FULL SAIL UNIVERSITY

COP2334

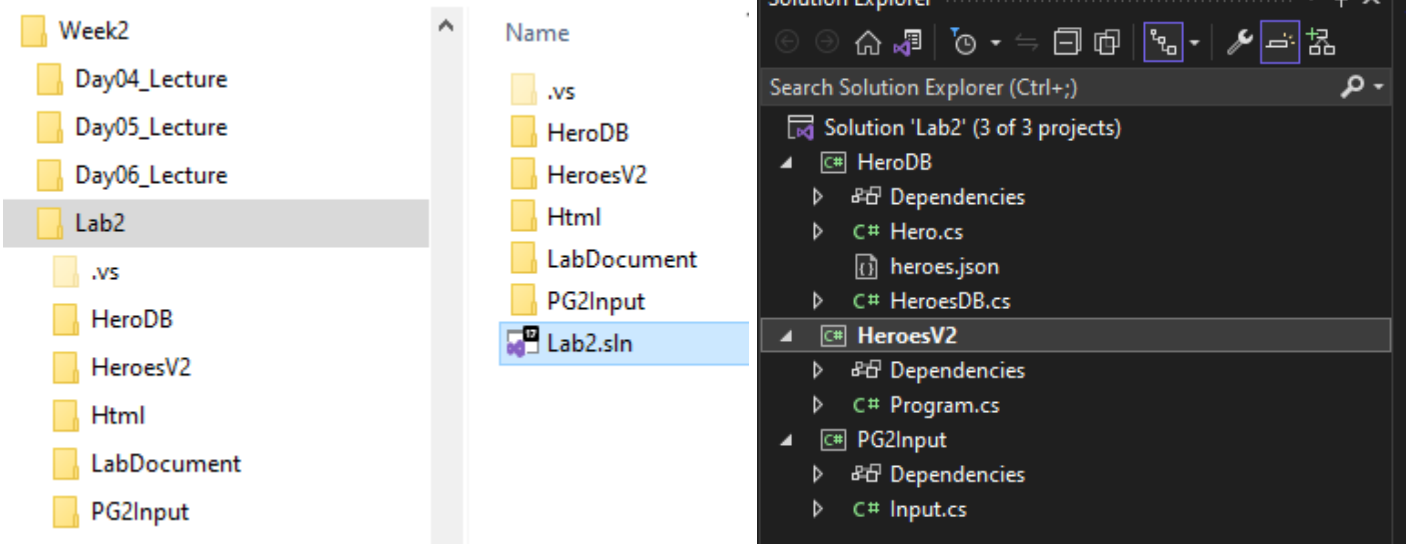
PG2 – LAB: HEROES V2

CONTENTS

FULL SAIL UNIVERSITY

SETUP

A **C# .NET Core 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://web.microsoftstream.com/video/01959e6a-381e-4992-846e-fa7aa91a8459

FULL SAIL UNIVERSITY

COP2334

PART A

Lecture Videos for Part A

SORTING LECTURE:

https://fullsailedu-my.sharepoint.com/:v:/g/personal/ggirod_fullsail_com/EfF9jZOAv-pPuJZdBNMqihABFMD5WMJEh5w_LJtiXlxrDA?e=d8YLbC

RECURSION LECTURE:

https://fullsailedu-my.sharepoint.com/:v:/g/personal/ggirod_fullsail_com/Ea-Ab-_20T1FkgFUxwmC4kMBpkxvwmpUKGN3oLDKBEU1Dg?e=tf3xNw

SEARCHING LECTURE:

https://fullsailedu-my.sharepoint.com/:v:/g/personal/ggirod_fullsail_com/EUHzQERcboRIiWiu0rSbRLEB3vipLPRU3zjjtFh4-KADQw?e=P8snKs

FULL SAIL UNIVERSITY

Part A-1: MergeSort

Implement the **MergeSort** and **Merge** methods in the **HeroesDB** class**. 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(list m) is
    // Base case. A list of zero or one elements is sorted, by definition.
    if length of m ≤ 1 then
        return m

    // Recursive case. First, divide the list into equal-sized sublists
    // consisting of the first half and second half of the list.
    // This assumes lists start at index 0.
    var left := empty list
    var right := empty list
    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 sublists.
    left := MergeSort(left)
    right := MergeSort(right)

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

NOTE: to compare heroes, use the 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 list

    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
    }
```

(continued on next page)

```
    // 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
```

Part A-2: SortByAttribute

Add a method called **SortByAttribute** to the **HeroesDB** class. The method should have a SortBy parameter passed to it. Call the MergeSort method from part A-1. Pass to it the _heroes list of the class and the SortBy parameter. Print the items in the sorted list that is returned from MergeSort.

NOTE: print the hero ID, selected attribute, and name (see screenshot). To get the selected attribute, call the **GetSortByAttribute** on each hero.

NAME	RETURNS	PARAMETERS	COMMENTS
SortByAttribute	nothing	SortBy	Calls the MergeSort method from part A-1 passing the _heroes list and SortBy parameter. Print the items in the list that is returned from MergeSort.

In Main, add code to case 2 of the switch. Call the 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
```

Part A-3: BinarySearch

Implement the **BinarySearch** method in the **HeroesDB** class. **Your code must follow the pseudo-code.**

```
// initially called with low = 0, high = N-1. A is a sorted list.
  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
  }
```

Part A-4: FindHero

Add a method called **FindHero** to the **HeroesDB** class. The method should have a string parameter for the name of the hero to find. Call the **BinarySearch** method from part A-3. 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, 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.
```

FULL SAIL UNIVERSITY

PART B

Lecture Videos for Part B

DICTIONARY LECTURE:

https://fullsailedu-my.sharepoint.com/:v:/g/personal/ggirod_fullsail_com/EYgLFeMMYXhBp9WKgfYoV5MBdnCbUoqH6wigGr1pohYdwg?e=bCwXUH

Part B-1: GroupHeroes

Add a method called **GroupHeroes** to the **HeroesDB** class. The method should initialize the _groupedHeroes dictionary. Make sure to make the keys **case insensitive** (ignore the case). HINT: look at the constructors of the Dictionary class for an overload that you can use.

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

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

Example:

When you start, the dictionary will be empty.

- Start looping over the _heroes list.
    - Get the first letter of the hero's name. (EX: "A" if the hero's name is "Aquaman")
    - Check if the first letter is a key in the dictionary. Use ContainsKey or TryGetValue.
    - If the key is found, then there is already a list in the dictionary for the key. Get the value for the key and add the hero to that list.
    - If the key is NOT found, then create a new list, add the hero to that list, then add the key and list to the dictionary.

Part B-2: PrintGroupCounts

Add a method called PrintGroupCounts to the HeroesDB class. In the method, if _groupedHeroes is null, call the **GroupHeroes** method from part B-1.

Loop over the dictionary and print each key and the count of the list for each key.

In Main, 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
```

Part B-3: FindHeroesByLetter

Add a method called **FindHeroesByLetter** to the **HeroesDB** class. The method should take a string parameter for the first letter. In the method, if _groupedHeroes is null, call the **GroupHeroes** method from part B-1. Then, check if the letter parameter is in the dictionary. If it is not, then print a message that no heroes were found that start with the letter. Else, loop over the list of heroes for the key and print the ID and name.

In Main, 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
```

FULL SAIL UNIVERSITY

COP2334

PART C

Lecture Videos for Part C

DICTIONARY LECTURE:

https://fullsailedu-my.sharepoint.com/:v:/g/personal/ggirod_fullsail_com/EYgLFeMMYXhBp9WKgfYoV5MBdnCbUoqH6wigGr1pohYdwg?e=bCwXUH

Chapters: **Removing Items** through **Removing Items Examples**.

Part C-1: RemoveHero

Add a method called **RemoveHero** to the **HeroesDB** class. The method should take a string parameter that is the name of the hero to remove.

In the method, if _groupedHeroes is null, call the **GroupHeroes** method from part B-1.

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

In Main, 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 dictionary for the “A” key. Use ContainsKey or TryGetValue.
- If the key “A” is NOT found, print that “Aquaman” was not found.
- If the Key “A” is found,
    - Get the value (List<Hero>) for the key. If you used TryGetValue earlier, you already have the list.
    - Use your BinarySearch method to search the list for “Aquaman”.
    - If BinarySearch returns -1, print that “Aquaman” was not found.
    - Else
        - use the index that BinarySearch to remove “Aquaman” from the list.
        - If the list becomes empty, then remove the “A” key from the dictionary.
        - Also remove “Aquaman” from the _heroes list.

RUBRIC

Part A

FEATURE	POINTS
Part A-1: MergeSort	15
Part A-2: SortByAttribute	10
Part A-3: BinarySearch	15
Part A-4: FindHero	10
TOTAL	50

Part B

FEATURE	POINTS
Part B-1: GroupHeroes	15
Part B-2: PrintGroupCounts	5
Part B-3: FindHeroesByLetter	15
TOTAL	35

Part C

FEATURE	POINTS
Part C-1: RemoveHero	15
TOTAL	15