# Assignment 7: Efficient Memory Management: Introducing Smart Pointers in a Doubly Linked List

In this Assignment, you will rewrite the attached code using smart pointers as taught in the class. C++11 introduces smart pointers that simplify memory management issues such as dangling pointers or memory leaks by automatic deletion. Smart pointers also facilitate sharing of objects in the memory by keeping track of the reference count for a particular heap-allocated object. You will implement a doubly linked list using smart pointers. As demonstrated in Figure 1, in a doubly linked, each intermediate node has a pointer to the next node and the previous node.

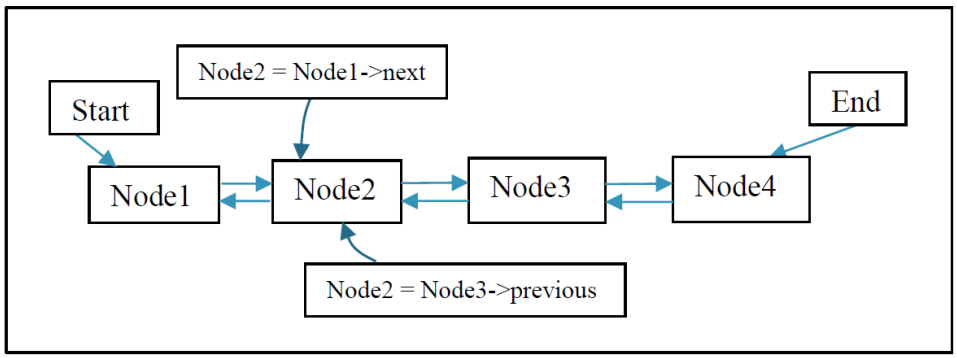


Figure 1: A Doubly Linked List.

### Introducing shared pointers and weak pointers in a doubly linked list

Among different types of smart pointers, shared pointer as defined by **shared_ptr** class is a template that is a wrapper around an object allocated onto the heap. The wrapper uses reference counting to track how many other pointers reference the object. The counter starts at zero. Whenever a new variable references the object, the counter is incremented by one. The counter is decremented by one every time a shared pointer ceases to reference the object.

Weak pointer as defined by **weak_ptr** class is also a smart pointer but it does not control the lifetime of the object to which it points to. Instead, a **weak_ptr** points to an object that is managed by a **shared_ptr**. If that **shared_ptr** ceases to reference the object, that object will be deleted even if there are **weak_ptr**s pointing to it.

### Assignment Specification:

However, implementing a doubly linked list only with **shared_ptr** can be troublesome. If both next and previous nodes are implemented with shared pointers as defined by the **shared_ptr** class, a circular reference to the same node can happen. For example, in Figure 1, "**Node2**" is the next node of "**Node1**" as well the previous node of "**Node3**". This will prevent the reference count for Node2 from being zero and memory will not be reclaimed.

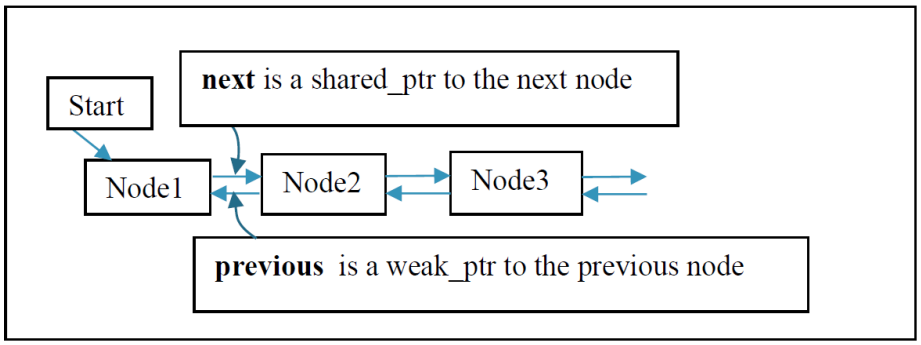


Figure 2: Use of shared_ptr and weak_ptr for the next and previous pointers respectively in a doubly linked list.

- Hence, in this assignment, in the "**Node**" class, a pointer to the next node will be represented by a **shared_ptr** and a **weak_ptr** will reference the previous node as demonstrated in Figure 2. Thus, when the reference count for the shared pointer will go down to zero, any reference via weak pointers will be automatically destroyed. You will need to write **Node.h**, **Node.cpp**, and the source file **Doublylinkedlist.cpp** using smart pointers.
- When you access the weak pointer, you may want to use the **lock()**, **expired()** functions to check whether the reference to the weak pointer has already been deleted as mentioned in the previous section.
- Every occurrence of a pointer to the Node class needs to be replaced with shared_ptr<Node<T>> or weak_ptr<Node<T>> as mentioned before.
- After you complete the code, the same output will be generated with the revised code with smart pointers.
- After you implement the codes with smart pointer, you will find that with smart pointers, you do not need to explicitly call deleteList. All nodes will be automatically deleted as they go out of scope.
- Run your code with at least two different data types, i.e., int, double, string, and so on and save the output in a pdf file named output.pdf.

**Submission:**

Your submission should include **Node.h**, **Node.cpp**, **DoublyLinkedList.cpp**, and a pdf file named **output.pdf**.

Compile and execute your code with g++ or the Microsoft compiler.

Place your solution in a zipped file named with your last name followed by the first initial of your first name followed by 7 (i.e., **YasminS7.zip**) and submit the solution via canvas.

Submission deadline is Tuesday, June 11, 11:59 pm.

This assignment weighs 5% of the total grade.