CS 15 Lab 00: Review

Introduction

Welcome to CS 15! The purpose of this lab is to help you refresh your memory and get ready to hit the ground running this semester — be sure to ask questions if you get stuck, and don't worry if you don't finish everything. The purpose of labs isn't to write perfect, beautiful code, though that's never a bad goal; it's to try out what you're learning in lecture in a low-pressure, hands-on environment, with lots of support from TAs who are here to help support your learning.

This week, you're working at the CS 15 Clothing Store. We sell lots of different Halligan-branded merchandise, and sometimes it's hard for us to keep track of what we have in stock! Your job is to write a program to help us keep track of our inventory.

If you are new to Tufts, or have not taken CS 11, please review the system setup documents provided before beginning this lab. This document assumes you know how to copy files from the Halligan server, and have your code environment ready to go.

You can find the files for the lab on the server here

```
/comp/15/files/lab00_review
```

The Lab

Part I: Reading and Storing Data

The inventory of the CS 15 Boutique changes all the time. In the summer, we sell t-shirt's and baseball hats, but in the winter, we stock mittens and scarves. Because memory is a limited resource, we want to allocate exactly the amount of space we need to store the clothing items we currently carry.

Finish implementing the `read_inventory` function, which takes two parameters: the name of a file as a string (`filename`) and a pointer to the beginning of an array of `ClothingItem` structs (inventory). Your function will need to open the file and read data from the file to be stored by the program. The first line in the file will have an integer: this is the number of different clothing items we currently have in our inventory.

A clothing item is represented by an instance of a `ClothingItem` struct, which stores the name of the clothing item as a string, and the number of that item currently in stock (an `integer`). The `read_inventory` function will dynamically allocate an array of `ClothingItem` structs on the heap. The input file will be structured like this:

```
7
T-shirt 15
Hat 23
Jacket 2
Sunglasses 42
Earrings 13
Water_bottle 1
Socks 37
```

To finish implementing this function, you'll need to open the input file and read the number of clothing items, then allocate an array on the heap to store that many `ClothingItem structs` (make sure the inventory parameter points to the beginning of this array so that we can hold onto it outside this function!). Then, iterate through the file, and read the data and populate the array of `ClothingItem structs`.

Part II: User Interaction

Lastly, you'll want to finish implementing the `find_in_array` function. Right now, `find_in_array` only works if we have the clothing item in stock — it won't tell the user if they ask for something we don't carry. You'll want to modify this function so that if we don't carry the item they ask for, we let them know! For an additional challenge, try modifying this function so that user input is case-insensitive.

Make sure to clean up/recycle the dynamic memory your program uses before it terminates by freeing the dynamically allocated array after you're done using it!

Part III: Submitting Your Work

When you're done, turn in the lab on gradescope

Hints and Reminders

Working with Structs

To access a property of an instance of a struct, use the dot syntax. For example, suppose I have an `Animal` struct, that has three properties: `age` (an `integer`), `name` (a `string`), and `num_legs` (an `integer`). I can declare an instance of the `Animal` struct just as I would declare any other variable:

```
Animal my_dog;
```

And I can access the properties of that animal using `.`:

```
my_dog.age      = 12;
my_dog.name     = "Ruby";
my_dog.num_legs = 4;
```

A few Common Linux Commands

- `mv` *`source destination`*
  Move *source* file to *destination* directory, or rename *source* file to `destination`. For example,

  ```
  mv clothing-store.cpp cs15/
  ```

  means to move the file `clothing-store.cpp` from the current working directory to `cs15`, which is a subdirectory of my current working directory.

  ```
  mv clothing-store.cpp clothingstore.cpp
  ```

  means rename `clothing-store.cpp` to clothingstore.cpp

- `cp` *`source1... destination`*
  Copy *source* file to *destination*. For example,

  ```
  cp /comp/15/files/lab00_review/* ./
  ```

  means to copy all the files in `lab00_review` (`*` means "all") to your current working directory (`.` is shorthand for your current working directory).

- `cat` [*filename*]
  `less` [*filename*]
  `more` [*filename*]
  `head` [*filename*]
  `tail` [*filename*]

  Among others, these tools print the contents of a file to the terminal — they may be useful to check the contents of your files before submitting, especially if you're using Sublime and SFTP to sync files to the homework server.

- `rm` *filename*
  remove a file. **Be careful — there is no undelete!!**

- `rm -rf` *directory*
  remove a directory and everything inside it (including subdirectories). **Be very careful - there is no undelete, and you can delete a lot of stuff this way!**