

**CSCI 4210 — Operating Systems**  
**Homework 1 (document version 1.0)**  
**Files, Strings, and Memory Allocation in C**

## Overview

- This homework is due by 11:59:59 PM on Tuesday, June 4, 2019.
- This homework is to be completed **individually**. Do not share your code with anyone else.
- You **must** use C for this homework assignment, and your code **must** successfully compile via `gcc` with absolutely no warning messages when the `-Wall` (i.e., warn all) compiler option is used. We will also use `-Werror`, which will treat all warnings as critical errors.
- Your code **must** successfully compile and run on Submittity, which uses Ubuntu v18.04.2 LTS. Note that the `gcc` compiler is version 7.4.0 (Ubuntu 7.4.0-1ubuntu1~18.04).

## Homework specifications

In this first homework, you will use C to implement a rudimentary cache of words found in a given text file. For this assignment, the cache will simply be a hash table of a fixed size. This hash table will handle collisions by simply replacing the existing word with the new word.

The goal of this assignment is to become more comfortable programming in C on Linux, in particular handling strings, working with pointers, and dynamically allocating (and re-allocating) memory. **And to emphasize and master the use of pointers, you are not allowed to use square brackets in your code!** If a '[' or ']' character is anywhere in your code, including within comments, you will receive a zero for this assignment. (Ouch!)

## Command-line arguments and memory allocation

The first command-line argument specifies the size of the cache, which therefore indicates the size of the dynamically allocated array that you must create. To create the array, use `calloc()` (rather than `malloc()`) to create an array of character pointers. Use `atoi()` for the conversion.

Your program must then open and read the regular file specified as the second command-line argument. Your program must parse all words from the given file (if any), determine the cache array index for each word (in the order each word is encountered), then store the word in the cache, replacing any existing word if a collision occurs.

To read in each word from the given file, you should use a dynamically allocated character array of size 128. In other words, you can assume that each word is no more than 127 characters long (since you want to save space for the end-of-string '\0' character).

Initially, your cache is empty, meaning it is an array of NULL pointers, since `calloc()` will zero out the allocated memory. Storing each word therefore also requires dynamic memory allocation. For this, use `calloc()` if the cache array slot is empty; otherwise, to replace an existing word, use `realloc()`. Be sure to calculate the number of bytes to allocate as the length of the given word plus one, since strings in C are implemented as `char` arrays that end with a '\0' character.

## What is a word and how do you “hash” it?

For this assignment, words are defined as containing only alphanumeric characters and being at least two characters in length. Further, words are case sensitive (e.g., `Lion` is different than `lion`).

To determine the cache array index for a given word (i.e., to properly “hash” the word), write a separate function to add up the sum of each ASCII character in the given word as an `int` variable, then use the “mod” operator to determine the remainder after dividing by the cache array size.

As an example, the valid word `Meme` consists of four ASCII characters, which sum  $77 + 101 + 109 + 101 = 388$ . If the cache array size was 17, for example, then the array index for `Meme` would be the remainder of  $388/17$  or 14.

## Error handling

If improper command-line arguments are given, report an error message to `stderr` and abort further program execution. In general, if an error is encountered, display a meaningful error message on `stderr` by using either `perror()` or `fprintf()`, then aborting further program execution.

Error messages must be one line only and use the following format:

```
ERROR: <error-text-here>
```

## Required Output

When you execute your program, you must display a line of output for each word that you encounter. And for each word, display the cache array index and whether you called `calloc()` or `realloc()`.

As an example, below is sample output, showing the format you must follow:

```
Word "Once" ==> 15 (calloc)
Word "when" ==> 9 (calloc)
Word "Lion" ==> 11 (calloc)
Word "was" ==> 8 (calloc)
Word "asleep" ==> 5 (calloc)
Word "little" ==> 8 (realloc)
Word "Mouse" ==> 11 (realloc)
Word "began" ==> 16 (calloc)
...
```

Further, when you have processed the entire file, show the contents of the cache by displaying a line of output for each non-empty entry in the cache, using the following format:

```
Cache index 5 ==> "asleep"
Cache index 8 ==> "little"
Cache index 9 ==> "when"
Cache index 11 ==> "Mouse"
Cache index 15 ==> "Once"
Cache index 16 ==> "began"
```

## Hints

Consider using `fopen()`, `fgets()`, `fgetc()`, `isalnum()`, `strlen()`, `strcpy()`, `strncpy()`, and other such string and character functions for this assignment. Be sure to check out the details of each function by reviewing the corresponding `man` pages from the terminal.

Also, be sure to use `free()` to ensure all dynamically allocated memory is properly deallocated. Use `valgrind` to verify that there are no memory leaks.

## Submission Instructions

To submit your assignment (and also perform final testing of your code), please use Submittity, the homework submission server.

Note that this assignment will be available on Submittity a minimum of three days before the due date. Please do not ask when Submittity will be available, as you should first perform adequate testing on your own Ubuntu platform.

That said, to make sure that your program does execute properly everywhere, including Submittity, use the techniques below.

First, as discussed in class (on 5/20), use the `DEBUG_MODE` technique to make sure you do not submit any debugging code. Here is an example:

```
#ifdef DEBUG_MODE
    printf( "the value of x is %d\n", x );
    printf( "the value of q is %d\n", q );
    printf( "why is my program crashing here?!" );
    printf( "aaaaaaaaaaaaagggggggghhhh!" );
#endif
```

And to compile this code in “debug” mode, use the `-D` flag as follows:

```
bash$ gcc -Wall -Werror -D DEBUG_MODE hw1.c
```

Second, as discussed in class (on 5/23), output to standard output (`stdout`) is buffered. To disable buffered output for grading on Submittity, use `setvbuf()` as follows:

```
setvbuf( stdout, NULL, _IONBF, 0 );
```

You would not generally do this in practice, as this can substantially slow down your program, but to ensure good results on Submittity, this is a good technique to use.