COSC1076/2207 Advanced Programming Techniques – Assignment General Requirements

These are general requirements that are applicable for both assignments.

General Requirement #1 – Functional Abstraction

This requirement relates to good program design.

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 process of 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.

General Requirement #2 – 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()` is demonstrated on Blackboard in this location: "Course Documents" -> "Function Examples" -> "Input Validation Examples". 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.

General Requirement #3 – 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?). Also, some data types can only be allowed to have a limited set of values, for example days of the week or months of the year.

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

Further, you are required to demonstrate correct use of the standard library functions:

```
    long strtol(const char *restrict str, char **restrict
    endptr, int base);
```

and

```
    char *strtok(char *restrict s1, const char *restrict s2);
```

where appropriate in your code. The use of alternate methods to achieve similar functionality (such as the use of the scanf family of functions or atoi()) will result in a deduction. Please note that “correct use” includes appropriate error handling. For example, your error messages displayed to the user should be self-explanatory.

General Requirement #4 – Coding conventions/practices

Marks are awarded for good coding conventions/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 (80 characters wide).
- 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.

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 format). This can occur when transferring your files from home (PC) to saturn or jupiter (coreteaching01 or coreteaching02) (Unix). You can remove extra end-of-line characters from PC formatted files using the **dos2unix** command or **tr**. Please see the UNIX Survival Guide for details on how to do this.
- Missing files (affected components get zero marks).
- Files incorrectly named, or whole directories submitted.
- Not using startup code or not filling-in the readme 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.

General Submission Information

Weblearn is used for submission of all assignments. Assignments are not accepted by any other means (including email).

You will be informed via the Blackboard discussion board when Weblearn is enabled. If you are unsure how to use Weblearn, please ask your lab assistant during a COSC1076 lab. You are required to submit multiple increments. Don’t panic and leave it to the last minute!

With Weblearn, it is important to note that any new submission completely replaces the previous one. Therefore you need to re-submit every file each time a submission is made. No special consideration will be given for people who submit incorrectly.

Requests for extension must be directed to the lecturer and must be substantiated with appropriate documentary evidence. A special consideration form (available from the CS&IT office) must be completed and submitted with the documentary evidence. A common excuse for requests for extensions is “My hard disk crashed and I lost all my work.” This won’t be accepted and we advise you to keep an up to date backup of all relevant source files.

When/how do I get my marks?

Assignment marks will be made available within 10 working days of the final submission deadline. An announcement will be made on the Blackboard discussion board when marks are available. An announcement will also be made with regards to what you need to do if there are any mistakes in your marks.

Help!

Please utilise the following with regards to getting help for your assignments:

- For general assignment questions, please use the Blackboard discussion board. That way all students can see all questions once.
- Please do not post large pieces of code to the Blackboard discussion board for plagiarism reasons. Show your code to your lab assistant instead.
- You are generally welcome to bring assignment-related questions to tutorial and lab classes.
- If you are having problems that may prevent you from completing your assignment on time, please talk to someone as early as possible.
- Consultation times are available.

If you find any problems with the assignment specifications, please post your queries to the Blackboard discussion board.