CSE 1320: Intermediate Programming
University of Texas at Arlington
Spring 2021
Dr. Alex Dillhoff

Assignment 4

This assignment focuses on using command line arguments, structs, typedefs, and file I/O.

1. Create a program that copies the data from one file to another while converting all characters matching the **search key** with that of the **replace key**. Your program should accept four arguments:

   (a) input filename
   (b) output filename
   (c) search character
   (d) replace character

   - Warn the user if the input file does not exist.
   - Your program should check to make sure the output file does not already exist. If it does, print `"DESTINATION FILE EXISTS."` to `stdout`.
   - Print the number of characters changed to `stdout` using the format string `"%d characters changed."`
   - The search and replace characters are case-sensitive.
   - Save your code as `search_and_replace.c`.

   Example Run

   ```
   $ ./a.out input.txt output.txt A d
   15 characters changed.
   ```

2. Create a program that reads a file of 2D coordinates and calculates the bounding box and center of the bounding box. The bounding box is defined as the minimum area that fully encompasses all the coordinates. The center of that bounding box is calculated by taking the mean of the bounding box coordinates: $(\frac{x_1+x_2}{2}, \frac{y_1+y_2}{2})$.

   - Use file `keypoints.txt` for an example of what the input data will look like.
   - If the input file cannot be opened, warn the user by printing `"CANNOT OPEN FILE."` to `stdout`.
   - Use `typedef` to create a type `vector2f` that defines an array of 2 `float` values. Use this type when working with coordinate data in your program.
   - Print the resulting bounding box and center to `stdout` using the format string `%.2f,%.2f,%.2f,%.2f,%.2f,%.2f`, following the pattern $x_{min}, y_{min}, x_{max}, y_{max}, x_{center}, y_{center}$.

- Save your code as `bounding_box.c`.

Example Run

```
$ ./a.out keypoints.txt
0.00,0.00,100.00,100.00,50.00,50.00
```

3. Create a program that parses a CSV file of product data and loads the items into a `struct` with the following members:

   - ID
   - Name
   - Price
   - Quantity

   Additional Requirements

   - Use file `prices.csv` for an example of what the input data will look like.
   - Your program should take two arguments: an input file to process and a price limit.
   - Print only the names of each item to `stdout` that have a price less than or equal to the given limit.
   - If an item has 0 stock, indicated by quantity, do not print it out.
   - If the given file does not exist or cannot be opened, warn the user by printing `"CANNOT OPEN FILE."` to `stdout`.
   - Save your code as `price_check.c`.

   Example Run

   ```
   $ ./a.out prices.csv 199.99
   MSI X470 Gaming Plus
   Corsair DDR4 Memory
   ```

4. Given a file containing keywords that were parsed from messages on a social media website, create a program that allows a user to determine if a keyword is trending or not. A keyword is considered trending if there are at least 3 occurrences of it in the given file.

   - In your code, create a function that accepts a string and an array of strings. The function should return 1 if the input string is considered trending, 0 otherwise.
   - Your program should accept two command line arguments (not including the program name) representing the name of the file to load and the keyword to check, respectively.
   - Make sure you check that the given file exists.
   - Format your output similar to the example run given below. The keyword should be printed in quotes.
   - Use the file `keywords.txt`.

- Save your code as `trending.c`.

```
$ ./a.out keywords.txt coffee
"coffee" is not trending.

$ ./a.out keywords.txt perseverance
"perseverance" is trending with 6 messages.
```

5. Create a program that reads and writes creature data from and to a file using `fread()` and `fwrite()`. The creature data should be represented as a `struct` with the following data members:

   - Name
   - CR
   - AC
   - HP

   Your program should include a menu that allows the user to list creature data or add a new creature. It should keep the user inside the program until they choose an option to exit.

   - The `struct` should be defined using `typedef`.
   - The data file should be passed as a command line argument.
   - If the file cannot be opened for any reason, warn the user.
   - If the file does not exist, create it.
   - The program should check for invalid input in the menu. You can assume valid data will be entered for the creatures.
   - Your output should match the output shown in the example below. Take care that the divider line is the same length as the name and that the entries have an empty line between them.
   - After adding a creature, it should be written (append) to the file.
   - Save your code as `creature_db.c`.

   Example Run – Adding Creatures

```
1. List Creatures
2. Add Creature
3. Exit
> 2
Enter Name: Adult Blue Dragon
Enter Challenge Rating: 16
Enter Armor Class: 19
Enter Hit Points: 225
Creature added!
```

   Example Run – List Creatures

```
1. List Creatures
2. Add Creature
3. Exit
> 1
Beholder
--------
CR: 13
AC: 18
HP: 180

Adult Blue Dragon
-----------------
CR: 16
AC: 19
HP: 225
```

Create a `zip` file using the name template `LASTNAME_ID_A4.zip` which includes the all required code files. Submit the `zip` file through Canvas.