CSCI 2500 — Computer Organization
Homework 2 (document version 1.0)
Files, Memory, and Histograms in C

- This homework is due in Submitty by 11:59PM EST on Thursday, February 10, 2022
- You can use at most three late days on this assignment
- This homework is to be done individually, so **do not share your code with anyone else**
- You **must** use C for this assignment, and all submitted code **must** successfully compile via `gcc` with 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
- In our `storage.c` example from January 27, we discussed the use of dynamic memory allocation instead of a variable-length array (VLA); you are not allowed to use VLAs and to enforce this, we will use the `-Wvla` flag of `gcc`
- All submitted code **must** successfully compile and run on Submitty, which uses Ubuntu v20.04.3 LTS and `gcc` version 9.3.0 (`Ubuntu 9.3.0-17ubuntu1~20.04`)

Homework specifications

In this second homework, you will implement two histogram generator programs using C. For both programs, use `man` pages and the posted code examples to help you write code that is correct, complete (i.e., checks for errors), and also reusable.

Dynamic memory allocation is required for both programs you will write. Specifically, you are required to use `calloc()` and `free()` as part of your solution for each.

Further, do not hard-code any array lengths in your code; instead, use dynamic memory allocation.

Program execution requirements

For both Part I and Part II below, your program must return either `EXIT_SUCCESS` or `EXIT_FAILURE`, which you can verify via `echo $?` in the terminal.

In general, if an error is encountered, display a meaningful error message on `stderr` by using either `perror()` or `fprintf( stderr, "..." )`, then aborting further program execution. Only use `perror()` if the given library or system call sets the global `errno` variable.

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

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

Part I — Character-based histogram generator

For Part I of this homework, use C to display a text-based histogram of the frequencies of different characters encountered on the `stdin` input stream. Ignore special non-printable characters (e.g., newlines) by only accumulating counts for printable ASCII characters, i.e., characters in the range from `' '` to `'~'` (space to tilde). Use the `isprint()` function from `ctype.h` here.

The order in which your histogram values must be displayed follows the ASCII table. Example program execution is shown below with sample input redirected in as `stdin` via `<` in the shell.

```
bash$ cat in.txt
MEME 8888888& !~!
bash$ ./a.out < in.txt
 : 2
!: 2
&: 1
8: 7
E: 2
M: 2
~: 1
```

Next, extend your program to support an optional `-b` flag that shows a bar of `'#'` characters instead of a count. Note that this option must come before the input file. Repeating the above example, program execution using this flag would be as follows:

```
bash$ cat in.txt
MEME 8888888& !~!
bash$ ./a.out -b < in.txt
 : ##
!: ##
&: #
8: #######
E: ##
M: ##
~: #
```

As a hint, character and string functions are organized in the `ctype.h` and `string.h` libraries.

Compilation requirements

Place your code in `hw2-part1.c` and feel free to also include one or more `.h` header files. Your code must successfully compile using the following command:

```
bash$ gcc -Wall -Werror -Wvla hw2-part1.c
```

Part II — File-based histogram generator

For Part II of this homework, you will use C to implement a more general-purpose histogram generator based on two input files. More specifically, your program uses the first input file to determine what data are stored in the second input file. Next, your program displays the data from the second input file in the form of a text-based histogram.

Input files

The two required input files are given as command-line arguments, which remember will be `argv[1]` and `argv[2]`. Attempt to open the first file, which you will treat as a configuration file with the following format.

The first line of this file specifies the *title* of the histogram (e.g., `"Number of Novels Authored"`). The maximum length of the title is 32 characters; truncate any titles longer than this maximum.

The second line specifies the *number of columns* in the given data. This value should be a positive integer and must be used for your dynamic memory allocation. There is no pre-defined maximum number of columns.

The third line specifies the *data point character* to use when you construct and display the histogram, i.e., what character you will display for each unit of data. Example characters include `'*'` and `'#'` (but any printable character is valid).

The fourth and subsequent lines of the input file specify the *column headers* (e.g., `"Author name"`). The number of lines here depends on the number of columns specified earlier. The maximum length of each column header is 16 characters; truncate any column headers longer than this maximum.

A sample valid configuration file is shown below:

```
bash$ cat authors-config.dat
Number of Novels Authored
3
#
Author name
Initials
Number of novels
```

If a column header starts with the string `"Number"` (e.g., `"Number of novels"`), then interpret the data in that column as an `int` variable; otherwise, treat the data as a character string.

The data are stored in the second file in a line-based format. Continuing the above example, a snippet of the data file is as follows:

```
Jane Austen
JA
6
Charles Dickens
CD
20
...
```

Be sure to validate all inputs, displaying error messages on `stderr` as appropriate.

Required program output

Given the configuration file on the previous page, example program execution is shown below. Be sure to match the output formatting exactly as shown. Specifically, the width W of each column is the width of the longest data element plus two to account for a leading and trailing space character. The number of hyphen `'-'` characters to use for each column is also the calculated value W.

Convert the title to uppercase and format as shown below with three leading spaces.

For each column, character strings are right-justified, including column headers, whereas numeric data are left-justified and printed as a histogram.

Note that long lines have been abbreviated in the example below.

```
   === NUMBER OF NOVELS AUTHORED ===

         Author Name | Initials | Number of novels
---------------------|----------|---------------------------------------
         Jane Austen |       JA | ######
     Charles Dickens |       CD | ####################
    Ernest Hemingway |       EH | #########
        Jack Kerouac |       JK | ######################
 F. Scott Fitzgerald |      FSF | ########
        Mary Shelley |       MS | #######
    Charlotte Bronte |       CB | #####
          Mark Twain |       MT | ###########
     Agatha Christie |       AC | ############ ... ####################
         Ian Fleming |       IF | ##############
        J.K. Rowling |      JKR | ##############
        Stephen King |       SK | ############ ... ###########
         Oscar Wilde |       OW | #
---------------------|----------|---------------------------------------
```

Use the formatting features of `printf()` to the extent that you can.

Compilation requirements

Place your code in `hw2-part2.c` and feel free to also include one or more `.h` header files. Your code must successfully compile using the following command:

```
bash$ gcc -Wall -Werror -Wvla hw2-part2.c
```

Submission instructions

Before you submit your code, be sure that you have clearly commented your code—this should not be an afterthought! Further, your code should have a clear and logical organization. In general, each function should easily fit within a single window and have a clear (and clearly documented) purpose; if not, separate into multiple functions. Variable names and function names should be intuitive and meaningful. Be consistent in your indentation.

To submit your assignment (and also perform final testing of your code), please use Submitty.

Note that this assignment will be available on Submitty a minimum of three days before the due date. There will be hidden test cases.

Also note that output to standard output (`stdout`) is buffered. To disable buffered output on Submitty, 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 you see output even on a segmentation fault on Submitty, this is a good technique to use.