CSE 1384 – Advanced Sorts and Searches
Lab 7

Objectives

1. Continue practicing past concepts
2. Practice building basic C++ classes
3. Practice using advanced sorts and searches

Assignment

Create a program that reads a list of widgets, costs, and totals from a file. The program should sort the list by widget name and display a "receipt." The user should then be given a chance to delete widgets from the list. Once the user has finished editing the list, the list should be sorted by cost and the "receipt" displayed again.

For this lab, create a basic C++ class using the following UML class diagram:

Widget
-name: string -unitCost: float -count: int
+Widget() +Widget(const string &name) +Widget(const string &name, float unitCost) +getName(): string +getUnitCost(): float +getCount(): int +getTotal(): float +setName(const string &name): void +setUnitCost(float unitCost): void +setCount(int count): void +addCount(int count): void

For any basic tips regarding UML class diagrams, refer to the lab 4 hints.

Then create a main file that will have the following functions:

main

Parameters: none
Return value: integer 0
Purpose: Read a list of widgets, costs, and totals from a user specified file, sort the list by widget name and loop thru the list of widgets and display their name, cost, count and the total cost. Display the total cost for every widget being purchased. Allow the user to search for and remove widgets from the list, this should be performed until the user enters "Done". Sort the updated list by widget cost and display their name, cost, count and the total cost.

readWidgetTable

Parameters: file name, reference to a vector of widgets

Return value: boolean value which is true if the read succeeded. False otherwise

Purpose: Open the specified file and read the contents of the file using the provided **getWidget** function. Stores every widget into the provided vector.

printTotalCost

Parameters: reference to a vector of widgets

Return value: none

Purpose: Loop thru the vector of allocated widgets and display their name, cost, count and the total cost. Display the total cost for every widget being purchased.

quickSortCost

Parameters: reference to a vector of widgets, starting index, ending index

Return value: none

Purpose: One of two components for quicksort, calls **partitionCost** and performs the recursions for quicksort.

partitionCost

Parameters: reference to a vector of widgets, starting index, ending index

Return value: the highest index of the low partition

Purpose: One of two components for quicksort, partitions the list around a pivot value. The pivot value will be the cost of the middle element of the list.

quickSortName

Parameters: reference to a vector of widgets, starting index, ending index

Return value: none

Purpose: One of two components for quicksort, calls **partitionName** and performs the recursions for quicksort.

partitionName

Parameters: reference to a vector of widgets, starting index, ending index

Return value: the highest index of the low partition

Purpose: One of two components for quicksort, partitions the list around a pivot value. The pivot value will be the name of the middle element of the list.

binarySearch

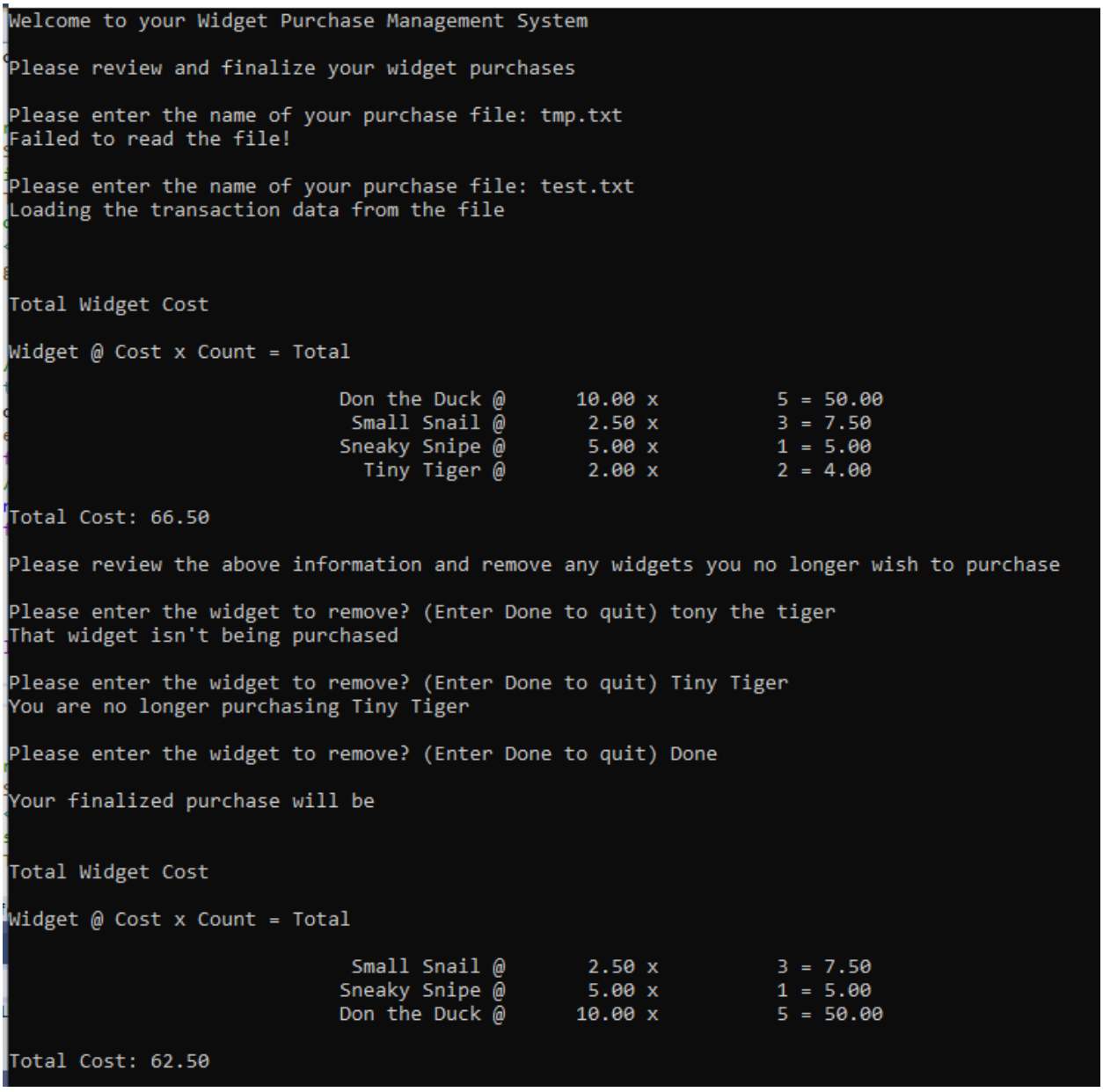
Parameters: reference to a vector of widgets, widget name being searched for, starting index, ending index

Return value: the index of the widget or -1 if it doesn't exist.

Purpose: Use the binary search to locate the specified widget and return its location in the list. Returns a -1 if the widget isn't found.

A starter file has been given that contains the **getWidget** function and repeats the outline of the process detailed above.

Example Execution



```
Welcome to your Widget Purchase Management System

Please review and finalize your widget purchases

Please enter the name of your purchase file: tmp.txt
Failed to read the file!

Please enter the name of your purchase file: test.txt
Loading the transaction data from the file



Total Widget Cost

Widget @ Cost x Count = Total

                          Don the Duck @      10.00 x          5 = 50.00
                           Small Snail @       2.50 x          3 = 7.50
                          Sneaky Snipe @       5.00 x          1 = 5.00
                            Tiny Tiger @       2.00 x          2 = 4.00

Total Cost: 66.50

Please review the above information and remove any widgets you no longer wish to purchase

Please enter the widget to remove? (Enter Done to quit) tony the tiger
That widget isn't being purchased

Please enter the widget to remove? (Enter Done to quit) Tiny Tiger
You are no longer purchasing Tiny Tiger

Please enter the widget to remove? (Enter Done to quit) Done

Your finalized purchase will be


Total Widget Cost

Widget @ Cost x Count = Total

                           Small Snail @       2.50 x          3 = 7.50
                          Sneaky Snipe @       5.00 x          1 = 5.00
                          Don the Duck @      10.00 x          5 = 50.00

Total Cost: 62.50
```

Comment Block

Your code should contain a comment block at the top containing information on who wrote the code, what the assignment is, when it is due, etc. Here is an example of a good comment block to put:

```
/*
Name: <your name>                    NetID: <your netID>
Date: <current date>                 Due Date: <enter in due date>
```

```
Description: <What is the program?>
*/
```

Deliverables

- C++ code (.cpp and .h files)
- A document (.pdf) with two screenshots showing the program running
    - The two program screenshots should have completely different inputs from each other
    - The two screenshots must be legible to count (too small or pixelated text will not be interpreted)
    - Show all error messages

Point Breakdown
(100 points total)
A submission that doesn't contain any code will receive a 0.

- 5pts – no global variables
- 10pts – IO
    - 5pts – user input handled correctly
    - 5pts – output handled correctly
- 5pts - vector handling
    - 5pts - vector is of Widget type
- 14pts - main (correct)
- 8pts – readWidgetTable
    - 4pts - parameters/return values
    - 4pts - correct (handles what it should, not what it shouldn't)
- 8pts - quickSortCost
    - 4pts - parameters/return values
    - 4pts - correct (handles what it should, not what it shouldn't)
- 8pts – partitionCost
    - 4pts - parameters/return values
    - 4pts - correct (handles what it should, not what it shouldn't)
- 8pts – binarySearch
    - 4pts - parameters/return values
    - 4pts - correct (handles what it should, not what it shouldn't)
- 8pts - quickSortName
    - 4pts - parameters/return values
    - 4pts - correct (handles what it should, not what it shouldn't)
- 8pts – partitionName
    - 4pts - parameters/return values
    - 4pts - correct (handles what it should, not what it shouldn't)
- 8pts – printTotalCost
    - 4pts - parameters/return values
    - 4pts - correct (handles what it should, not what it shouldn't)
- 5pts – turned in two unique screenshots
- 5pts – programming style (includes good commenting, variable nomenclature, good whitespace, etc).