

# Assignment

**Due:** Monday 21 Sept at 9.00 am (UTC +8)

**Weight:** 15% 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:

- Read the sentence, keywords, and color choices from command line argument.
- Able to visualize the basic pattern-searching-and-replacement algorithm.
- Manipulate 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 Curtin's 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 be writing a program to perform pattern-matching replacements on a string. The details of how your program is expected to run are detailed in the follow section.

#### 4.1 Quick Preview

First of all, please watch a short video demo on the Assignment link on Blackboard (demo.mp4). This video demonstrates what we expect your program can do. 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 five (5) command-line parameters:

- The full string sentence. (Use double quote to cover the sentence) . The sentence can contain both lower and upper cases.

- The word pattern to be replaced. (can use double quote as well)
- The word replacement. (can use double quote as well)
- The color of the word to be replaced. (case insensitive, more details on later section)
- The color of the replacement word. (case insensitive, more details on later section)

This is one example:

```
root $> ./demoStringReplace "this tea is good!" tea biscuit red green
```

With the above parameters, the program will replace the string “this tea is good!” into “this biscuit is good!”. Keep in mind that the pattern matching is **case insensitive**.

### 4.3 Expected Output & Algorithm

Once you run the program, it should clear the terminal screen (see the example demo-ClearAndSleep.c) and show the word pattern to be replaced:

```
replace: tea -> biscuit
```

It is then followed by the full sentence with a simple arrow pointing at the first character. The purpose is to pattern-check the character one by one (and show the process), and replace the content when there is a pattern match. You must print the following strings under the arrow:

*Note: The following colours are for illustrative purposes. The actual colours are to be determined by your own preferences and/or the command line arguments*

- **no match** — when there is no matching pattern. i.e.

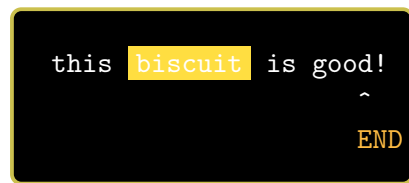
```
this tea is good!  
^  
no match
```

- **replaced** — when there is a matching pattern, the program immediately replaces the word with the highlight color specified by your last command line argument. i.e.

```
this biscuit is good!  
^  
replaced
```

- **END** — when you reach the location at which it is no longer possible to find a match. For example, if the program is searching for “tea”, which contains three

characters, the searching will END when there are two characters left in the sentence (ignoring the null terminator). i.e.



```
this biscuit is good!  
      ^  
      END
```

#### 4.3.1 Program Speed & Sleep

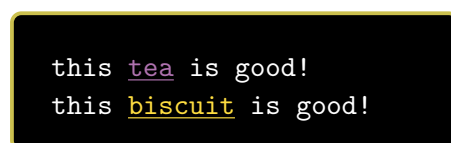
If you run the algorithm without a means of delaying the display, it will execute faster than is reasonable to follow. For this reason **there must be a one second delay** between each character being checked. For more details, see the `demoClearAndSleep.c` file provided.

#### 4.3.2 Built-in Functions

C provides a number of functions that could make this assignment trivial (such as `strcmp()` and `strstr()`). For this assignment you are **NOT allowed** to use these functions. You must write your own set of functions for comparing and manipulating string variables. In addition, remember that the matching should not be sensitive to case.

#### 4.3.3 Final Display

After the pattern searching is complete, your program should clear the screen a final time and print *both* original and new sentences. The output should also underline the matched pattern with the color based on the fourth and fifth command line arguments. Note that this will likely require your program to keep track of the locations where all pattern matching occurs. Example output may look like this:



```
this tea is good!  
this biscuit is good!
```

For the sake of simplicity, you may assume that the length of the sentence will not be so long as to cause it to wrap onto the next line of the terminal.

### 4.4 Font Color Modification

You are to write and call functions that utilize ANSI color codes to modify font colors when using `printf()`. This may require you to do some research on “ANSI color in C”. You are to write functions that cover at least six (6) distinct font colors (which should also include underlining and highlighting). Example colors are: red, green, blue, yellow,

magenta, cyan. Specific prototypes for these functions will not be mandated, but they should still be readable and work as described in section 4.3.

## 4.5 Conditional Compilation: Printing Index

The program should also allow for conditional compilation with a flag called “INDEX” in your makefile. When this is defined, your program will print out the index location of the string where the pattern matches on the original sentence. This index list will be updated in real time as you find more occurrences of pattern matching. You can print these indices on the new line of the terminal screen.

## 4.6 Makefile

You should write the makefile according to the format explained in the lecture and practical. It should include the Make variables, appropriate flags, 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.

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

*$\langle student - ID \rangle\_ \langle full - name \rangle\_ Assignment1.tar.gz$*

Example: 12345678\_Antoni-Liang\_Assignment1.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.

**End of Assignment**