Lab 11 - Arrays, Searching and Sorting

Directions

- The labs are marked based on attendance and effort.
  It is your responsibility to ensure the TA records your progress by the end of the lab.
- While completing these labs, you are encouraged to help your classmates and receive as much help as you like. Assignments, however, are individual work.
  You may not work on assignments during your lab time.
- If you do not finish the lab exercises during your lab time, you are encouraged to complete them later to finish learning the material.

1. Perfect Scores with an Array

1. Write a program which reads in five test scores (integers) from the user.
   - Create an array to store each score.
   - Ensure each value being read in is between 0 and 100. Assume they are integers.
   - Sample Run:
```
Enter score #0: 5
Enter score #1: -1
Invalid score. Must be between 0 and 100.
Enter score #1: 101
Invalid score. Must be between 0 and 100.
Enter score #1: 123
Invalid score. Must be between 0 and 100.
Enter score #1: 50
Enter score #2: 70
Enter score #3: 100
Enter score #4: 0
```
2. Write a function which returns the number of perfect scores (i.e. scores of 100).
   `int countPerfect(int data[], int arraySize);`
   - Have `main()` call `countPerfect()` and have `main()` output the number of perfect scores.
3. Write a function which returns the average score. Have `main()` output it to the screen.
   - Remember that your function should solve the general problem of finding the average for an array; it should not be hard-coded to a specific size of an array.
   - Additional output:
```
Number of perfect scores: 1.
Average scores: 45.
```
4. **Understanding:**
   - How to create, write to, and read from arrays.
   - How are arrays passed to functions? Why do you need to pass the number of elements when working with arrays, but not with vectors?

  - What change would you make to how the program stores data if requirements for the program changed to "Read in values until the user enters -1, then do the same processing as before"? (You do not need to actually make this change.)

2. Many Mopeds

Using the Moped class created in last week's lab (use solution on website if needed), create a program which manages a tourist shop's collection of Mopeds.

1. Add a `displayInfo()` method to the Moped class which will output (on one line), a short summary of the Moped's information. For example (values from member variables are underlined):
   `"Red 2011 moped named Hot Stuff".`
   - Comment out any `cout` statements in the Moped's constructors and destructor.
2. In your client code, create a function named `createMoped()` which reads in a name, year and colour from the user, and then creates a moped based on this.
   - Just use `cin` to read strings (i.e., you only need to handle single word strings).
   - Have the function return this Moped object. What return type does the function need?
   - When called, the function has the following output:
     ```
     Enter name: Jack
     Enter year: 2011
     Enter colour: Blue
     ```
3. In `main()` create a menu to manage a list of Mopeds.
   - Start by creating a local variable in `main()` to store the Mopeds as a vector:
     ```
     vector<Moped> mopeds;
     ```
   - Display a simple menu with three options:
     - Add moped.
     - List mopeds.
     - Exit.
4. Handle the "Add Moped".
   - Use the `createMoped()` function and the vector's `push_back()` function:
     ```
     Moped newMoped = createMoped();
     mopeds.push_back(newMoped);
     ```
   - Remember: If you are creating a variable inside a switch statement, you need to encase the case's code in {...}.
5. Create a function `listMopeds()` which takes a vector of Mopeds as its one parameter.
   - Have the function display all the mopeds to the screen, using the moped's `displayInfo()` method.

6. Sample output:

```
1. Add moped.
2. List mopeds.
3. Exit.
1
Enter name: Lightning
Enter year: 2011
Enter colour: Red
1. Add moped.
2. List mopeds.
3. Exit.
1
Enter name: PuttPutt
Enter year: 1952
Enter colour: Black
1. Add moped.
2. List mopeds.
3. Exit.
1
Enter name: NoGo
Enter year: 2001
Enter colour: Blue
1. Add moped.
2. List mopeds.
3. Exit.
2
Red 2011 moped named Lightning
Black 1952 moped named PuttPutt
Blue 2001 moped named NoGo
1. Add moped.
2. List mopeds.
3. Exit.
3
```

7. When an object is pushed-back into the vector, only a copy of the object is inserted into the vector. Therefore, any changes to the original object after it is inserted into the vector will not change the object held in the vector.
   - Experiment with this by changing the year of the moped to 9999 after a copy of it is inserted in the vector (step 4).
8. Passing a vector by value to a function creates a copy of the vector, and that is what the function uses. This usually works fine, but can be slow for larger vectors (for example, all SFU students!).
   - We can get around having to copy the object by instead passing it by reference.
   - Modify your code to pass the vector by reference instead. Prove that the program works as before.

3. Sorting and Searching Cities

3.1 Display

1. Create a new program named `citySearch.cpp` which declares the following array of strings in `main()`:

```
string cities[] = { "St Louis du Ha! Ha!",
                    "WaWa",
                    "Forget",
                    "Come by Chance",
                    "Vulcan",
                    "Blow Me Down",
                    "Noggin Cove",
                    "Paradise",
                    "Sucker Creek"};
```

2. Create a function with the following header:
   `void displayCities(string data[], int size);`
   - Have `main()` call this function to display all the cities. For example:

```
Cities:
        1. St Louis du Ha! Ha!
        2. WaWa
        3. Forget
        4. Come by Chance
        5. Vulcan
        6. Blow Me Down
        7. Noggin Cove
        8. Paradise
        9. Sucker Creek
```

3.2 Sort

1. Write a function with the following header which sorts the passed-in array of strings by the length of the strings.
   `void sortByLength (string data[], int size);`
   - You can use any sort algorithm you like. I suggest you start with the sample sort algorithms posted online from lecture.
   - It should **put the shortest string first, and the longest string last.**
   - In `main()`, add a call to this function to sort the cities, followed by another call to `displayCities()`, which should add the following output.

```
Cities:
        1. WaWa
        2. Forget
        3. Vulcan
        4. Paradise
        5. Noggin Cove
        6. Blow Me Down
        7. Sucker Creek
        8. Come by Chance
        9. St Louis du Ha! Ha!
```

2. Write a function with the following header:
   `void displayMedian(string data[], int size);`
   - Make this function display to the screen the median element in the string array. The median element is the middle element.
   - In `main()`, place a call to `displayMedian()` passing-in the sorted array of cities to generate an output similar to:

```
Median element: Noggin Cove
```

3. Add another sort function, this time to sort the strings alphabetically:
   `void sortAlphabetically (string data[], int size);`
   - You may use any sort algorithm you like.
   - Tip: You can directly compare two strings to see which is alphabetically first:

```
string s1 = "Alfred", s2 = "Betty";
if (s1 < s2) {
   cout << s1 << " comes before " << s2 << endl;
}
```

   - Add to `main()` extra calls to `sortAlphabetically(), displayCities(),` and `displayMedian():`

```
Cities:
        1. Blow Me Down
        2. Come by Chance
        3. Forget
        4. Noggin Cove
        5. Paradise
        6. St Louis du Ha! Ha!
        7. Sucker Creek
        8. Vulcan
        9. WaWa
Median element: Paradise
```

3.3 Search

1. In `main(),` add code to read in a string from the user.
   - Use `getline()` to read in a string including spaces.
   - Your input might look like:

```
Enter a city to search for: Vulcan
```

2. Implement the following function:
   `int searchCity (string data[], int size, string target);`
   - Have this function use **binary search**.
   - Return the **index** of the element in the array which matches target.
   - Return -1 if there is no match in the array.
   - Note that we have already sorted the cities alphabetically, so we can use a binary search routine without having to sort the array again.

3. Use the `searchCity()` function to find the city the user entered. If the city does not exist, display an error message.
   - Example input and output for a city which does exist:

```
Enter a city to search for: Vulcan
City at array index 7 = Vulcan
```

   - Example output for a city which does not exist:

```
Enter a city to search for: Vancouver
City not found.
```

4. **Understanding**:
   - How to sort objects.
   - What happens when you use binary search when the array is not correctly sorted? Experiment with this by commenting out the call to `sortAlphabetically()`. Try searching for a couple different cities; does it always work?
   - Why is it necessary to use an `if` statement to check the return value of `searchCity()`, instead of just using its return value to look-up an element in the array?

4. Challenges

- Experiment with the array version of the perfect score program. How do arrays appear in the debugger? Can you view everything it contains?

5. Skills and Understanding

You should now be able to answer all the "understanding" questions in the previous sections. Complete the following to get credit for the lab:

- Show the TA the following:
  - Your operational programs which complete all of the above tasks.
  - The TA may ask you to explain any section of the lab, or answer any of the "Understanding" questions.
- **Nothing** is to be submitted electronically or in hard-copy for this lab.