgets()`. Marks will be deducted for the following buffer-related issues:

- Prompts being printed multiple times or your program "skipping" over prompts because left over input was accepted automatically. These problems are a symptom of not using buffer overflow code often enough.
- Program halting at unexpected times and waiting for the user to hit enter a second time. Calling your buffer clearing code after every input routine without first checking if buffer overflow has occurred causes this.
- Using `fflush(stdin)`. This function is not appropriate for reasons mentioned in the course FAQ on Blackboard: "Course Documents" -> "FAQ" -> "Alternative to non-standard fflush(stdin)"
- Using `rewind(stdin)`. This function is not appropriate as it is not intended to be used for buffer clearing.
- The use of buffer clearing code may have been avoided with `gets()` or `scanf()` for scanning string input. These functions are not safe because they do not check for the amount of input being accepted.
- Using long characters arrays as a sole method of handling buffer overflow. We want you to use the `readRestOfLine()` function.
- Other buffer related problems.

For example, what happens in your program when you try to enter a string of 40 characters when the limit is 20 characters?

General Requirement #2 – Input validation
For functionality that we ask you to implement that relies upon the user entering input, you will need to check the length, range and type of all inputs where applicable.

For example, you will be expected to check the length of strings (e.g., 1-20 characters), the ranges of numeric inputs (e.g., an integer with value 1-7) and the type of data (e.g., is this input numeric?)

For any detected invalid inputs, you are asked to re-prompt for this input - don't truncate extra data or abort the function.

General Requirement #3 – Coding practices
Marks are awarded for good coding practices such as:

- Avoiding global variables.
- Avoiding goto statements.
- Consistent use of spaces or tabs for indentation. We recommend 4 spaces for every level of indentation. Be careful to not mix tabs and spaces. Each "block" of code should be indented one level.
- Keeping line lengths of code to a reasonable maximum such that they fit in the default **xterm** screen width.
- Commenting (including function header comments).
- Complete author identification on all files.
- Appropriate identifier names.
- Avoiding magic numbers.
- Avoiding platform specific code with `system()`.

- Checking the return values of important functions such as `fopen()`, `malloc()` and so on.
- General code readability.

4. Penalties

Marks will be deducted for the following:

- Compile errors and warnings.
- Fatal run-time errors such as segmentation faults, bus errors, infinite loops, etc.
- Files submitted in PC format (instead of Unix/Linux format).
- Missing files (affected components get zero marks).
- Files incorrectly named, or whole directories submitted.
- Not using startup codes or not filling-in the README.txt file.

Programs with compile errors that cannot be easily corrected by the marker will result in a maximum possible score of 40% of the total available marks for the assignment. Any sections of the assignment that cause the program to terminate unexpectedly (i.e., segmentation fault, bus error, infinite loop, etc) will result in a maximum possible score of 40% of the total available marks for those sections. Any sections that cannot be tested as a result of problems in other sections will also receive a maximum possible score of 40%.

It is not possible to resubmit your assignment after the deadline due to mistakes.

5. Assignment #2 Information

Scenario: Christopher, Xiaodong and Quang can't get enough coffee, and they have asked you to implement a system to manage details about Gloria Jean's coffee – their favourite coffee shop. In implementing this system you will be demonstrating your understanding and programming ability, with respect to more advanced C programming principles.

The concepts covered include:

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

Your assignment must compile and execute cleanly on `mekong` in the fashion below:

Compile:
```
[s1234567@mekong] make
```

Execute:
```
[s1234567@mekong] ./gjc [command line args]
```

The start-up source code for `gjc` is 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 to do so (such as a bug identified in the start-up code). You will need to build upon the code that is provided, instead. If you have any concerns about the start-up code, please post your query to the Blackboard discussion board.

6. 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.

Requirement #1 – Command line arguments - 5 marks
The user of your program must be able to execute it by passing in the names of two data files that the system is to use. You need to check that exactly three command-line arguments are entered. Make sure that the menu and submenu files exist already. 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:
```
[s1234567@mekong] ./gjc <menufile> <submenufile>
```

For example:
```
[s1234567@mekong] ./gjc menu.dat submenu.dat
```

Requirement #2 – Load Data - 12 marks
Your program needs to be pre-populated with the data provided in the menu and submenu files whose names are conveyed via the command line. You will need to tokenize this data in order to load it into the system. Use the structure definitions provided in the start up code to store your system data.

Menu file format:
```
[CategoryID]|[CategoryType]|[CategoryName]|[CategoryDescription]
```

Example:
```
C0001|H|Expresso Classics|The traditional expresso favourites
```

Note: Category type is either 'H' or 'C' for hot or cold drinks.

Submenu file format:
```
[ItemID]|[CategoryID]|[ItemName]|[Price1]|[Price2]|[Price3]|[ItemDes
cription]
```

Note: "Price1", "Price2", and "Price3" represent the prices for small, medium, and large drinks respectively.

Example:
```
I0001|C0001|Cappuccino|2.60|3.00|3.40|Espresso and steamed milk
beneath a thick layer of dense velvety smooth milk
```

Each file contains a variable number of records. The menu data is stored in a sorted linked list by category name. For each menu record, a separate linked list for the submenu is stored and sorted by item name. This is a two-dimensional linked list structure.

Please note that you may not assume that the data contained in these files is valid. For example, there may be lines with too many or too few fields, and the data in each field may not be of the correct type, range and/or length. If you find invalid data, you should abort the program with an appropriate error message – please don't attempt to partially load data.

Sample (valid) data files have been provided with the start-up code. Make sure that your program works with at least these files. Additionally, we will be testing your program with invalid data files.

We recommend that you get your program working for the provided (valid) data files first. Then spend any remaining time on your assignment testing for invalid files after you have completed most/all of the remainder of the assignment.

Requirement #3 - Main menu - 5 marks
Your program must display an interactive menu displaying nine options as below.

```
Main Menu:
(1) Hot Drinks Summary
(2) Cold Drinks Summary
(3) Detailed Menu Report
(4) Add Menu Category
(5) Delete Menu Category
(6) Add Menu Item
(7) Delete Menu Item
(8) Save & Exit
(9) Abort
Select your option (1-9):
```

Your program must allow the user to select these options by typing the number and hitting enter. Upon completion of all options except "Save and Exit" and "Abort", the user is returned to the main menu. The behaviours of these options are described in requirements #4 to #11.

Requirement #4 – "Summary Menus" – 4 marks
This requirement is for the "Hot Drinks Summary" and "Cold Drinks Summary" menu options. This option should display a summary of all menu categories and items in the system in a tabular and hierarchical format that are classified as either "hot" or "cold" drinks.
The format of both summaries is the same. Here's an example for the hot drinks summary:

```
Hot Drinks Summary
-------------------
C0001 - Expresso Classics (3 items)
-------------------------------------------------
ID    Name                         Small Med   Large
----- -------------------------  ----- ----- -----
I0004 Cafe Mocha                   $3.30 $3.60 $4.40
I0001 Cappuccino                   $2.60 $3.00 $3.40
I0008 Filtered Coffee              $1.80 $2.30 $2.80

C0003 - Tea & Cocoa (2 items)
-------------------------------------------------
ID    Name                         Small Med   Large
----- -------------------------  ----- ----- -----
I0015 Chai Tea                     $3.00 $3.60 $4.30
I0017 Hot Chocolate                $3.00 $3.50 $4.00
```

The output should be aligned in columns. The output should demonstrate the sorted order of the category names and item names. Pay attention to the column alignment.

Requirement #5 – "Detailed Menu Report" – 4 marks
This option allows the user to generate a detailed report for single menu category.

Example:
```
Detailed Menu Report
--------------------
Enter category id (5 characters): C0001
File C0001.report has been created.
```

The report filename is always in the "[categoryid].report" format. Any existing file with the same name is overwritten. An example is on the next page.

```
Category C0001 - Expresso Classics - Detailed Report
-------------------------------------------------------------
Item ID     : I0004
Item Name   : Cafe Mocha
Prices      : $3.30 $3.60 $4.40
Description : Rich chocolate, espresso, and steamed milk finished
with whipped cream and chocolate sprinkles.

Item ID     : I0005
Item Name   : Short Black
Prices      : $2.40 $2.80 $3.20
Description : Espresso Blend coaxed to yield the essence of fine
coffee. Wonderfully intense and aromatic.
```

Note: The Description field will often span 2-3 lines. It is your job to break the lines at the 78 column mark so that they don't word-wrap beyond the default Unix screen width (80 columns). The line breaks should be done in between words, and not in the middle of words.

Requirement #6 – "Add Menu Category" – 8 marks
This option allows the user to add new menu categories to the system. A new distinct category ID is automatically generated and the user is prompted for information as demonstrated in the example below.

Example:

```
Add Menu Category
-----------------
New category ID is C0015.
Category Name (1-25 characters): Fruit Chillers
(H)ot or (C)old drink?: C
Description (1-250 characters): A 97% fat free non-diary blend of
natural fruit puree and ice.
```

Store the new menu category in the category linked list and maintain the category name in sorted order.

Requirement #7 – "Delete Menu Category" – 6 marks
This option allows the user to remove a menu category and all associated menu items in that category from the system.

Example:

```
Delete Menu Category
--------------------
Warning: All menu item data for a menu category will be
deleted if you proceed.
Menu category ID: C0015
Category "C0015 - Fruit Chillers" deleted from the system.
```

Requirement #8 – "Add Menu Item" – 8 marks
This option allows the user to add new menu items to existing menu categories in the system. The user is prompted for information as demonstrated below. The item ID is automatically generated and must be distinct.

```
Add Menu Item
-------------
Category ID (5 characters): C0015
New item ID is I0035.
Item name (1-25 characters): Strawberry Fruit Chiller
Small Price ($0.05-$9.95): 3.95
Medium Price ($0.05-$9.95): 4.95
Large Price ($0.05-$9.95): 5.95
Description (1-250 characters): A strawberry-flavoured 97% fat
free non-diary blend of natural fruit puree and ice.
```

Store the new menu item in the linked list matching the category ID. Maintain the menu item name in sorted order in the linked list.

Requirement #9 – "Delete Menu Item" – 6 marks
This option allows the user to remove a menu item from a chosen category from the system.

Example:

```
Delete Menu Item
----------------
Category ID (5 characters): C0001
Item ID (5 characters): I0001
Menu item "I0001 - Cappuccino" deleted from system.
```

Requirement #10 – "Save and Exit" – 6 marks
This option saves all information in the system back to the data files in the correct format. The data files are overwritten. Don't forget to delete any dynamically allocated memory. This option then terminates the program.

Requirement #11 – "Abort" – 1 mark
This option should terminate your program. All program data will be lost.

Requirement #12 – Return to menu functionality – 5 marks
This requirement is an extension of the five menu options that require the user to enter input. Your program should allow the user to return to the main menu at any point during these options. The user can do this by simply hitting enter. i.e: An empty line was entered.

Requirement #12 – `makefile` **- 5 marks**
Your program must be compilable using a `makefile`. All compile 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 for examples of `makefiles`.

Requirement #14 – Memory leaks - 5 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. Use `valgrind` to check for memory leaks. Full marks will only be awarded for this requirement if a generated `valgrind` report is clean and reports zero memory leaks and no other memory related problems.

Special note: Debugging symbols (from the `gdb` debugger) included in object and executable files may not be compatible with `valgrind`. For this reason, you may wish to substitute the "`-g`" (debugging symbols) flag in your `makefile` with "`-gstabs`".

Other requirements – 10 marks
You must read "Input Validation", "Buffer Handling" and "Coding conventions/practices" sections of the General Information section.

Input validation will form a key component of allocated marks for all requirements that deal with any form of input.

There will be 5 marks allocated for correct “Buffer handling” and 5 marks allocated for “Coding conventions/practices” throughout your code.

7. Demonstration

One demonstration is required for Assignment #2 to show your progress. This will occur in your scheduled class. Demonstrations will be brief, and you must be ready to demonstrate in a two minute period when it is your turn. Buffer handling and input validation will not be assessed during demonstrations, so you are advised to incorporate these only after you have implemented the demonstration requirements. Coding conventions and practices too will not be scrutinised, but it is recommended that you adhere to such requirements at all times.

Requirement#15 – Demonstration – 10 marks
This demonstration will occur on **Tuesday of week 11 (Aug 30, 2011)** in the class. You will need to demonstrate the following, without, at this stage, requiring to implement validation.

- Ability to compile/execute your program from scratch (1.0 marks).
- Requirements 1-6, 12 and 13 implemented and running (0.5/3.0/0.5/0.5/1.5/1.5/1.0/0.5 marks respectively).

8. Submission

The deadline for this assignment is **11: 59 pm Monday of week 12 (Sep 5, 2011).**

For assignment #2, you need to submit 8 files:

gjc.c
This file will include your main function.

gjc.h
Header file for gjc.c.

gjc_options.c
This file contains functions for each of the eight major menu options.

gjc_options.h
Header file for gjc_options.c.

gjc_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.

gjc_utility.h
Header file for gjc_utility.c.

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

gjc_readme.txt
This file will contain important additional information about your program that you wish your marker to see. Examples include incomplete functionality, bugs, assumptions and assignment release permission. It is recommended that you ask about any assumptions you want to make on the Blackboard discussion board first before you make them.

Submission instructions:

- Create a zip archive using the following command in `mekong`:
  ```
  [s1234567@mekong] make archive
  ```
  This command would generate a zip file named "username-a2.zip" with your username substituted for "username".
- Submit the zip file to Blackboard before the deadline.
- Submit SoA to the Drop Box in front of room 1.4.04 on the next day.

Late submission of assignments will be penalized as follows:

- For assignments 1 to 5 days late, a penalty of 10% (of total available marks) per day.
- For assignments more than 5 days late, a penalty of 100% will apply