Jeremiah Ruvalcaba

CSC340.02 TIC

Assignment03

Assignment Due Date : August 1, 2020 11:55pm

Assignment 03

1. In order to properly manage the memory in a program, we must declare a pointer as null then create a new instance of this pointer using the keyword new. We then delete the pointer and make sure to set it to nullptr. We Then continue on with proper deallocation of a pointer. We create a new pointer with a value then we delete the new pointer and set it to nullptr. But since we have to pointer we must use the method of double delete in order to properly manage the memory.

```
String *ptr1 = nullptr;
ptr1 = new string("ABC");

Delete ptr1; // Stale pointer

String *ptr2 = new String("DEF");
Delete ptr2;
ptr2 = nullptr

// double delete
Delete ptr1;
Delete ptr2;
```

2. When using a smart pointer we must clarify that we are no longer going to raw pointers. We consider if using Smart pointer will be more efficient that using a raw pointer. With raw pointer the program must be told to delete the object. With smart pointers a programmer doesn't have to think about deleting an object because the smart pointer will already do it for them. The smart pointer destruction of an object occurs after the equation of the code.

```
Unique_ptr<name> ptr {fname}

Ptr -> printName();
```

3. The use of smart pointer that allow objects to be allocated could be seen using the passByMove method. In the pass by move method the smart pointer that is being is used is unique. For any new object we create it will be deleted when the function is called along with the data that is

being held with a destructor. We are able to find reference to memory location and address with the unique pointer. Once the program is over then all the owned data is deleted, just like a shopping cart or deleting cookies.

```
// unique pointer is created with passByMove
Void passByMove(const unique_ptr<name> uPtr_move){
…..
}

Int main(){

passByMove(move(name_uPtr));

}
```

4. Converting a unique_ptr to shared ptr we must use unique first and convert uniqueptr to shared ptr when needed. We use make_unique() and make_share() to convert the pointers. When the unique pointer is converted, the custom delete for array objects will be managed by a shared_ptr. We also must initialize smart pointers during their declarations. In addition, we must use caution when using the get() in using smart pointers.

```
Unique_ptr<name> goofy_unique{ make_unique<name>(“Goofy”,”Dog”)};

Shared_ptr<name> goofy_shared{move(goofy_unique)};
```

5. When using Shared_ptr and weak_ptr we must know that the Shared_Ptr is used for shared ownership and weak_ptr is used for temporary ownership of an object. Shared_ptr can references the same object and detect when an object is no longer reachable by any other smart pointer. The weak_ptr does not have ownership of the object and it cannot affect the objects lifetime. Thus, a weak_ptr cannot be used to delete objects.

```
Shared_ptr<Name> goofy(new Name{“Goofy”, “Dog”};
Weak_ptr<name> goofy_wptr{ goofy };
```

Part B-

1.
```
-->>>>> Test 1 --->>>>>
!add()...     #-END 5-FIVE 4-FOUR 4-FOUR 3-THREE 2-TWO 1-ONE 0-ZERO #-BEGIN
!Display bag: #-BEGIN 0-ZERO 1-ONE 2-TWO 3-THREE 4-FOUR 4-FOUR 5-FIVE #-END
 -----------> 9 item(s) total
```

2.
```
-->>>>> Test 2 --->>>>>
!removeSecondNode340()...
!removeSecondNode340()...
!Display bag: #-BEGIN 2-TWO 3-THREE 4-FOUR 4-FOUR 5-FIVE #-END
 -----------> 7 item(s) total

!removeSecondNode340()...
!removeSecondNode340()...
!Display bag: #-BEGIN 4-FOUR 4-FOUR 5-FIVE #-END
 -----------> 5 item(s) total
```

3.
```
->>>>> Test 3 --->>>>>
addEnd340()...
addEnd340()...
Display bag: #-BEGIN 4-FOUR 4-FOUR 5-FIVE #-END 9-NINE 4-FOUR
-----------> 7 item(s) total

addEnd340()...
addEnd340()...
Display bag: #-BEGIN 4-FOUR 4-FOUR 5-FIVE #-END 9-NINE 4-FOUR 9-NINE 0-ZERO
-----------> 9 item(s) total
```

4.
```
-->>>>> Test 4 --->>>>>
!getCurrentSize340Iterative - Iterative...
 ---> Current size: 9
!Display bag: #-BEGIN 4-FOUR 4-FOUR 5-FIVE #-END 9-NINE 4-FOUR 9-NINE 0-ZERO
 -----------> 9 item(s) total
```

5.
```
-----------> 9 item(s) total

-->>>>> Test 5 --->>>>>
!getCurrentSize340Recursive() - Recursive...
 ---> Current size: 9
!Display bag: #-BEGIN 4-FOUR 4-FOUR 5-FIVE #-END 9-NINE 4-FOUR 9-NINE 0-ZERO
 -----------> 9 item(s) total
```

6.
```
-->>>>> Test 6 --->>>>>
!getCurrentSize340RecursiveNoHelper() - Recursive...
 ---> Current size: 9
!Display bag: #-BEGIN 4-FOUR 4-FOUR 5-FIVE #-END 9-NINE 4-FOUR 9-NINE 0-ZERO
 -----------> 9 item(s) total
```

7.
```
--->>>>> Test 7 --->>>>>
 !getFrequencyOf()...
  ---> 0-ZERO:  1
  ---> 1-ONE:   0
  ---> 2-TWO:   0
  ---> 4-FOUR:  3
  ---> 9-NINE:  2
 !Display bag: #-BEGIN 4-FOUR 4-FOUR 5-FIVE #-END 9-NINE 4-FOUR 9-NINE 0-ZERO
  -----------> 9 item(s) total

 !getFrequencyOf340Recursive() - Recursive...
  ---> 0-ZERO:  1
  ---> 1-ONE:   0
  ---> 2-TWO:   0
  ---> 4-FOUR:  3
  ---> 9-NINE:  2
 !Display bag: #-BEGIN 4-FOUR 4-FOUR 5-FIVE #-END 9-NINE 4-FOUR 9-NINE 0-ZERO
  -----------> 9 item(s) total
```

8.
```
-->>>>> Test 8 --->>>>>
!getFrequencyOf340RecursiveNoHelper() - Recursive...
 ---> 0-ZERO:  1
 ---> 1-ONE:   1
 ---> 2-TWO:   1
 ---> 4-FOUR:  4
 ---> 9-NINE:  3
!Display bag: #-BEGIN 4-FOUR 4-FOUR 5-FIVE #-END 9-NINE 4-FOUR 9-NINE 0-ZERO
 -----------> 9 item(s) total
```

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