FULL SAIL UNIVERSITY

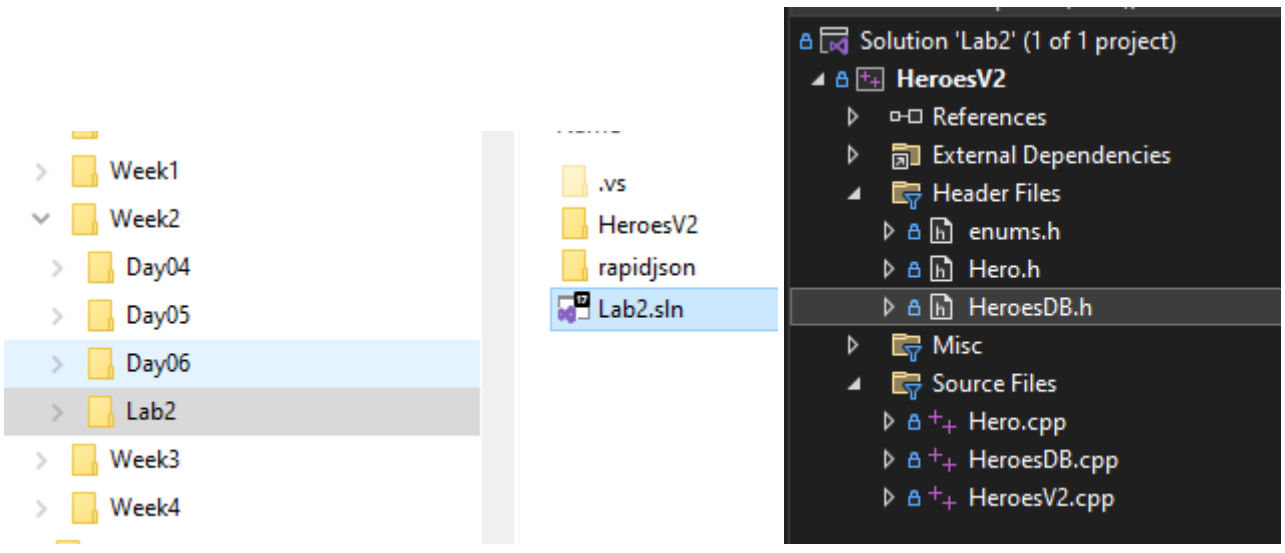
COP2334

PG2 – LAB: HEROES V2

CONTENTS

SETUP

A **C++ console application** has been provided for you in your **GitHub repo**. **Use the provided solution.**



Lab Video

Here's a video showing what the lab could look like when completed:

https://fullsailedu-my.sharepoint.com/:v:/g/personal/ggirod_fullsail_com/EaAOzNsVDgxCrk-n6mzxuzIBTPLbULtk7_z4yf1efQseRQ?e=nI35bd

How should you proceed?

The typical process for approaching the labs is…

Work on each part in order.

1. Watch the lecture video for Part A.
    a. The lectures have ***optional*** coding challenges. If you have time, then attempt them because they will help explain what you need to do for the lab.
2. Start working on Part A for the lab (Parts A-1 through A-3).
3. Once done with Part A, move on to Part B and repeat the steps above.

What to do if you need help?

Don’t struggle for too long (more than 1 hour). Reach out for help to get your questions answered.

- Take advantage of the **open lab sessions** (they are in the evening so check the announcement channel in Discord for the zoom link). Here is the schedule: https://fullsailedu.sharepoint.com/:x:/s/emergingtechstu/EbBgblouNORLsms4V4hpAYgB0Mejv59w8IVGADkVgfJW8Q?e=0l2svR
- Ask your questions in the **PG2 Discord channels**. There will be instructors and students online who can help.
- Check the availability of **Full Sail tutors**. (https://discord.com/invite/8nV8PBqq7z)

Committing your code

You MUST commit and push your code to GitHub after completing each section. For instance, part A of this lab has 2 sections. You must commit and push after completing each section which would mean you end up with 2 separate commits for part A.

If you do not commit and push for every section, you could be deducted up to 15 points for the lab.

COP2334

PART A

Lecture Videos for Part A

Day 04 Recursion and Sorting Lecture:

https://fullsailedu-my.sharepoint.com/:v:/g/personal/ggirod_fullsail_com/Eew57oZxYs5MtwNvh7-JBqkB2EWkQM0rYtS3QjNAxibe1Q?e=hoqUsI

Part A-1: MergeSort

Implement the **MergeSort** and **Merge** methods in the **HeroesDB** class**. Declare your method in the HeroesDB.h file and define the method in the HeroesDB.cpp file (**C++ Class Methods (w3schools.com)**).**

Your code must follow the pseudo-code. NOTE: you must add a parameter of type **SortBy** to both methods. You will get the user's sort by selection in part A-2 and pass it to MergeSort.

```
function MergeSort(vector m) is
    // Base case. A vector of zero or one elements is sorted, by definition.
    if length of m ≤ 1 then
        return m

    // Recursive case. First, divide the vector into equal-sized subvectors
    // consisting of the first half and second half of the vector.
    // This assumes vectors start at index 0.
    var left := empty vector
    var right := empty vector
    for I = 0 to length(m) do
        if i < (length of m)/2 then
            add m[i] to left
        else
            add m[i] to right

    // Recursively sort both subvectors.
    left := MergeSort(left)
    right := MergeSort(right)

    // Then Merge the now-sorted subvectors.
    return Merge(left, right)
```

NOTE: to Compare heroes, use the static Hero::Compare method.

EX: int compResult = **Hero::**Compare(hero1, hero2, sortBy); //returns -1 is hero1 < hero2, 0 if hero1 = hero2, or 1 is hero1 > hero2

```
function Merge(left, right) is
    var result := empty vector

    while left is not empty and right is not empty do
    {
        if first(left) ≤ first(right) then
            add first(left) to result
            remove first from left
        else
            add first(right) to result
            remove first from right
    }


    // Either left or right may have elements left; consume them.
    // (Only one of the following loops will actually be entered.)
    while left is not empty do
    {
        add first(left) to result
        remove first from left
    }
    while right is not empty do
    {
        add first(right) to result
        remove first from right
    }
    return result
```

Make sure to commit + push after completing the section.

Part A-2: SortByAttribute

Add a method called **SortByAttribute** to the **HeroesDB** class. **Declare your method in the HeroesDB.h file and define the method in the HeroesDB.cpp file (**C++ Class Methods (w3schools.com)**).** The method should have a **SortBy** parameter passed to it. Call the MergeSort method. Pass to it the _heroes vector of the class and the SortBy parameter. Print the items in the sorted vector that is returned from MergeSort.

NOTE 1: print the hero ID, selected attribute, and name (see screenshot). To get the selected attribute, call the non-static **GetSortByAttribute** method on each hero. Here's an example of calling the GetSortByAttribute method: `std::string attribute = hero.GetSortByAttribute(sortByChoice);`

NOTE 2: The SortBy value is collected already in main. Pass it to your SortByAttribute when calling it.

NAME	RETURNS	PARAMETERS	COMMENTS
SortByAttribute	nothing	SortBy	Calls the MergeSort method passing the _heroes vector and SortBy parameter. Print the items in the vector that is returned from MergeSort.

In **main** (which is in **HeroesV2.cpp**), add code to case 2 of the switch. Call your SortByAttribute method and pass the sortByChoice variable to it.

```
Choice? 2
1. Intelligence
2. Strength
3. Speed
4. Durability
5. Power
6. Combat
Sort by? 1
351:    6 Jack-Jack
231:    8 Doppelganger
510:    8 Paul Blart
396:    9 Krypto
509:    9 Parademon
609:    9 Solomon Grundy
 10:   10 Agent Bob
```

Make sure to commit + push after completing the section.

PART B

Lecture Videos for Part B

Day 05: Searching and Maps Lecture:

https://fullsailedu-my.sharepoint.com/:v:/g/personal/ggirod_fullsail_com/EXU_sa-ikFVLsstrpRP3EBABiJnlo7w5aJXfgD2j_EEd7A?e=u94LZc

Part B-1: BinarySearch

Implement the **BinarySearch** method in the **HeroesDB** class. **Declare your method in the HeroesDB.h file and define the method in the HeroesDB.cpp file (**C++ Class Methods (w3schools.com)**).**

Your code must follow the pseudo-code.

```
// initially called with low = 0, high = N-1. A is a sorted vector.
  BinarySearch(A[0..N-1], searchTerm, low, high) {
      if (high < low)
          return -1 // -1 means not found
      mid = (low + high) / 2
      if (searchTerm < A[mid])
          return BinarySearch(A, searchTerm, low, mid-1)
      else if (searchTerm > A[mid])
          return BinarySearch(A, searchTerm, mid+1, high)
      else
          return mid //the searchTerm was found
  }
```

Make sure to commit + push after completing the section.

Part B-2: FindHero

Add a method called **FindHero** to the **HeroesDB** class. **Declare your method in the HeroesDB.h file and define the method in the HeroesDB.cpp file (**C++ Class Methods (w3schools.com)**).** The method should have a string parameter for the name of the hero to find. Call the **BinarySearch** method to search the _heroes vector. Print the result. If the found index is -1, print "<insert heroName> is not found" otherwise print "<insert heroName> was found at index <insert found index>".

In **main** (which is in **HeroesV2.cpp**), add code to case 3 of the switch. Using **Input::GetString**, ask the user to enter the name to find. Call the FindHero method and pass the string the user entered.

```
Choice? 3
Please enter the name of the hero to find: Batman
Batman was found at index 51
```

```
Choice? 3
Please enter the name of the hero to find: Steve
Steve was not found.
```

Make sure to commit + push after completing the section.

Part B-3: GroupHeroes

Add a method called **GroupHeroes** to the **HeroesDB** class. **Declare your method in the HeroesDB.h file and define the method in the HeroesDB.cpp file (**C++ Class Methods (w3schools.com)**).** The method should initialize the _groupedHeroes map. Make sure to make the keys **case insensitive** (ignore the case).

You want to create a map where the keys are the first letters of the heroes and the value for each key is a vector of the heroes whose names start with that letter. EX: for the key "B", the value would contain a vector of all the heroes whose names start with B.

Loop over the heroes vector. Check if the first letter of each hero name is in the _groupedHeroes map. If not, then create a new vector, add the hero to the vector, then add the vector to the map as the value for that initial letter. If it is in the map already, then add the hero to the vector that is stored for that key.

Example:

When you start, the map will be empty.

- Start looping over the _heroes vector.

- Get the first letter of the hero's name. (EX: "A" if the hero's name is "Aquaman")
- Using the letter as a key, add the hero to vector for the key in the map. Remember that the value for the key in the map is a vector. So, you can't add the hero directly as value for the key because the value is defined as a vector. Add the hero to the vector using push_back.

Make sure to commit + push after completing the section.

Part B-4: PrintGroupCounts

Add a method called **PrintGroupCounts** to the **HeroesDB** class. **Declare your method in the HeroesDB.h file and define the method in the HeroesDB.cpp file (**C++ Class Methods (w3schools.com)**).** In the method, if _groupedHeroes is empty, call the **GroupHeroes** method.

Loop over the map and print each key and the count of the vector for each key.

In **main** (which is in **HeroesV2.cpp**), add code to case 4 to call the PrintGroupCounts method.

```
Key:  Hero Count
a: 45
b: 57
c: 33
d: 33
e: 13
f: 18
g: 20
h: 24
i: 10
j: 22
k: 19
l: 18
m: 51
n: 14
o: 8
p: 18
q: 4
r: 27
s: 66
t: 24
u: 3
v: 12
w: 16
x: 2
y: 4
z: 2
```

Make sure to commit + push after completing the section.

PART C

Lecture Videos for Part C

Day 06: Maps Lecture:

https://fullsailedu-my.sharepoint.com/:v:/g/personal/ggirod_fullsail_com/Ed7SjeJ-qM9Pljh9eayDXOgBlv2FAIKI-qnIT_jqkqhQ1g?e=4ktNYI

Part C-1: FindHeroesByLetter

Add a method called **FindHeroesByLetter** to the **HeroesDB** class. **Declare your method in the HeroesDB.h file and define the method in the HeroesDB.cpp file (**C++ Class Methods (w3schools.com)**).** The method should take a parameter for the first letter. In the method, if _groupedHeroes is empty, call the **GroupHeroes** method. Check if the letter parameter is in the map. If it is not, then print a message that no heroes were found that start with the letter. Else, loop over the vector of heroes for the key and print the ID and name.

In **main** (which is in **HeroesV2.cpp**), add code to case 5 of the switch. Using **Input::GetString**, ask the user to enter the letter to find. Call **FindHeroesByLetter** passing the string that the user enters.

```
Please enter the first letter of the heroes to find: B
 60: Bane
 61: Banshee
 62: Bantam
 63: Batgirl
 66: Batgirl IV
 68: Batgirl VI
 69: Batman
 70: Batman-Bruce
```

Make sure to commit + push after completing the section.

Part C-2: RemoveHero

Add a method called **RemoveHero** to the **HeroesDB** class. **Declare your method in the HeroesDB.h file and define the method in the HeroesDB.cpp file (**C++ Class Methods (w3schools.com)**).** The method should take a string parameter that is the **name** of the hero to remove.

In the method, if _groupedHeroes is empty, call the **GroupHeroes** method.

Check if the _groupedHeroes map contains a key with **the first letter of the name**.

- If the key is not found, print a message saying the hero was not found.
- If the key is found, then get the vector for the key. The vector is the value stored in the map for the key.
    - **call the BinarySearch** method to get the index of the hero to remove for the vector.
        - If BinarySearch returns the index, then remove the hero from the vector AND from the _heroes vector. Print that the hero was removed.
            - NOTE: if removing the hero makes the vector empty for the letter, then remove the letter (which is the key) from the map.
        - If BinarySearch returns -1 (meaning the hero is not in the vector), print a message that the hero was not found.

In **main** (which is in **HeroesV2.cpp**), add code to case 6 of the switch. Using **Input::GetString**, ask the user to enter the letter to find. Call **RemoveHero** passing the string that the user enters.

```
Please enter the name of the hero to remove: Aquaman
Aquaman was removed.
```

```
Please enter the name of the hero to remove: Bob
Bob was not found.
```

EXAMPLE:

If the user enters “Aquaman” to remove…

- Check the _groupedHeroes map for the “A” key.
- If the key “A” is NOT found, print that “Aquaman” was not found.
- If the Key “A” is found,
    - Get the value (vector<Hero>) for the key.
    - Use your BinarySearch method to search the vector for “Aquaman”.
    - If BinarySearch returns -1, print that “Aquaman” was not found.
    - Else
        - use the index that BinarySearch to remove “Aquaman” from the vector.
        - If the vector becomes empty, then remove the “A” key from the map.
        - Also remove “Aquaman” from the _heroes vector.

Make sure to commit + push after completing the section.