Curtin University — Department of Computing
Bentley Campus

Unix & C Programming (COMP1000)

Semester 2, 2020

Assignment 2

Due: Monday 26 October at 9.00 am (UTC +8)

Weight: 35% of the unit mark.

1 Introduction

Your task for this assignment is to design, code (in C89 version of C) and test a program. In summary, your program will:

- Be able to read the source code from a file. (can be .txt or .c file)
- Implement struct and linkedlist to store the relevant data.
- Able to visualize the basic bracket-matching algorithm.
- Manipulate multiple strings (array of char) to accomplish the task.
- Write a suitable makefile, with a conditional compilation.

2 Code Design

You must comment your code sufficiently in a way that we understand the overall design of your program. Remember that you cannot use // to comment since it is C99-specific syntax. You can only use /**/ syntax.

Your code should be structured in a way that each file has a clear scope and goal. For example, main.c should only contain a main function, color.c should contain functions that handle and manipulate color, and so on. Basically, functions should be located on reasonably appropriate files and you will link them when you write the makefile. DO NOT put everything together into one single source code file. Make sure you use the header files and header guard correctly.

Make sure you free all the memories allocated by the malloc() function. Use valgrind to detect any memory leak and fix them. Memory leaks might not break your program, but penalty will still be applied if there is any.

Please be aware of our coding standard (can be found on Blackboard under Resources) Violation of the coding standard will result in penalty on the assignment. Keep in mind

that it is possible to fail the assignment with a fully-working program that is written messily, has no clear structure, full of memory leaks, and violates many of the coding standards. One purpose of this unit is to teach you good habits in programming.

3 Academic Integrity

This is an assessable task, and as such there are strict rules. You must not ask for or accept help from anyone else on completing the tasks. You must not show your work to another student enrolled in this unit who might gain unfair advantage from it. These actions are considered plagiarism or collusion.

Do NOT just copy some C source codes from internet resources. If you get caught, it will be considered plagiarism. Always reference your sources. Be aware that if significant portions of the assignment are not your own work (even if referenced), there may be penalties. If in doubt, ask!

Staff can provide assistance in helping you understand the unit material in general, but nobody is allowed to help you solve the specific problems posed in this document. The purpose of the assignment is for you to solve them on your own, so that you learn from it. Please see Curtins Academic Integrity website for information on academic misconduct (which includes plagiarism and collusion).

The unit coordinator may require you to attend quick interview to answer questions about any piece of written work submitted in this unit. Your response(s) may be referred to as evidence in an Academic Misconduct inquiry. In addition, your assignment submission maybe analysed by systems to detect plagiarism and/or collusion.

4 Task Details

You will write a program to perform bracket-balance-checking from a source code file.

4.1 Quick Preview

First of all, please watch both videos on the Assignment link on Blackboard (Assignment Supplementary 1 & 2). Like the first assignment, these videos provides the summary of what we expect from your assignment attempt. I also added few tips and advices as a bonus. Further details on the specification will be explained on the oncoming sections.

4.2 Command Line Arguments

You can use any name for your executable file. Your executable should accept one (1) command-line parameter:

- The file name of the source code to be investigated. This file can be in .c or .txt format.

For example, you can run the program like this:

```
root $> ./assignment2 input.txt
```

OR

```
root $> ./assignment2 input.c
```

With this parameter, the program will check the bracket balance on the source code inside the input file. If any imbalance is encountered, the program will show appropriate error message and stop the program.

4.3 Expected Output & Algorithm

Note: Please watch the first supplementary video for better intuition of the program.

Once you run the program, it should clear the terminal screen and print the whole source code on the terminal. Similar to the first assignment, you will have a moving arrow pointing from the first character to the last character in the file. The purpose is to keep track of various brackets in the file and highlight them when they are properly matched. There are 4 types of bracket you need to check:

- Round brackets – ()
- Square brackets – []
- Curly brackets – {}
- Angled brackets – <>

You can choose which highlight color for each bracket type.

If the input file contains balanced brackets, then the program will highlight all the brackets and showing a message indicating no error encountered. For example:

```
#include<stdio.h>

int main()
{
        printf{"hello world\n"};
        return 0;
}

All Good
```

However, if there are imbalanced brackets, then the program has to be stopped and shows appropriate message. There are 3 scenarios you have to cover:

- The most recent closed bracket does not match the most recent open bracket. In this case, you have to show an error message below the arrow indicating the expected correct closed bracket. For example:

```
#include<stdio.h>

int main(]
          ^
          )' expected
{
        printf{"hello world\n"};
        return 0;
}
```

- A closed bracket is encountered when there is no open bracket to be matched. In this case, the error message indicates the expected open bracket prior to the closed bracket. For example:

```
#include<stdio.h>

int main))
        ^
         '(' expected before this
{
        printf{"hello world\n"};
        return 0;
}
```

- The end of the source code is reached when there is at least one open brackets to be matched. In this case, the error message indicates:

```
#include<stdio.h>

int main()
{
        printf{"hello world\n"};
        return 0;
^
'}' expected before End of Code
```

4.4 newSleep() Function

As mentioned in the supplementary video, you will not use the sleep() function from assignment 1 anymore. Instead, you will use the newSleep() function which has been pre-written for you. The header of the function looks like:

void newSleep(float seconds);

You can now put the program to sleep with real number instead of integer so that you can iterate every character faster. For example, the demonstration in the video sleeps for 0.4 seconds for every iteration of the arrow. The file "newSleep.h" and "newSleep.o" are provided for you. You need to use and include them properly on the makefile to be able to use the newSleep() function.

4.5 File I/O

Since you need to read the content of the input file, you need to properly open and read the content of the file, including all the error handlings (refer to lecture 5). For simplicity, you can assume the input source code file **will not exceed 50 lines**. You are allowed to read the file multiple times if needed. (e.g for counting the line numbers). You have to find a way to store the source code from the input file in your program.

4.6 Struct & Linked List

You are required to use struct and linked list to keep track of the brackets you found on the input file. You have to use malloc() and free() for every node struct you create. As mentioned in the supplementary video, you will use the stack data structure with linked list to update the list as you encounter every occurrence of the brackets.

At the very least, you need to write these linked list functions:

- Create a new linked list.
- Insert new node at the start of the linked list.

- Remove the first node of the linked list. (You might want to perform bracket matching verification here before using free() function)
- Print the content of all the data from all nodes in the linked list. (function pointer parameter is recommended here to assist the printing)

4.7 Conditional Compilation: Printing the Contents of the Linked List

Please create a conditional compilation called "STACK" in your makefile. When this is defined, your program will print out the content of the linked list below the printed source code. Basically, you will print the list of the brackets stored in the linked list starting from the first node (from top of the stack) separated by "\n" (new line). This list will be updated every time the arrow encounter any bracket. If it encounters open bracket, the list will grow. On the other hand, if it encounters closed bracket, the list will shrink. When the program stops due to imbalance brackets, leave the list as it is. Since you have to traverse the linked list to print the data, I would suggest to create a simple function to print the content of each node in the linked list. Then, you can pass the function pointer to the printLinkedList function.

4.8 Makefile

You should write the makefile according to the format explained in the lecture and practical. It should include the Make variables, appropriate flags (-ansi -pedantic -Werror -Wall), appropriate targets, clean rule, and the conditional compilation. We will compile your program through the makefile. **If the makefile is missing, we will assume the program cannot be compiled**, and you will lose marks.

5 Final Check & Submission

After you complete your assignment, please make sure it can be compiled and run on our Linux lab environment. If you do your assignment on other environments (e.g on Windows operating system), then it is your responsibility to ensure that it works on the lab environment. In the case of incompatibility, it will be considered not working and some penalties will apply. You have to submit a tarball (see practical worksheet for the tarball creation instruction) containing:

- **Declaration of Originality Form** — Please fill this form out digitally and submit it with your assignment. Submissions without a declaration of originality will **not be marked**.
- **Your essential assignment files** — Submit all .c and .h files along with your makefile. Please do not submit any executable or object (.o) files (except newSleep.o).

The name of the tarball should be in the format of:

$$\langle student - ID \rangle\_\langle full - name \rangle\_Assignment2.tar.gz$$

Example: 12345678_Antoni-Liang_Assignment2.tar.gz

Please make sure your submission is complete and not corrupted. You can re-download the submission and check if you can compile and run the program again. Corrupted submissions will not receive marks. You may submit multiple times, but only your latest submission will be marked.

5.1 Demonstration

At the last practical of the semester, you are given a chance to briefly demonstrate your assignment. This will not contribute to the mark, but it might give us a better understanding of how your program works, which can assist us to do the marking more effectively and clearly. There is no penalty for skipping the demonstration, but you might miss the precious chance to justify your work.

End of Assignment 2