CSCI 2270 Data Structures and Algorithms
Spring 2016
Instructor: Hoenigman
Assignment 1
Due Wednesday, January 27, by 3pm

It's like Craigslist, only different

Read the entire assignment carefully before beginning. In this assignment, you're going to develop a simulated community message board that monitors items *wanted* and items *for sale* and looks for matches. When a match is found, e.g. there is a bike for sale for $50 and a bike wanted, where the buyer will pay up to $60, then the item is removed from the message board.

There is a file on Moodle called *messageBoard.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 with a fixed size of 100. Each struct represents an item and has a type, such as bicycle or truck, 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.) The struct array represents the message board.

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, representing new items being posted to the message board, compare the item to existing items in the struct array to search for a match.

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.

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 to the array. Remove the matched item that is already in the array and shift the array to fill the gap left by the removed item. 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<<itemArray[x].type<<" "<<itemArray[x].price<<endl;
```

where *itemArray* is the array of structs 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.

Other things your program needs to do

Handle the file name as a command-line argument

Your code needs to accept the file name as a command line argument and then open the file with that name. You will be running your code on the COG autograder, and the filenames we use will not be *messageBoard.txt*.

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

for "for sale" items and

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

for "wanted" items.

Count loop iterations

For this program, we're going to count the number of times that loops execute. Iterations of a loop is an example of an operation that scales with the size of the input data that needs to be processed. The block of code that reads each line from the file depends on a loop because the number of lines in the file can change if you use a different input file. The code needed to search the message board, or shift data in the array, also scales with data input because the size of the message board will change as items are added and removed.

At the end of your program, after displaying the items left on the message board, display the total number of loop iterations needed to run the program. The count should be displayed on its own line, formatted as:

```
cout<<"loop iterations:"<<counter<<endl;
```

Loop iterations example

After reading the first line in the file, your loop counter should be 1 for the read from the file.

When the first sale occurs, all loop iterations needed to complete the sale should be the sum of: 1 for the read from the file, *x* for the number of iterations it takes to find the matching item in the array, and *y* for each shift in the contents of the array.

Format and ordering of program output
You will be submitting your assignment to the COG autograder, so 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.
#
Number of loop iterations

Use the # as the delimiter for each output type, the COG autograder will look for that character. Don't output anything other than what's specified, it will confuse COG. Even the seemingly harmless newline character could spell doom.

Submitting Your Code:

Submit your assignment to the COG autograder:
https://web-cog-csci2270.cs.colorado.edu/submit.html.

Login to COG using your identikey and password. Select the CSCI2270 - Hoenigman – HW #01 from the dropdown. Upload your file and click Submit. **Your file needs to be named Assignment1.cpp for the grading script to run.** COG will run its tests and display the results in the window below the Submit button. If your code doesn't run correctly on COG, read the error messages carefully, correct the mistakes in your code, and upload a new file. You can modify your code and resubmit as many times as you need to, up until the assignment due date.

Submit your assignment to Moodle
In addition to submitting through COG, submit your .cpp file through Moodle using the Assignment 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. **If you do not submit your work to Moodle, we will deduct points from your grade, even if COG gives you a perfect score.**

If you do not get your assignment to run on COG, you will have the option of scheduling an interview grade with your TA to get a grade for the assignment. Even if you do get the assignment to run on COG, you can schedule the interview if you just want to talk about the assignment and get feedback on your implementation.

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 Peer Discussion Forum on our Moodle page that is a good place to post technical questions, such as how to shift an array. When you answer other students' questions on the forum, please do not post

entire assignment solutions. The multi-course PLAs 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 discussion forum or the PLAs.