CSCI 2270
2018 Instructor: Lambrocco
Assignment 1: Due Saturday 6/16 9:00am

Community Garage Sale

Read the entire assignment carefully before beginning.

Goals:

- Read file and parse values
- Create new types to fit best the context of the problem we are dealing with
- Organize data in convenient structures
- Analyze the algorithms implemented

What should your program do:

- Read in the file *garageSale.txt* and parse values as needed
- Store the values of a single item in an object of type *GarageItem*
- Store all the *GarageItem* in an array called *arrayOfGarageItems*
- Have the following functionalities:
    - Find item (25pts)
    - Add item if item is not found (20pts)
    - Buy item/Remove item (25pts)
    - Print list of item in specific order (15pts)
    - Profile sorting routines (15pts)

Description

There is a file on Moodle called *garageSale.txt* that includes up to 100 wanted or for sale items in five categories: bike, microwave, dresser, truck, or chicken. Each line in the file is one item. Your program needs to open the file, read each line, and use an array of *structs* to store the available items. You can assume there will never be more than 100 lines in the file, therefore, you can declare an array of *structs* with a fixed size of 100 to represent the items available on the message board. Each *struct* represents an item and has a type (bicycle, truck, etc...), a price, and whether it is for sale or wanted. (You can treat for sale or wanted as an integer or Boolean, where 0 is for sale and 1 is wanted, for example.)

Your program needs to read the file until it reaches the end, and you can't assume there will always be 100 lines, there may be less. As lines are read from the file, you need to check if there is a match with the existing items in the message board. There are two options to consider:

1) **Match is not found in the array**
   If a match is not found in the array, add the item to the array at the first unused position, e.g. if there are four items, add the item to position five.

2) **Match is found in the array**
   If a match is found, use the first match found and stop searching the array. Do not add the new item read from the file to the array. Remove the matched item from the array and shift the array to fill the gap left by the removed item. (Section 3.2.4 of your book shows the algorithm for deleting an item from an array and shifting.) Write the action performed to the terminal, formatted as <type><space><price>, such as *bike 50*. Your printing should be done with the command:

   ```
   cout<<arrayOfGarageItems[x].type<<" "<<arrayOfGarageItems[x].price<<endl;
   ```

   where arrayOfGarageItems is the array of *structs (GarageItem)* and *x* is the index where the item was found in the array. The *type* is one of the following: **bike, microwave, dresser, truck,** or **chicken**. The *price* is the actual item cost, not what the user is willing to pay.

Moreover, your program should:

1) **Handle the file name as a command line argument**
   Require the user to enter the name of the file to open as an argument to the program. When the TAs test your code, the filename we use will not be *garageSale.txt*. We will run your program from the command line, such as

   ```
   ./Assignment1 testFile.txt
   ```

   where *Assignment1* is the name of the executable they build when they compile your code, and *testFile.txt* in the filename. Your program needs to use *argv[1]* as the filename in the ifstream.

2) **Print array contents after all lines read from file**

After all lines have been read from the file and all possible matches have been made, there will be items left in the array that no one wanted. Include a function in your program that prints out the final state of the message board, and call the function after displaying the matched items. The function parameters and return values are at your discretion, but the function needs to correctly print the contents of the array using the command:

```
cout<<arrayOfGarageItems[x].type<<", "<<"for sale"<<", "<<arrayOfGarageItems[x].price<<endl;
```

for "for sale" items and

```
cout<<arrayOfGarageItems[x].type<<", "<<"wanted"<<", "<<arrayOfGarageItems[x].price<<endl;
```

for "wanted" items.

3) **Format and ordering of program output**

It's important that the output of your program is formatted and ordered in a certain way. You should use the *cout* statements given in the above sections and output your results in the following order:

Items sold.
#
Items remaining in the message board after reading all lines in the file. #

Don't output anything other than what's specified.

4) **Print all the remaining items ordered at discretion of the user**

Once you print all the remaining items you should ask the user if they want to *exit* the program or if they want to display the remaining items in alphabetical order. If the user selects "*exit"*, the program should end; if the user selects "*order items"* you should output the remaining items in alphabetical order.

To achieve this you should write:

1) a function called *orderRemainingItems()*
2) a sorting function called *bubbleSortRoutine(...)*
3) a sorting function called *insertionSortRoutine(...)*
4) a sorting function called *quickSortRoutine(...)*

The function *orderRemainingItems()* should call the three routines/functions:

- *bubbleSortRoutine*
- *insertionSortRoutine*
- *quickSortRoutine*

to sort an array that has the remaining items.

The goal of *orderRemainingItems()* is to access the array of remaining items and sort them alphabetically using three different algorithms.

- Bubble Sort
- Insertion Sort
- Quick Sort

You should "profile" the sorting algorithms. This means that you will have to compute how much time it took each function call in *orderRemainingItems()* to execute.

After each sorting algorithm executes you should output the execution time in the following matter:

```
cout<< "<sorting_algoritm> executed in" <<time<<endl;
```

where <sorting_algoritm> is the name of the specific sorting algorithm that just executed and time is a variable that stores the execution time of the algorithm. Finally, at the end of the function *orderRemainingItems()* you should print the ordered items following the printing format at bullet point 2).

How to know if your output is correct?

There is a Piazza forum for this class and you are welcome to post the output you get for the garageSale.txt file on Piazza and ask if other students get the same answer. If you want to test your code with a smaller data set, create a new .txt file and use that file as the input file when you run your code. Please don't post your code on Piazza.

What is a Struct?

I mentioned structs in class on Wednesday during the C++ review. For more information on what *structs* are and how to create them, there are links on Moodle under Useful YouTube Videos that describe structs in more detail

Submitting Your Code:

Submit your assignment to Moodle

Submit your .cpp file through Moodle using the Homework 1 Submit link. Make sure your code is commented enough to describe what it is doing. Include a comment block at the top of the .cpp file with your name, assignment number, and course instructor.

What to do if you have questions

There are several ways to get help on assignments in 2270, and depending on your question, some sources are better than others. There is a discussion forum on Piazza that is a good place to post technical questions, such as how to shift an array. When you answer other students' questions on Piazza, please do not post entire assignment solutions. The CAs are also a good source of technical information, especially questions about C++. If, after reading the assignment write-up, you need clarification on what you're being asked to do in the assignment, the TAs and the Instructor are better sources of information than the CAs. We will be monitoring Piazza regularly.