Homework 2

Due Thursday, September 20 by 10pm

Handout 4
CSCI 334: Spring 2018

Turn-In Instructions

For this assignment, create one separate source code file for each question (e.g., `q1.c`).

Supply a `Makefile` (30 points) with one rule per homework assignment. The naming convention for targets should be the name of the source file without the `.c` extention. For example, `q1.c` should compile to `q1`. You must also provide an `all` target so that your assignment can be compiled with `make all`.

For full credit, be sure that your code compiles without emitting warnings even when using the `-Wall` flag. Note that if you use your own computer to do this assignment, you should check your assignment using a lab machine before submitting, since different compiler versions do not always behave the same way. The final authority will be the lab environment.

Turn in your work using the Github repository assigned to you. The name of the Github repository will have the form `cs334hw2_<your user name>`. For example, my repository would be `cs334hw2_dbarowy`. You should have received an invite to commit in the repository in your email. If you did not receive this email, please contact me right away!

Honor code: You may collaborate with one or more people on this assignment, but you may not write code together. All submitted work must be your own original work. If you work with a partner, please submit a `collaborators.txt` file that includes their names.

This assignment is due on Thursday, September 20 by 10pm.

Unix Accounts

We will be working on the Unix lab computers throughout the semester. **If you have not used these machines before or don't remember your password, please see Mary Bailey in TCL 312 to obtain a password and verify that you can log in.**
I encourage you to work in the Unix lab whenever you like, but also keep in mind that you can `ssh` to our computers from anywhere on campus. For example, if your username is `bcool`, you can connect to `lohani` on the command line by typing: `ssh bcool@lohani.cs.williams.edu`

Reading

1. **(Required)** Read "Memory Management in C".

Problems

Q1. (*10 points*) ..... Allocating a zeroed array

Write a function that allocates an `int` array of size n, fills the array with the value `0`, and returns a pointer to the value. `main` should obtain n from the command line. Demonstrate that you've correctly implemented your function by printing all of the values in the calling function. Be sure to correctly manage the lifetime of the array. In other words, given the following starter code, fill in the `???` sections of the implementation.

```
??? zero_fill(int n) {
    // ???
}

int main(int argc, char **argv) {
    if (argc != 2) {
       printf("Usage: p1 <n>\n");
       return 1;
    }

    int n = atoi(argv[1]);

    ??? = zero_fill(n);

    for(int i = 0; i < n; i++) {
        printf("%d\n", ???);
    }

    ???

    return 0;
}
```

Supply your answer as a C source code program called `q1.c`.

Q2. (*10 points*) ..... Passing an array

Rewrite your `zero_fill` function from Question 1 so that allocation of the array occurs in the `main` function instead. You will need to think carefully about `zero_fill`'s parameters; they will need to be changed. `zero_fill` should also return `void`. Be sure that the program behaves exactly the same way as the program you wrote for Question 1.

Supply your answer as a C source code program called `q2.c`.

Q3. (*10 points*) ..... Allocation

Write code to perform the following. Refer to `rand 3 man` for documentation on sampling a random `int`.

(a) Generate a random `int` and copy it to every element of an array of length 10. The array should have *automatic* storage duration. Print the array.

(b) Generate a random `int` and copy it to every element of an array of length 10. The array should have *allocated* storage duration. Print the array. Be sure to deallocate when done.

(c) Generate a random C string (a `char *`) and copy it to every element of an array of length 10. Make sure that every element is a *copy of the string, not a copy of the pointer*. The array should have *automatic* storage duration. Print the array.

(d) Generate a random C string (a `char *`) and copy it to every element of an array of length 10. Make sure that every element is a *copy of the string, not a copy of the pointer.* The array should have *manual* storage duration. Print the array. Be sure to deallocate when done.

Supply your answers as a C source code programs called `q3a.c`, `q3b.c`, `q3c.c`, and `q3d.c`.

Q4. (*15 points*) ..... File reading

`fscanf` is a function that simultaneously lets a C programmer read from a file and convert the bytes read into usable values. In a way, it's like the inverse of `printf`. Instead of printing a string using formatted values from a set of variables, `fscanf` reads a formatted string and converts elements of the string into variables.

For example, given a text file called `file.txt`, where each line contains *up to three* space-delimited words,

```
aardvark beaver chicken
dog elephant falcon
gorilla hamster ibis
jaguar kangaroo lemur
manatee nuthatch oriole
porcupine quagga rabbit
snake tiger urubu
vicuna whale xerus
yak zebrafish
```

the following program

```
FILE *f = fopen("file.txt", "r");
char word[100];
fscanf(f, "%s", word);
printf("%s", word);
fclose(f);
```

will read the first word and print: `aardvark`

Write a program that reads an entire file conforming to the above format, word by word, and prints it back out, one word per line. The name of the file to open should be a command line parameter to the program. You may assume that no word is longer than 99 characters.

Be sure to look up the defintion for `fscanf`, as well as functions you do not know, like `fopen` and `fclose`.

Supply your answer as a C source code program called `q4.c`.

Q5. (*15 points*) ..... File processing

Write a program that reads an entire file conforming to the same format as in Question 4, word by word, into an array. The array should then be printed back out, one word per line. The order that words are printed should be the same as the order found in the text file. The array's length should depend *only on the number of words in the file*, and the name of the file to open should be a command line parameter to the program. Again, you may assume that no word is longer than 99 characters in length.

If you allocate any memory dynamically (i.e., you use `malloc`), be sure to deallocate afterward. You may also find `rewind` to be a useful function. Finally, remember to ensure that all of your strings are correctly `NULL`-terminated. If you see odd characters (e.g., `?`) appear when you print the strings, you are probably not terminating the string correctly. `memset` may be used to initialize a buffer to zero (i.e., all `NULL`s), which sometimes helps with termination.

Supply your answer as a C source code program called `q5.c`.

Q6. (*0 points*) ..... Optional Feedback

How hard was this assignment on a scale of 1 to 5? (where 1 = "very easy" and 5 = "very hard")

Do you have any additional comments or feedback that you would like me to know?

Please supply your answer as a `feedback.txt` file.

Q7. (*1 point*) ..... Bonus

Does the reading "Memory Management in C" have any errors? One bonus point will be awarded for every verified problem that you find and report.

Submit as a text file called `errors.txt`.