Assignment Report for Assignment 4

Course and Section	CSC 340-01 (TIC)
Assignment Name	Assignment 4
Due Date and Time	07-28-2022 at 11:55 PM
First Name and Last Name	Juan Segura Rico
SFSU Email Account	jseguarico@sfsu.edu
First Name and Last Name of Teammate	N/A
SFSU Email Account of Teammate	N/A

PART A

Question Description and Analysis:

- Please change only files: LinkedBag340.cpp and Include.h, no other files.

- We are to implement 8 small additional functions and 2 helper functions to the Linked Bag.

- Our programs must produce identical output to the output in the 2 sample runs:

Asmt04_Run1.txt and Asmt04_Run2.txt

▪ Our Test 9's output must also be identical to the sample output accepts the random values.

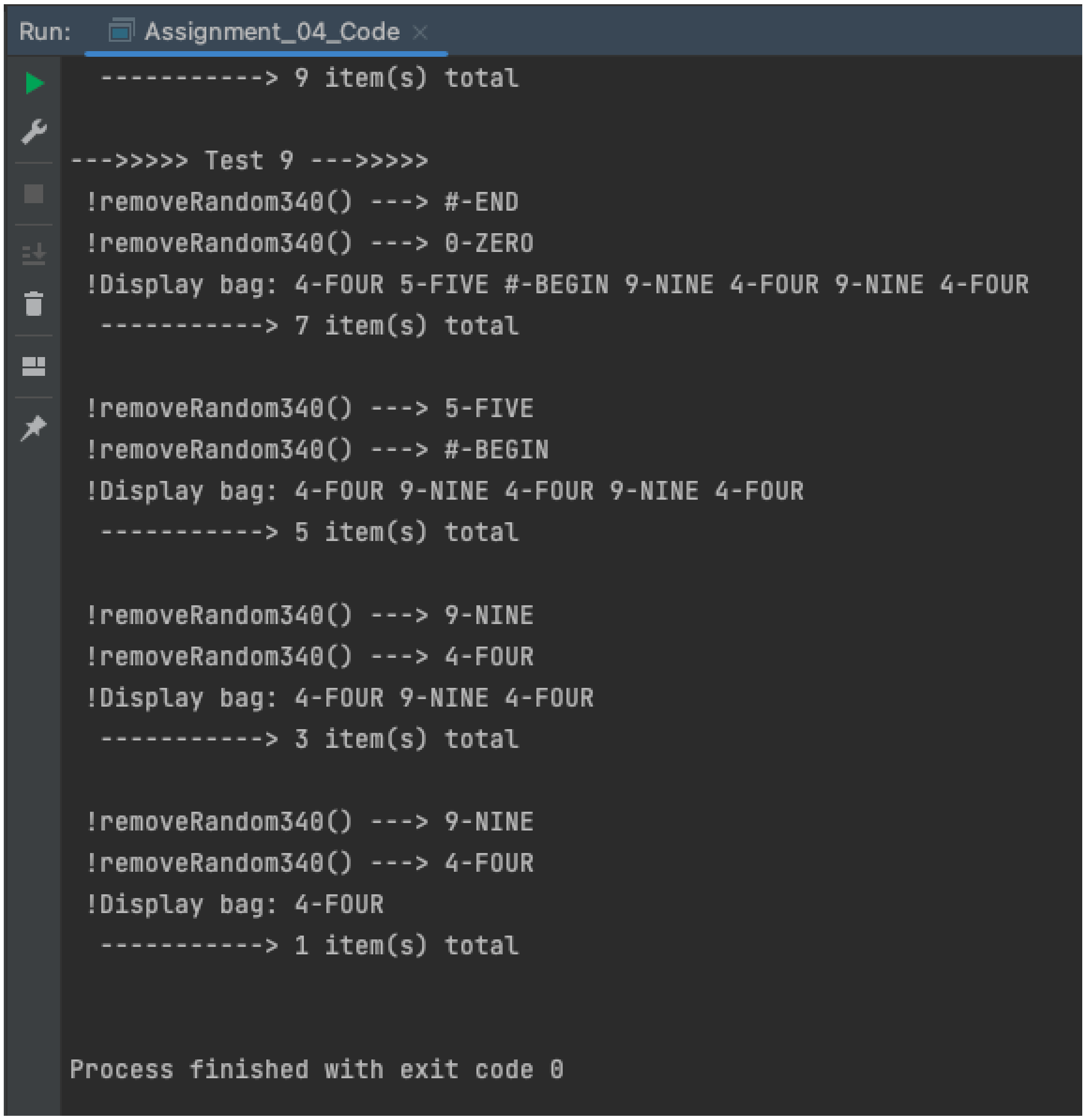
▪ Our Test 9's random values in our 2 sample runs' output must be different.

Answer:

(In screenshot section is our sample runs (both different as required))

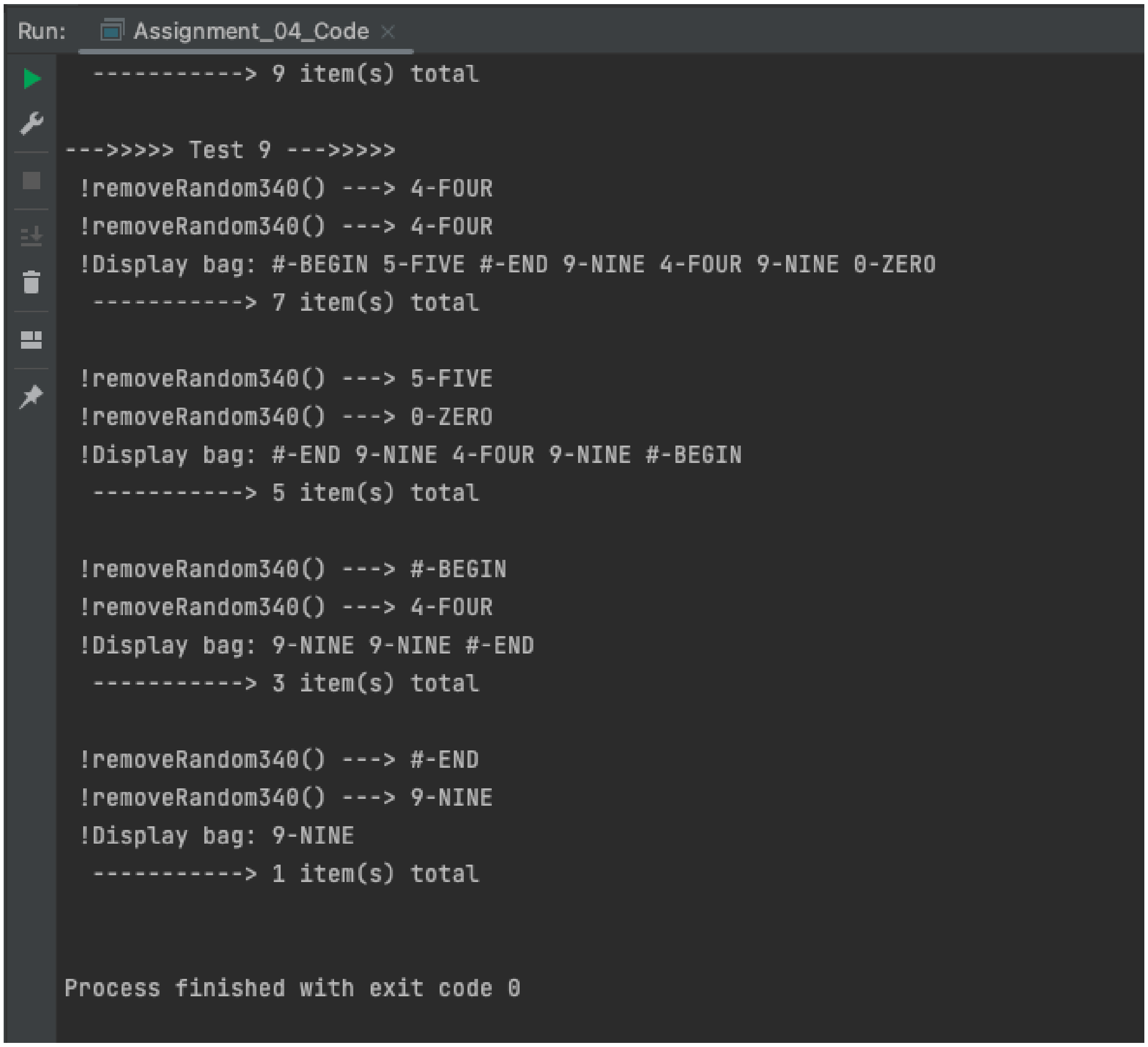
Screenshots of Outputs and Explanation:

Run 1:



```
  -----------> 9 item(s) total

--->>>>> Test 9 --->>>>>
 !removeRandom340() ---> #-END
 !removeRandom340() ---> 0-ZERO
 !Display bag: 4-FOUR 5-FIVE #-BEGIN 9-NINE 4-FOUR 9-NINE 4-FOUR
  -----------> 7 item(s) total

 !removeRandom340() ---> 5-FIVE
 !removeRandom340() ---> #-BEGIN
 !Display bag: 4-FOUR 9-NINE 4-FOUR 9-NINE 4-FOUR
  -----------> 5 item(s) total

 !removeRandom340() ---> 9-NINE
 !removeRandom340() ---> 4-FOUR
 !Display bag: 4-FOUR 9-NINE 4-FOUR
  -----------> 3 item(s) total

 !removeRandom340() ---> 9-NINE
 !removeRandom340() ---> 4-FOUR
 !Display bag: 4-FOUR
  -----------> 1 item(s) total


Process finished with exit code 0
```

Run 2:



```
  -----------> 9 item(s) total

--->>>>> Test 9 --->>>>>
 !removeRandom340() ---> 4-FOUR
 !removeRandom340() ---> 4-FOUR
 !Display bag: #-BEGIN 5-FIVE #-END 9-NINE 4-FOUR 9-NINE 0-ZERO
  -----------> 7 item(s) total

 !removeRandom340() ---> 5-FIVE
 !removeRandom340() ---> 0-ZERO
 !Display bag: #-END 9-NINE 4-FOUR 9-NINE #-BEGIN
  -----------> 5 item(s) total

 !removeRandom340() ---> #-BEGIN
 !removeRandom340() ---> 4-FOUR
 !Display bag: 9-NINE 9-NINE #-END
  -----------> 3 item(s) total

 !removeRandom340() ---> #-END
 !removeRandom340() ---> 9-NINE
 !Display bag: 9-NINE
  -----------> 1 item(s) total


Process finished with exit code 0
```

PART B

Question Description and Analysis:

- For each of the following statements, please:

▪ Explain the statement in 5 or more sentences. Please think Interviews.

▪ Create a new code experiment to demonstrate our understanding.

▪ Please remember to submit our code and document our experiment in our assignment report.

1. Deleting the same memory twice: This error can happen when two pointers address the same dynamically allocated object. If delete is applied to one of the pointers, then the object's memory is returned to the Free store. If we subsequently delete the second pointer, then the Free-store may be corrupted.
2. Use smart pointers… Objects that must be allocated with new, but you like to have the same lifetime as other objects/variables on the Run-time stack. Objects assigned to smart pointers will be deleted when the program exits that function or block.
3. Use smart pointers… Data members of classes, so when an object is deleted all the owned data is deleted as well (without any special code in the destructor).
4. Converting unique_ptr to shared_ptr is easy. Use unique_ptr first and covert unique_ptr to shared_ptr when needed.
5. Use weak_ptr for shared_ptr like pointers that can dangle.

Answer: **(Example code in screenshot section along with zip file)**

1. This statement refers to the error that can occur when attempting to deallocate memory that has already been freed. In C++, memory can be dynamically allocated using the **new** keyword, and it must be deallocated using the **delete** keyword to free up the memory for reuse. The error arises when two pointers are pointing to the same dynamically allocated memory, and both of them attempt to delete the memory. This leads to undefined behavior and can potentially result in memory corruption or program crashes.

2. Objects that must be allocated with new, but you'd like to have the same lifetime as other objects/variables on the Run-time stack. Objects assigned to smart pointers will be deleted when the program exits that function or block. In an interview, we should start by explaining that smart pointers are a type of C++ pointer that automatically manage the memory they point to. End off by giving an example if possible as well.

3. While using smart pointers, when an object is deleted, all the owned data is deleted as well (without any special code in the destructor). In an interview, the candidate could be asked to explain the concept of using smart pointers for data members of classes to automatically manage the memory of owned data when the object is deleted. The candidate should start by explaining that smart pointers, such as std::unique_ptr and std::shared_ptr, are powerful C++ features that handle automatic memory management for dynamically allocated objects. When smart pointers are used as data members of a class, they enable the class to "own" the dynamically allocated resources. As a result, when an object of that class is deleted or goes out of scope, the smart pointers will automatically handle the cleanup, ensuring that the owned data is deallocated properly.

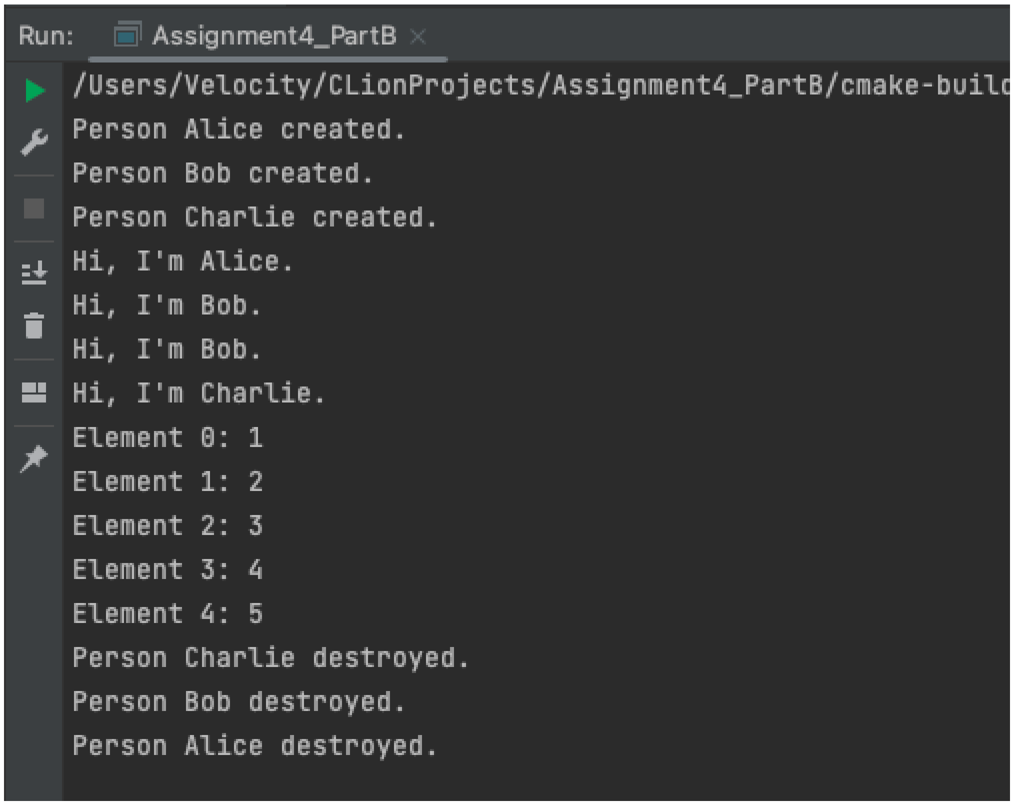
4. The reason for using std::unique_ptr first and then converting to std::shared_ptr when needed is to follow the principle of "single ownership." It is generally better to use std::unique_ptr when a resource has a single owner to ensure clear ownership and proper resource management. This conversion ensures that the resource's lifetime is still correctly managed, now with shared ownership. In an interview, we should explain it in a similar way as I mentioned above.

5. The main use of std::weak_ptr is to break potential reference cycles that can occur when using std::shared_ptr. If two or more objects have shared pointers to each other, they can create a reference cycle that will lead to memory leaks because the shared pointers will never reach a zero reference count to deallocate the memory. By using std::weak_ptr, we can have a temporary, non-owning reference that does not contribute to the resource's reference count. When in an interview, we should keep in mind these points mentioned above. We can also safely check if the resource still exists before accessing it, and the std::weak_ptr will automatically expire if the resource is deleted.

Screenshots of Outputs and Explanation:

In this program, we created a class Person, and we used std::unique_ptr to have exclusive ownership of a Person object named Alice. Then, we used std::shared_ptr to create two shared pointers, Bob and Charlie, and we demonstrated how shared ownership works. We also used std::weak_ptr with personWeak to avoid circular dependencies and prevent potential dangling pointer issues. Finally, we used std::unique_ptr to manage memory for an array of integers.

Here's the output (code attached in zip file):



```
/Users/Velocity/CLionProjects/Assignment4_PartB/cmake-buil
Person Alice created.
Person Bob created.
Person Charlie created.
Hi, I'm Alice.
Hi, I'm Bob.
Hi, I'm Bob.
Hi, I'm Charlie.
Element 0: 1
Element 1: 2
Element 2: 3
Element 3: 4
Element 4: 5
Person Charlie destroyed.
Person Bob destroyed.
Person Alice destroyed.
```

PART C (N/A)

Question Description and Analysis:

- Create a Smart Pointers version of our PART A's Linked Bag:

1. Please create a copy of our entire PART A solution and name it: PartC_SmartPointers.
2. Then go through all the files, not just LinkedBag340.cpp, and use smart pointers properly where possible.
3. In our assignment report, list the file names and the line numbers in which we use smart pointers. For each smart pointer, explain in 5 or more sentences why it is a proper use.
4. This Smart Pointers version must work properly and produce identical output like that of our PART A version.
5. In addition, please update and add destructor(s) so that the program displays more information (in addition to the output required and described above) when object(s) get destroyed.
6. Please remember to submit our code for this part. Save the code under a folder named "PartC_SmartPointers" and include this folder in the assignment submission ZIP. Please remember to document this part in the assignment report.

Answer:

N/A

Screenshots of Outputs and Explanation:

N/A

PART D - (N/A

Question Description and Analysis:

- Please create a copy of our entire PART C solution and name it: PartD_IamCreative

- This part is to show off our creative minds. Please implement a new function for Part C's LinkedBag. We need to add code to LinkedBag340.cpp and Include.h and write PartD.cpp to demonstrate how this new function works.

- Requirements, this function shall:

1. Perform one meaningful task. Please use the first paragraph of at least 5 sentences in PART D to explain why it is a meaningful task.
2. Modify the LinkedBag's content every time it runs.
3. Use Smart Pointers in its parameter list, in its implementation, and as return value(s). - Our graders expect higher quality in this part: creativity, a meaningful task, clean code, and clear documentation and report.

Answer:

N/

Screenshots of Outputs and Explanation:

N/A