5ACW – Session 16 – Linked Lists - Lab Exercises

Lab 01- Node Chains

1. Create a new Solution in Visual Studio and add a new Console Project called NodeChains.
2. Add a new class called Node with the following properties as shown below:

```
public class Node
{
    public int Value { get; set; }
    public Node Next { get; set; }
}
```

3. Open the projects Program.Main(..), and add create 3 instances of the Node class as shown below:

```
Node first = new Node { Value = 3 };

Node middle = new Node { Value = 5 };
```

4. Chain the nodes together as shown below:

```
first.Next = middle;
```

5. Create a third Node object and chain it to the last node in th previous node chain as shown below:

```
Node last = new Node { Value = 7 };

middle.Next = last;
```

6. Create the following method called PrintList that will enumerate through all the nodes and display all its values. Call this method in the Program.Main(..)- see below:

```
private static void PrintList(Node node)
{
    while (node != null)
    {
        Console.WriteLine(node.Value);
        node = node.Next;
    }
}
```

7. Run the console application and check the output.

Lab 02- Linked List

1. Add a new Console project called LinkedListDemo to the Solution you created in the previous Lab.
2. Create a class called LinkedListNode and add the following code:

```
public class LinkedListNode<T>
{
    public LinkedListNode(T value)
    {
        Value = value;
    }

    public T Value { get; set; }

    public LinkedListNode<T> Next { get; set; }
}
```

3. Create a new class called LinkedList and add the following properties and constructor as shown below:

```
public class LinkedList<T> :
    System.Collections.Generic.ICollection<T>
{

    public LinkedListNode<T> Head
    {
        get;
        private set;
    }


    public LinkedListNode<T> Tail
    {
        get;
        private set;
    }
```

4. Add the following method called AddFirst(..) to add a value of type <T> that is used to create an instance of a new Node and adds it to the start of the list: as shown below:

```
public void AddFirst(T value)
{
    AddFirst(new LinkedListNode<T>(value));
}
```

5. Add the following overloaded method called AddFirst(..) to add a Node instance to the list: as shown below:

```
public void AddFirst(LinkedListNode<T> node)
{
    // Save off the head node so we don't lose it
    LinkedListNode<T> temp = Head;

    // Point head to the new node
    Head = node;

    // Insert the rest of the list behind the head
    Head.Next = temp;

    Count++;

    if (Count == 1)
    {
        // if the list was empty then Head and Tail should
        // both point to the new node.
        Tail = Head;
    }
}
```

6. Add the following method AddLast(..) to add a value of type <T> that adds a new instance of a Node to the end of the list as shown below:

```
public void AddLast(T value)
{
    AddLast(new LinkedListNode<T>(value));
}
```

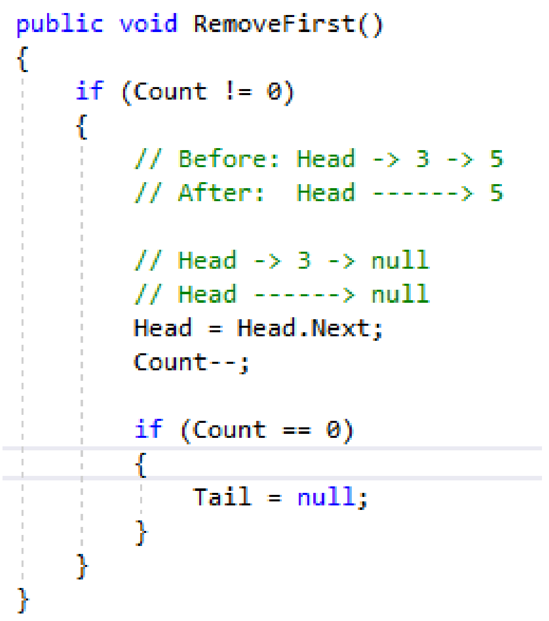
7. Add the following overloaded method called AddLast(..) to add a Node instance to the end of the list, as shown below:

```
public void AddLast(LinkedListNode<T> node)
{
    if (Count == 0)
    {
        Head = node;
    }
    else
    {
        Tail.Next = node;
    }

    Tail = node;

    Count++;
}
```

8. Add the following method RemoveFirst() that removes the first Node instance in the list, as shown below:

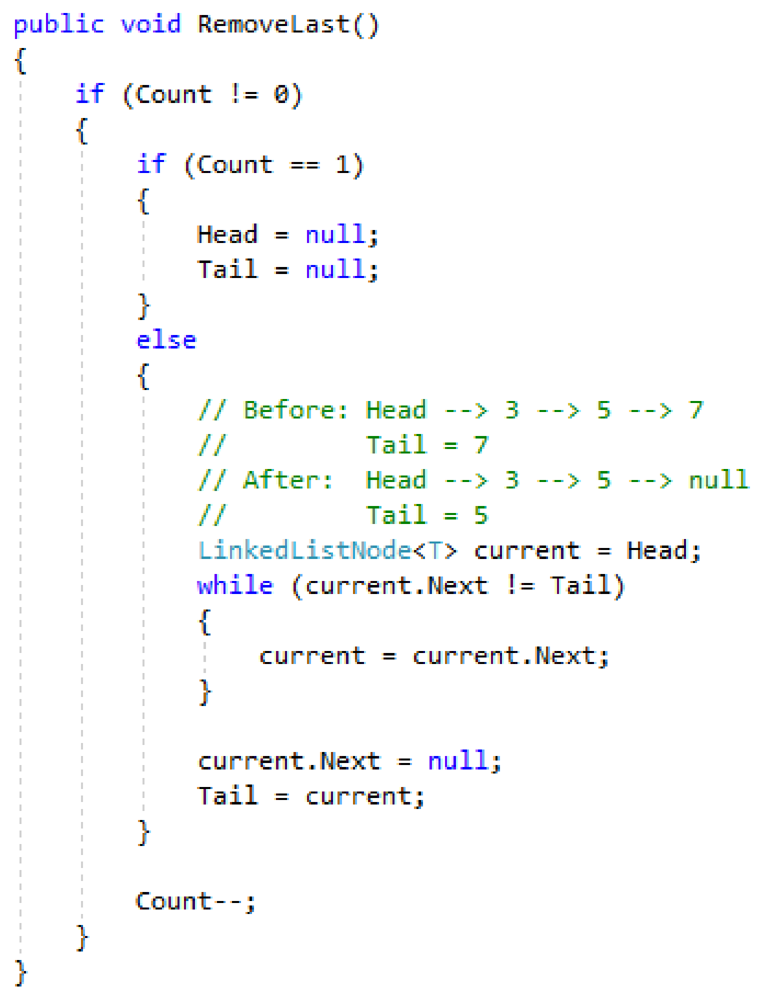


```
public void RemoveFirst()
{
    if (Count != 0)
    {
        // Before: Head -> 3 -> 5
        // After:  Head ------> 5

        // Head -> 3 -> null
        // Head ------> null
        Head = Head.Next;
        Count--;

        if (Count == 0)
        {
            Tail = null;
        }
    }
}
```

9. Add the following method Remove Last() that removes the first Node instance in the list, as shown below:



```
public void RemoveLast()
{
    if (Count != 0)
    {
        if (Count == 1)
        {
            Head = null;
            Tail = null;
        }
        else
        {
            // Before: Head --> 3 --> 5 --> 7
            //         Tail = 7
            // After:  Head --> 3 --> 5 --> null
            //         Tail = 5
            LinkedListNode<T> current = Head;
            while (current.Next != Tail)
            {
                current = current.Next;
            }

            current.Next = null;
            Tail = current;
        }

        Count--;
    }
}
```

10. You are now going to implement the methods of the ICollection interface which this class implements. Let's start with add the Count property to your code (after the RemoveLast method) as shown below:

```
public int Count
{
    get;
    private set;
}
```

11. Implement the Add(..) to add values to the list (will add this at the beginning of the list) as shown below:

```
public void Add(T item)
{
    AddFirst(item);
}
```

12. Implement the Contains (..) that return true if the items is found, false otherwise.

```
public bool Contains(T item)
{
    LinkedListNode<T> current = Head;
    while (current != null)
    {
        if (current.Value.Equals(item))
        {
            return true;
        }

        current = current.Next;
    }

    return false;
}
```

13. Implement the CopyTo (..) to copy the list values into an array at a particular index value, as shown below:

```
public void CopyTo(T[] array, int arrayIndex)
{
    LinkedListNode<T> current = Head;
    while (current != null)
    {
        array[arrayIndex++] = current.Value;
        current = current.Next;
    }
}
```

14. Implement the IsReadOnly( ) to make the list read only or not, as follows:

```
public bool IsReadOnly
{
    get
    {
        return false;
    }
}
```

15. Implement the Remove(..) to return true if item was found and removed, false otherwise to copy the list values into an array at a particular index value, as shown below:

```
public bool Remove(T item)
{
    LinkedListNode<T> previous = null;
    LinkedListNode<T> current = Head;

    // 1: Empty list - do nothing
    // 2: Single node: (previous is null)
    // 3: Many nodes
    //    a: node to remove is the first node
    //    b: node to remove is the middle or last

    while (current != null)
    {
        if (current.Value.Equals(item))
        {
            // it's a node in the middle or end
            if (previous != null)
            {
                // Case 3b

                // Before: Head -> 3 -> 5 -> null
                // After:  Head -> 3 ------> null
                previous.Next = current.Next;

                // it was the end - so update Tail
                if (current.Next == null)
                {
                    Tail = previous;
                }

                Count--;
            }
            else
            {
                // Case 2 or 3a
                RemoveFirst();
            }

            return true;
        }

        previous = current;
        current = current.Next;
    }

    return false;
}
```

16. Implement the GetEnumerator ( ) to enumerate over the linked list values from Head to Tail as follows:

```
System.Collections.Generic.IEnumerator<T> System.Collections.Generic.IEnumerable<T>.GetEnumerator()
{
    LinkedListNode<T> current = Head;
    while (current != null)
    {
        yield return current.Value;
        current = current.Next;
    }
}
```

17. Lastly, implement the Clear( ) to remove all nodes from the list as follows:

```
public void Clear()
{
    Head = null;
    Tail = null;
    Count = 0;
}
```

18. Test your linked list implementation in the Program.Main (..) , by adding three integers to the lists appropriate Add ( ) . Also create three other LinkedListNode instances and add them to your list using the appropriate Add (...) Now enumerate through the list to display the node values.
    Test if the other methods work by calling the appropriate methods followed by displaying the items in the list after each operation.

Lab 03- Doubly Linked List

1. Add a new Console project called DoublyLinkedListDemo to the Solution you created in the previous Lab.
2. Create a class called DoublyLinkedListNode and add the following code:

```
public class LinkedListNode<T>
{
    public LinkedListNode(T value)
    {
        Value = value;
    }

    public T Value { get; set; }

    public LinkedListNode<T> Next { get; set; }

    public LinkedListNode<T> Previous { get; set; }
}
```

3. Create a new class called DoublyLinkedList and add the following properties and constructor as shown below:

```
public class DoublyLinkedList<T> :
    System.Collections.Generic.ICollection<T>
{

    public LinkedListNode<T> Head
    {
        get;
        private set;
    }

    public LinkedListNode<T> Tail
    {
        get;
        private set;
    }
```

4. Implement the two overloaded AddFirst ( ) as shown below:

```
public void AddFirst(T value)
{
    AddFirst(new LinkedListNode<T>(value));
}

public void AddFirst(LinkedListNode<T> node)
{
    // Save off the head node so we don't lose it
    LinkedListNode<T> temp = Head;

    // Point head to the new node
    Head = node;

    // Insert the rest of the list behind the head
    Head.Next = temp;

    if (Count == 0)
    {
        // if the list was empty then Head and Tail should
        // both point to the new node.
        Tail = Head;
    }
    else
    {
        // Before: Head -------> 5 <-> 7 -> null
        // After:  Head -> 3 <-> 5 <-> 7 -> null

        // temp.Previous was null, now Head
        temp.Previous = Head;
    }

    Count++;
}
```

5. Implement the two overloaded AddLast( ) methods as shown below:

```
public void AddLast(T value)
{
    AddLast(new LinkedListNode<T>(value));
}
public void AddLast(LinkedListNode<T> node)
{
    if (Count == 0)
    {
        Head = node;
    }
    else
    {
        Tail.Next = node;

        // Before: Head -> 3 <-> 5 -> null
        // After:  Head -> 3 <-> 5 <-> 7 -> null
        // 7.Previous = 5
        node.Previous = Tail;
    }

    Tail = node;
    Count++;
}
```

6. Implement the RemoveFirst( ), and RemoveLast( ) methods as follows:

```
public void RemoveFirst()
{
    if (Count != 0)
    {
        // Before: Head -> 3 <-> 5
        // After:  Head -------> 5

        // Head -> 3 -> null
        // Head ------> null
        Head = Head.Next;

        Count--;

        if (Count == 0)
        {
            Tail = null;
        }
        else
        {
            // 5.Previous was 3, now null
            Head.Previous = null;
        }
    }
}
```

```
public void RemoveLast()
{
    if (Count != 0)
    {
        if (Count == 1)
        {
            Head = null;
            Tail = null;
        }
        else
        {
            // Before: Head --> 3 --> 5 --> 7
            //         Tail = 7
            // After:  Head --> 3 --> 5 --> null
            //         Tail = 5
            // Null out 5's Next pointer
            Tail.Previous.Next = null;
            Tail = Tail.Previous;
        }

        Count--;
    }
}
```

7. Add the Count property after the above method as follows:

```
public int Count
{
    get;
    private set;
}
```

8. Implement the Add( ) and Contains( ) methods as shown below:

```
public void Add(T item)
{
    AddFirst(item);
}

public bool Contains(T item)
{
    LinkedListNode<T> current = Head;
    while (current != null)
    {
        // Head -> 3 -> 5 -> 7
        // Value: 5
        if (current.Value.Equals(item))
        {
            return true;
        }

        current = current.Next;
    }

    return false;
}
```

9. Implement the CopyTo(..) and IsReadOnly ( ) methods as follows:

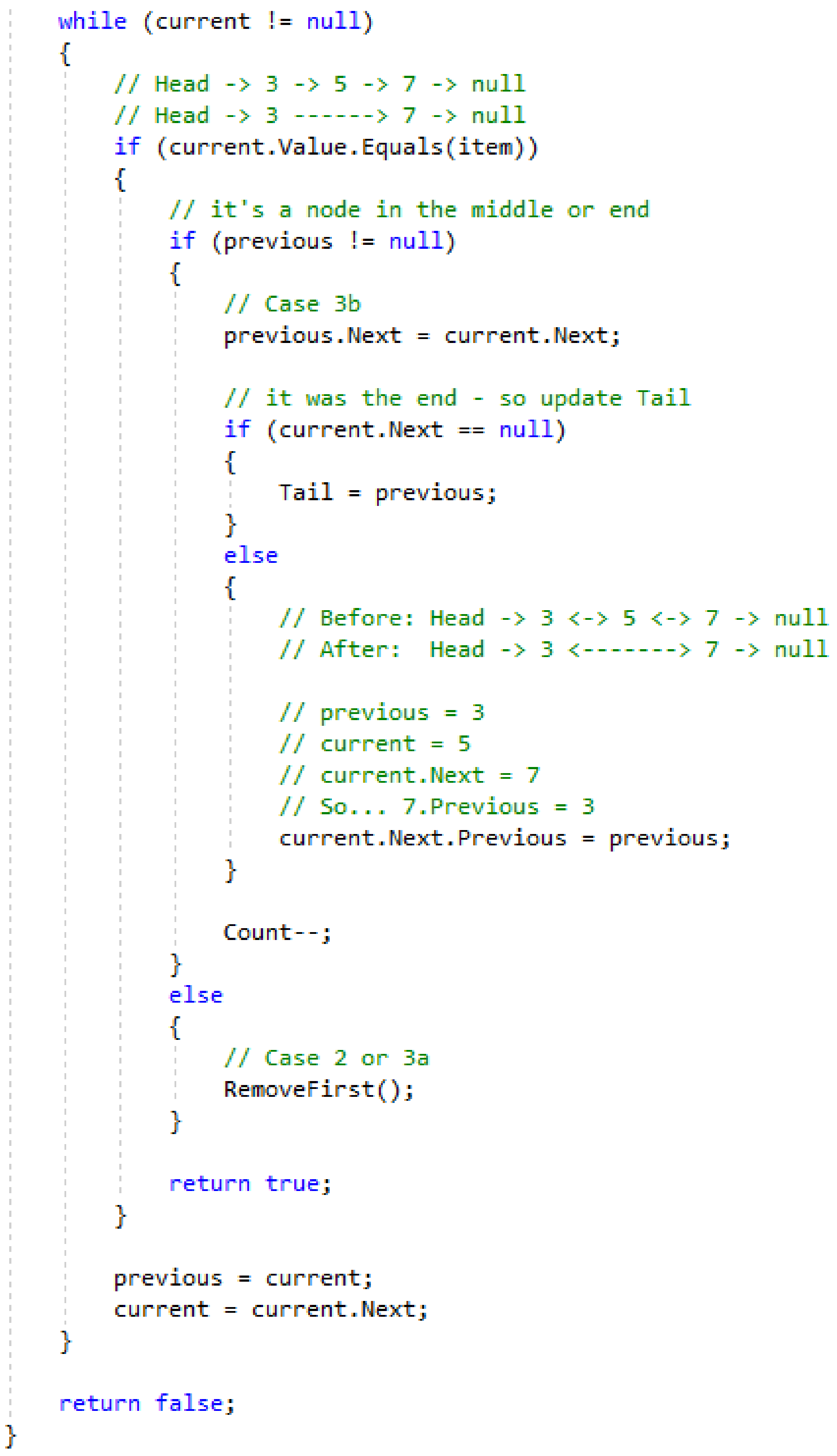
```
public void CopyTo(T[] array, int arrayIndex)
{
    LinkedListNode<T> current = Head;
    while (current != null)
    {
        array[arrayIndex++] = current.Value;
        current = current.Next;
    }
}
```

```
public bool IsReadOnly
{
    get
    {
        return false;
    }
}
```

10. Implement the Remove(..) method as follows:

```
public bool Remove(T item)
{
    LinkedListNode<T> previous = null;
    LinkedListNode<T> current = Head;

    // 1: Empty list - do nothing
    // 2: Single node: (previous is null)
    // 3: Many nodes
    //    a: node to remove is the first node
    //    b: node to remove is the middle or last
```



```
    while (current != null)
    {
        // Head -> 3 -> 5 -> 7 -> null
        // Head -> 3 ------> 7 -> null
        if (current.Value.Equals(item))
        {
            // it's a node in the middle or end
            if (previous != null)
            {
                // Case 3b
                previous.Next = current.Next;

                // it was the end - so update Tail
                if (current.Next == null)
                {
                    Tail = previous;
                }
                else
                {
                    // Before: Head -> 3 <-> 5 <-> 7 -> null
                    // After:  Head -> 3 <-------> 7 -> null

                    // previous = 3
                    // current = 5
                    // current.Next = 7
                    // So... 7.Previous = 3
                    current.Next.Previous = previous;
                }

                Count--;
            }
            else
            {
                // Case 2 or 3a
                RemoveFirst();
            }

            return true;
        }

        previous = current;
        current = current.Next;
    }

    return false;
}
```

11. Implement the two GetEnumerator( ) methods as shown below:

```
System.Collections.Generic.IEnumerator<T> System.Collections.Generic.IEnumerable<T>.GetEnumerator()
{
    LinkedListNode<T> current = Head;
    while (current != null)
    {
        yield return current.Value;
        current = current.Next;
    }
}
```

```
System.Collections.IEnumerator System.Collections.IEnumerable.GetEnumerator()
{
    return ((System.Collections.Generic.IEnumerable<T>)this).GetEnumerator();
}
```

12. Implement the Clear(..) method as follows:

```
public void Clear()
{
    Head = null;
    Tail = null;
    Count = 0;
}
```

Lab 04- Using LinkedList from .NET library

1. Add a new Console project called DotNetLinkedList to the solution created in Lab 01.
2. Open the Program.Main (..) and add the following code to create a Linked List from the .NET library as follows:

```
static void Main(string[] args)
{
    LinkedList<int> list = new LinkedList<int>();
    list.AddLast(3);
    list.AddLast(5);
    list.AddLast(7);

    foreach (int value in list)
    {
        Console.WriteLine(value);
    }
}
```

3. Run the application and observe the output.