CS 202 - S22 - Assignment 1

Mr. Piotrowski

CS 135 Review

1 Assignment Introduction

The goal of this assignment is to help you review subjects you have learned in CS 135. These are the following things you need to know:

- Writing a basic C++ program:
  - Importing Libraries
  - Using the namespace
  - Declaring **int main()**
- Printing to terminal (using cout)

- Input from terminal (using cin)
- Data types (ints, floats, doubles, chars, strings, bools, etc...)
- Declaring variables
- Declaring arrays
- Control statements:
  - if else statements
  - while loops
  - for loops
  - switch statements ? (*not required*)
- Constant variables
- Scopes (global vs. local)
- Declaring basic functions
- Pass by Value / Pass by Reference
- Calling functions
- Inputting and outputting with files (ifstream & ofstream)
- Custom data types (structs)
- Command line arguments (argv, argc)
- Bubblesort

This is a big list and I am sure there are things up there that you are unsure about. We will go over most of the scary ones in class (*like structs and command line arguments*). I will also explain many of these things in the assignment, so please read along and do your best. You are always welcome to ask questions, use your notes, write your own tests, and use the internet. **Note:** Copy-pasting from the internet is plagiarism. I also do not support importing any weird libraries to get your solution. I strongly recommend using the internet only to help you understand the subject.

2 Assignment Overview

You will be writing a program to read in a **tab-delimited text file** (*essentially an excel spreadsheet*) containing nutrition information about popular cereals. Your program will store all cereals and their information in an array. The program will then present the user with options allowing them to search, sort, and find cereal in the array. Below is the program flow:

1. Open the Cereal.txt file using input either from the user or from the command line. If the file does not open properly, the user should be re-prompted to enter the correct file name.

   ```
   [piotrj1@bobby test]$ ./a.out Cereal.txt
   File Opened!
   ```

   or...

   ```
   [piotrj1@bobby test]$ ./a.out
   Please Enter File Name
   Cereal.txt
   File Opened!
   ```

   or...

   ```
   [piotrj1@bobby test]$ ./a.out wrongFile.txt
   Error: File failed to open.
   Please Enter File Name
   wrongFile.txt
   Error: File failed to open.
   Please Enter File Name
   Cereal.txt
   File Opened!
   ```

   All of the above methods should work. You do not get to pick and choose.

2. Once the file is open, store all cereals and their information into an array of **cereal** type. You will need to declare a struct for the cereal type and add variables inside the struct for each field (*i.e. column*).

3. Once all the cereals have been stored, the user is presented with this menu:

   ```
   --------------------------
   |          Menu          |
   --------------------------
    (1) Search by Name
    (2) Sort Cereal Array
    (3) Find Max or Min Cereal
    (4) Write Cereal Array to File
    (5) Print the Whole Array
    (6) Quit
   --------------------------
   ```

   This menu will be presented to the user after each selection. The only way the user can quit the program is by entering 6.

   - Option 1 allows a user to search a cereal by name. If the cereal is found, the cereal is then printed.

     ```
     --------------------------
     |          Menu          |
     --------------------------
      (1) Search by Name
      (2) Sort Cereal Array
      (3) Find Max or Min Cereal
     ```

```
 (4) Write Cereal Array to File
 (5) Print the Whole Array
 (6) Quit
--------------------------
1
Enter Cereal Name
Cap'n'Crunch
Cereal:
*******************
* Cap'n'Crunch
*******************
* MFR: Q
* TYPE: C
* CALORIES: 120
* PROTEIN: 1
* FAT: 2
* SODIUM: 220
* FIBER: 0
* CARBO: 12
* SUGARS: 12
* POTASS: 35
* VITAMINS: 25
* SHELF: 2
* WEIGHT: 1
* CUPS: 0.75
* RATING: 18.04
*******************
```

- Option 2 allows a user to sort the cereal array using any field. The sort is always done from greatest to least.

```
--------------------------
|          Menu          |
--------------------------
 (1) Search by Name
 (2) Sort Cereal Array
 (3) Find Max or Min Cereal
 (4) Write Cereal Array to File
 (5) Print the Whole Array
 (6) Quit
--------------------------
2
What is the type of field you are sorting on?
  (n)umber
  (c)haracter
  (s)tring
n
What field are you sorting on?
cups
Sorting complete.
```

- Option 3 allows for a user to find the cereal with a max or min value in a field of the users choice. Once the cereal is found, it is printed.

```
--------------------------
|          Menu          |
--------------------------
 (1) Search by Name
 (2) Sort Cereal Array
 (3) Find Max or Min Cereal
 (4) Write Cereal Array to File
 (5) Print the Whole Array
 (6) Quit
--------------------------
3
Do you want to find the max or the min value?
(0) min
(1) max
1
What numeric field do you want to look for?
potass
Max Found!
Cereal:
*******************
* All-Bran with Extra Fiber
*******************
* MFR: K
* TYPE: C
* CALORIES: 50
* PROTEIN: 4
* FAT: 0
* SODIUM: 140
* FIBER: 14
* CARBO: 8
* SUGARS: 0
* POTASS: 330
* VITAMINS: 25
* SHELF: 3
* WEIGHT: 1
* CUPS: 0.5
* RATING: 93.7
*******************
```

- Option 4 allows the user to write the cereal array in its current form to a file. The user must provide a file name.

```
--------------------------
|          Menu          |
--------------------------
 (1) Search by Name
 (2) Sort Cereal Array
 (3) Find Max or Min Cereal
 (4) Write Cereal Array to File
 (5) Print the Whole Array
 (6) Quit
--------------------------
4
Enter the file name.
myCereal.txt
```

```
Saved to myCereal.txt.
```

- Option 5 allows the user to print the whole cereal array to the screen. I will not display this one here... please see the sample output files.
- Option 6 allows the user to quit the program.

Now if you think you are a code pro, go ahead and do this any way you want. Spaghetti code it all in main if you want (*not recommended*), but if you are uncertain about how to get this done, feel free to follow the steps below. The following section describes how Mr. P would accomplish this program.

3 Coding the Program

3.1 Create your main.cpp file and code skeleton

First you should make your main.cpp file. Import all your libraries here. You are only allowed 3:

- iostream
- fstream
- string

Also be sure to use the STD namespace and declare your int main(). After this, your code should compile and execute, though it will not do anything.

3.2 Declare your cereal struct

In the global scope (above main), declare a struct for the cereal. It might look like this:

```
struct cereal{
    // Fields go here
};
```

Inside your struct you should have 16 fields, one for each column in the text file. Make sure the types are correct. You will have variables that are string, char, int, and float. If you don't know, just open the file and look at them! Easiest to do if you can copy-paste it into excel.

3.3 Declare a Global Const Variable for Array Size

In CS 135 you should have learned about arrays. You should know the arrays that you coded in the class were not re-sizable (*meaning they have a constant size*). At this point in your program, declare a global constant integer **maxSize** that you can use to declare the cereal array in main. A good maxSize for this program is 100.

3.4 Declare Function Prototypes

Remember that function definitions can be written above and below main. If the function definitions go below main, the function headers (*prototypes*) need to go above main so that the program can see them. The most accepted way is with the definitions below main. This makes it easy for readers of our code to see all available functions and main().

I will give you all prototypes that I used in my solution here:

```
// Function to read the file
bool readFile(string fName, cereal cArr[], int &cSize);
// Function to print a cereal
void printCereal(const cereal &c, ostream &out);
// Function to print all cereals
void printCerealArr(const cereal cArr[]
    , const int &cSize, ostream &out);
// Function to extract a numeric field from cereal
float getNumber(string fieldName, const cereal &c);
// Function to extract char field from cereal
char getChar(string fieldName, const cereal &c);
// Function to extract the name of cereal
string getName(const cereal &c);
// Function to print the menu and get input
int menu();
// Function to search cereal array by name
void searchByName(string name, const cereal cArr[]
    , const int &cSize, ostream& out);
// Function to swap two cereals in array (for sorting)
void swapCereal(cereal &c1, cereal &c2);
// Function to sort the array on any field
void sortGreatestToLeast(char fieldType, string fieldName
    , cereal cArr[], const int &cSize);
// Function to get cereal with max or min value
cereal getMaxOrMin(string fieldName, const cereal cArr[]
    , const int &cSize, bool max);
```

I will go over each of these functions below. For now, we will just include the prototypes.

Note: I would recommend completing them in the order listed above, as you will be able to use functions for other functions.

3.5 Complete the readFile Function

The first thing that happens in this program is reading in the file, so let us code that function. Notice a few things here:

- string fName - the file name is provided as a parameter
- cereal cArr[] - the cereal array is passed by reference (*all arrays are pass by reference*). We will be storing the cereal in this array.

- int &cSize - the size of the cereal array. This is not the maxSize. As we read in cereal, we should increment the cSize so we know exactly how much cereal is in the array.

First thing you should do is to try and open the file using the string that was passed into the function. If the file fails to open, print "*Error: File failed to open.*" and return **false**. The purpose of returning the bool is so main can know easily if the file read was successful or not. If the file open succeeds, print "*File Opened!*" and read in the file. Here are some helpful tips:

- Initialize cSize to 0 to ensure you start at the beginning of the array.
- Read in the header line (*the column headers*) first. You do not have to do anything with it, you just don't want it messing with your integer or float conversions.
- Use a while loop. You will want to make sure that the cSize is always less than the maxSize **AND** there is at least 1 more line to be read. If at any point one of these things is false, close the file and return **true**.
- getline() is your friend here. It can be called in 2 ways:

  ```
      getline(ifstream, string)
      // Reads one whole line from the file into the string
  ```

  ```
      getline(ifstream,string,char)
      // Reads all characters in the file up to the first
      appearance of the char.
  ```

  In fact, getline(myFile, temp) is the exact same thing as getline(myFile, temp, '\n'), as it will read everything up until it sees the first endline. You may want to use getline(myFile, temp, '\t'). The '\t' is the character representing the tab.
- stoi() can be used to convert a string to an integer. stof() can be used to convert a string to a float. Lastly, you can index the string at the first character (*i.e. - myString[0]*) to convert the string to a character. This is assuming that the string has a length of one.

3.6 Complete the printCereal Function

Print the cereal in the format below. The cereal is provided as a parameter. You will also notice the **ostream** parameter in there too. An ostream is an "out-stream". One popular ostream you know is **cout**. It might also be an **ofstream** (*out-file-stream*). So, if you wanted to use this function to print to the screen, it is called like this:

```
printCereal(myCereal, cout);
```

... or if you want to write it you your file:

```
printCereal(myCereal, myOutFile);
```

So, instead of hard-coding **"cout << ..."** for this function, use the ostream parameter instead.

```
Cereal:
*******************
* All-Bran with Extra Fiber
*******************
* MFR: K
* TYPE: C
* CALORIES: 50
* PROTEIN: 4
* FAT: 0
* SODIUM: 140
* FIBER: 14
* CARBO: 8
* SUGARS: 0
* POTASS: 330
* VITAMINS: 25
* SHELF: 3
* WEIGHT: 1
* CUPS: 0.5
* RATING: 93.7
*******************
```

3.7 Complete the printCerealArr Function

This function prints every cereal in the array using the same format as above. In fact, you could call the printCereal function in this function! You will need to use some sort of loop to go through all the cereal. Notice the size is passed as a parameter and the same ostream parameter is there too (*for the same reason as mentioned in the previous section*).

3.8 File Input and Testing Code

Now we have 3 of the functions done. It is time to go into your empty main and write some code there. First thing you need to do is to declare your cereal array and cereal size. Next, write code that will open the file as described in the program flow. To remind you:

1. Check to see if the filename was provided as argv[1]. If so, save the file name into a file string variable.

2. If the file name was not provided as a command line argument, prompt the user for the file name and read it into your file string variable using cin. "*Please Enter File Name*"

Now try to open the file. Consider calling the readFile function we coded already. If the function returns false, we know that the file open failed and the user should be re-prompted for the correct file name with the same message as above. If the readFile function returns true, print "*File Opened!*" At this point, we know that the cereal array should now contain all the cereal data and the cereal size should have the number of cereals in the array. Next, test your printCereal function like this:

```
printCereal(cerealArr[0], cout);
```

... and your printCerealArr function like this:

```
printCerealArr(cerealArr, cerealSize, cout);
```

Make sure all this stuff is working before continuing.

3.9 Completing getNumber and getChar

These functions allow for easy access to the cereal fields. For example, lets say we wanted to extract the sugar content of Cap'n'Crunch. We would could easily extract it like this:

```
int sugars = getNumber("sugar", myCereal);
```

Similarly, if we wanted to get a cereal type and mfr, it could be done like this:

```
char type = getChar("type", myCereal);
char mfr = getChar("mfr", myCereal);
```

We could have 2 separate functions for the float fields and the int fields, but that is not necessary for the purpose of this program. Integers can easily be cast to floats, so the getNumber function only returns floats.

3.9.1 getNumber

Check the fieldName parameter to see which field it is. If it is "sugar", return the cereals sugar field. If it is "carbo", return the cereals carbo field. Etc... There are 13 total number fields.

If the fieldName is unknown, simply return -1.

3.9.2 getChar

Same as above but for the characters. There are 2 total char fields.

If the fieldName is unknown, simply return '\0' (*null character*).

Once finished, consider testing them in main.

3.10 Completing getName Function

Simply return the name field of the cereal parameter.

3.11 Completing the menu function

This function will print the menu below and take input from the user. The function should ensure that a number 1 through 6 is entered. If it is a valid entry, return the number. If there is an invalid entry, print "*Error: Invalid Entry.*" and re-prompt the user with the entire menu.

```
-------------------------
|          Menu         |
-------------------------
 (1) Search by Name
 (2) Sort Cereal Array
 (3) Find Max or Min Cereal
 (4) Write Cereal Array to File
 (5) Print the Whole Array
 (6) Quit
-------------------------
```

3.12 Completing the searchByName Function

The name of the cereal a user is looking for is passed into this function as the name parameter. Search the entire cereal array looking for the name. If the name is found, print that cereal. If the name is not found, print "*Error: Did not find that cereal.*"

Note that the same ostream parameter from the print functions is passed here as well, indicating that a user may want to write the cereal to a different out stream other than cout.

At this point you will want to test your code. Go into main and setup your loop. The loop should call the menu() function and save the return value into some variable. Using this variable, you can set up cases for searching cereal by name, writing the cereal array to a file, printing the whole array, and quitting. I would recommend making sure all of these things are working before moving on.

3.13 Completing the swapCereal Function

This function simply swaps two cereals. The cereals are pass by reference, so changes that are made to the parameters are reflected in the original variables too (*in this case, the array*). Write the code that swaps the two cereals.

3.14 Completing the sortGreatestToLeast Function

This function sorts the cereal array on any of the fields. This means we could sort the array based on sugar content, alphabetical order of the name, alphabetical order of the type, or even the nutrition rating.

You will notice the fieldType character passed to this function. This indicates what data type the user wants to sort on. It could be a string 's' (*the cereal name*), a char 'c' (*cereal mfr or type*), or a number 'n' (*one of the numerical fields*). In other words, the fieldType determines which of the getChar/getNumber/getName functions you will call.

Next, the fieldName is the parameter that tells us which field we are sorting on. For example, if we were to sort on sugar content, we would call the function like this:

```
sortGreatestToLeast('n', "sugar", cerealArray, cerealSize);
```

We pass an 'n' because the sugar field is a number.

You should use bubblesort for this function. I understand you may not know this one by heart, so here is a skeleton. I will talk about bubblesort in class.

```
for(int i = 0; i < size - 1; i++){
    for(int j = 0; j < size - i - 1; j++){
        // EDIT THE CODE BETWEEN THIS LINE
        if(/* compare arr[j] and arr[j+1] */){
            swap(arr[j],arr[j+1]);
        }
        // AND THIS LINE
    }
}
```

Inside the comments is an if statement that will check to see if the arr[j] is less than arr[j+1]. If so, it will swap them. You will need to add some more logic here to compare the proper fields of the cereal.

3.15 Completing the getMaxOrMin Function

Last function! Let us first talk about parameters. The fieldName string parameter is telling us which numeric field we are finding the max or min of. The cArr and cSize is our cereal array and cereal size. Lastly, the max parameter is a boolean telling us if we are looking for the max (*true*) or the min (*false*).

This function only cares about numeric values, so you can always expect to use the getNumber function. Now, there are 2 ways we can find a max or min:

- The first way would be to iterate over the entire list and keep track of the min/max cereal as you go along. This will likely be the harder one to code, but it will be the most efficient.
- The second way would be to sort the entire array first, Then, if you are finding a min, take the element at the end of the array. If you are finding the max, take the element at the beginning of the array. Be careful with this one though... All arrays are pass by reference. If you sort the array, you will be modifying it. This function is not supposed to do that. If you want to use this method, you will want to create a copy of the array (*you*

will need to declare a new array and use a for loop to copy elements 1 by 1).

Once you have found the max/min, return that cereal variable.

3.16 Finishing Up

You will now want to move up into main and fill out the rest of it. All of the expected outputs for each menu selection can be seen on pages 3-6 of this document.

1. Option 1 will prompt the user with "*Enter Cereal Name*" and then take the users input. After this is done, you can call the searchByName function you coded earlier.

2. Option 2 will prompt the user for 2 things. First it asks for the data type of the field we are sorting on. This is prompted with this:

   ```
   What is the type of field you are sorting on?
     (n)umber
     (c)haracter
     (s)tring
   ```

   Then, take the user input as a char and save it. Next, you need to ask the user for the name of the field they are sorting on. Prompt them with "*What field are you sorting on?*" and then take the input. At this point you can call the sort function we coded earlier. Once the sort function returns, print "*Sorting complete.*"

3. Option 3 should prompt the user to select either a max or min.

   ```
   Do you want to find the max or the min value?
   (0) min
   (1) max
   ```

   Note that the user should enter a 0 or a 1. In c++, false == 0 and true == 1, so you can save the user input directly into a bool. Next, ask the user to input the field they want to find that max or min of with "*What numeric field do you want to look for?*". Take the user input and then call the function we coded earlier. Once the function returns, print "*Max/Min Found!*" depending on the user selection. Then print the cereal that was found.

4. Option 4 allows the user to save the entire cereal array to a file of their choice. Prompt the user with "*Enter the file name.*" and save the user input. Next, open an out file stream with the file name they entered. After that, write every cereal to the out file. This is easily done with the printCerealArr function we coded earlier. Instead of passing cout, pass in your out file stream. Once the function returns, close the file and print "*Saved to <userFile>*" where the <userFile> is replaced with the entered file name.

5. Option 5 allows the user to print the entire array to the screen. Simply call the same printCerealArr with cout.
6. Option 6 quits the program. You can just flat out return 0, or exit your loop so that the end of main can be reached.

4 Compiling & Execution

Compile your code with the following statement:

```
g++ -std=c++11 main.cpp
```

Execute your code with the following statements:

```
./a.out
```

```
./a.out Cereal.txt
```

```
./a.out wrongFile.txt
```

5 Submitting to Codegrade

All assignments this semester are going to be submitted to codegrade. I do understand that it can be quite annoying for many of you but it is quite helpful to me. In class I will go over using codegrade.

Depending on how codegrade is set up, it may dock you **lots** of points if your program fails to run correctly. In these cases, I will have TAs manually review your code to give some points back if necessary.

This assignment **will not** be manually reviewed. The codegrade is set up to check each portion of the program individually. As long as you get some things correct, you will be guaranteed to get partial credit.