```
using System;
using System.CodeDom;
using System.CodeDom.Compiler;
using System.Collections.Generic;
using System.Runtime.ExceptionServices;
using System.Runtime.InteropServices.WindowsRuntime;
using System.Security.Policy;

namespace Unit4
{
    public static class NodeUtils
    {
        public static Node<T> CreateListFromArray<T>(T[] arr)
        {
            Node<T> head = null;

            for (int i = arr.Length-1; i >= 0; i--)
            {
                head = new Node<T>(arr[i], head);
            }

            return head;
        }

        public static Node<T> CreateListFromArrayR<T>(T[] arr, int i)
        {
            Node<T> head = new Node<T>(arr[i]);

            if (i <= 0)
                return head;

            head = new Node<T>(arr[i - 1], head);

            return CreateListFromArrayR(arr, i - 1);
        }

        public static void PrintList<T>(Node<T> l)
        {
            while(l != null)
            {
                Console.Write(l.GetValue() + "-->");
                l = l.GetNext();
            }
            Console.WriteLine();
        }

        public static void PrintListR<T>(Node<T> l)
        {
            if (l == null)
```

```
                return;

            Console.Write(l + "-->");

            PrintList(l.GetNext());
        }

        public static Node<T> CloneList<T>(Node<T> l)
        {
            if (l == null)
                return null;

            Node<T> new_head = new Node<T>(l.GetValue());
            Node<T> pos = new_head;

            while(l.GetNext() != null)
            {
                l = l.GetNext();
                pos.SetNext(new Node<T>(l.GetValue()));
                pos = pos.GetNext();
            }

            return new_head;
        }

        public static bool CompareList(Node<int> left, Node<int> right)
        {
            bool equal = true;

            while(left != null && right != null && equal)
            {
                if ((left.GetNext() == null && right.GetNext() != null) || (left.GetNext() != null && right.GetNext() == null))
                    equal = false;

                equal = left.GetValue() == right.GetValue() && equal;

                left = left.GetNext();
                right = right.GetNext();
            }

            return equal;
        }

        public static bool CompareListR(Node<int> left, Node<int> right)
        {
            if (left == null && right == null)
                return true;

```

```
            if ((left.GetNext() == null && right.GetNext() != null) ||
              (left.GetNext() != null && right.GetNext() == null))
                return false;

            return left.GetValue() == right.GetValue() && CompareListR
              (left.GetNext(), right.GetNext());
        }

        public static int CountList<T>(Node<T> lst)
        {
            int cnt = 0;

            while(lst != null)
            {
                cnt++;
                lst = lst.GetNext();
            }

            return cnt;
        }

        public static int CountListR<T>(Node<T> lst)
        {
            if (lst == null)
                return 0;

            return 1 + CountListR(lst.GetNext());
        }

        public static int SumList(Node<int> lst)
        {
            int sum = 0;

            while (lst != null)
            {
                sum += lst.GetValue();
                lst = lst.GetNext();
            }

            return sum;
        }

        public static int SumListR(Node<int> lst)
        {
            if (lst == null)
                return 0;

            return lst.GetValue() + SumListR(lst.GetNext());
        }
```

```
        public static bool IsExist(Node<int> lst, int val)
        {
            bool found = false;

            while (!found && lst != null)
            {
                found = lst.GetValue() == val;
                lst = lst.GetNext();
            }

            return found;
        }

        public static bool IsExistR(Node<int> lst, int val)
        {
            if (lst == null)
                return false;

            return lst.GetValue() == val || IsExistR(lst.GetNext(), val);
        }

        public static int FindMax(Node<int> lst)
        {
            int max = int.MinValue;

            while(lst != null)
            {
                if(lst.GetValue() > max)
                {
                    max = lst.GetValue();
                }
                lst = lst.GetNext();
            }

            return max;
        }

        public static int FindMaxR(Node<int> lst)
        {
            if (lst == null)
                return int.MinValue;

            return Math.Max(lst.GetValue(), FindMaxR(lst.GetNext()));
        }

        public static void AbsValue(Node<int> lst)
        {
            while(lst != null)
```

```
            {
                lst.SetValue(Math.Abs(lst.GetValue()));
                lst = lst.GetNext();
            }
        }

        public static void AbsValueR(Node<int> lst)
        {
            if (lst == null)
                return;

            lst.SetValue(Math.Abs(lst.GetValue()));

            AbsValue(lst.GetNext());
        }

        public static Node<T> GetNodeRef<T>(Node<T> lst, int loc)
        {
            Node<T> r = null; // refernce

            int save_i = 1;

            for (int i = 1; i <= loc && lst != null; i++)
            {
                r = lst;
                save_i = i;
                lst = lst.GetNext();
            }

            if (save_i < loc)
                return null;
            return r;
        }

        public static Node<T> GetNodeRefR<T>(Node<T> lst, int loc)
        {
            if (loc == 0)
                return lst;

            if (lst == null)
                return null;

            return GetNodeRef<T>(lst.GetNext(), loc - 1);
        }

        public static bool IsSorted(Node<int> lst)
        {
            bool is_sorted = true;

```

```
            while (lst.GetNext() != null && is_sorted)
            {
                is_sorted = lst.GetValue() <= lst.GetNext().GetValue();
                lst = lst.GetNext();
            }

            return is_sorted;

        }

        public static bool IsSortedR(Node<int> lst)
        {
            if (lst.GetNext() == null)
                return true;

            return lst.GetValue() <= lst.GetNext().GetValue() && IsSortedR (lst.GetNext());
        }

        public static int CountSeqs(Node<int> lst, int n)
        {
            int seq_cnt = 0;
            int curr_seq_len = 1;

            while(lst != null)
            {
                if (lst.GetValue() ==  n)
                {
                    if (curr_seq_len == 1)
                        seq_cnt++;

                    curr_seq_len++;
                }

                else
                {
                    curr_seq_len = 1;
                }

                lst = lst.GetNext();
            }

            return seq_cnt;
        }

        public static void PrintAtoB(Node<int> lst, int a, int b)
        {
            for (int i = 0; i < a-1; i++)
            {
```

```
                lst = lst.GetNext();
            }

            for (int i = 0; i < b-a+1; i++)
            {
                Console.Write(lst + "-->");
                lst = lst.GetNext();
            }
        }

        public static Node<int> RemoveDuplicates(Node<int> lst)
        {
            Node<int> new_lst = new Node<int>(lst.GetValue());
            lst = lst.GetNext();
            Node<int> head = new_lst;

            while(lst != null)
            {
                if (!IsExist(head, lst.GetValue()))
                {
                    new_lst.SetNext(new Node<int>(lst.GetValue()));
                    new_lst = new_lst.GetNext();
                }

                lst = lst.GetNext();
            }

            return head;
        }

        public static bool BalancedList(Node<int> lst)
        {
            int sum = 0;
            int cnt = 0;

            Node<int> save_lst = lst;

            while(lst != null)
            {
                sum += lst.GetValue();
                cnt++;
                lst = lst.GetNext();
            }

            double avg = (double)(sum) / cnt;

            lst = save_lst;

            int aboveAvg = 0;
```

```
            int belowAvg = 0;

            while(lst != null)
            {
                if (lst.GetValue() > avg)
                    aboveAvg++;

                else if (lst.GetValue() < avg)
                    belowAvg++;

                lst = lst.GetNext();
            }

            return aboveAvg == belowAvg;

        }

        public static (Node<int>, Node<int>) RemoveMax(Node<int> lst)
        {
            Node<int> save_lst = lst;

            Node<int> maxNode = lst;
            Node<int> previousMaxNode = null;
            int max = lst.GetValue();
            previousMaxNode = null;


            while(lst.GetNext() != null)
            {

                if (lst.GetNext().GetValue() > max)
                {
                    maxNode = lst.GetNext();
                    max = lst.GetNext().GetValue();
                    previousMaxNode = lst;
                }

                lst = lst.GetNext();
            }

            if (previousMaxNode == null)
                lst = maxNode.GetNext();

            else
            {
                previousMaxNode.SetNext(maxNode.GetNext());
                maxNode.SetNext(null);
                lst = save_lst;
            }
```

```
            return (maxNode, lst);
        }

        public static Node<int> InsertToSortedList(Node<int> lst, Node<int> node)
        {
            Node<int> save_lst = lst;
            Node<int> previous = null;

            while (lst != null && lst.GetValue() < node.GetValue())
            {
                previous = lst;
                lst = lst.GetNext();
            }

            if (previous == null)
                lst = new Node<int>(node.GetValue(), lst);

            else
            {
                node.SetNext(lst);
                previous.SetNext(node);
                lst = save_lst;
            }

            return lst;

        }

        public static Node<int> Sort(Node<int> lst)
        {
            Node<int> sorted = new Node<int>(lst.GetValue());

            while(lst.GetNext() != null)
            {
                lst = lst.GetNext();
                Node<int> new_node = new Node<int>(lst.GetValue());
                sorted = InsertToSortedList(sorted, new_node);
            }

            return sorted;
        }

        public static Node<T> ReverseList<T>(Node<T> lst)
        {
            Node<T> curr = lst;
            Node<T> next = lst.GetNext();
            Node<T> next_next = lst.GetNext().GetNext();

```

```
            while(next != null)
            {
                next.SetNext(curr);
                curr = next;
                next = next_next;
                if (next_next != null)
                    next_next = next_next.GetNext();
            }

            lst.SetNext(null);

            return curr;
        }

        public static Node<T> ReverseListR<T>(Node<T> lst)
        {
            if (lst.GetNext() == null)
                return lst;

            Node<T> reversed = ReverseListR(lst.GetNext());
            lst.GetNext().SetNext(lst);
            lst.SetNext(null);

            return reversed;
        }

        public static int NegativeSequence(Node<int> lst)
        {
            int max_seq_len = 0;
            int curr_seq_len = 1;

            while(lst.GetNext() != null)
            {
                if (lst.GetValue() < 0 && lst.GetNext().GetValue() < 0)
                {
                    curr_seq_len++;
                }

                else if (curr_seq_len > 1)
                {
                    if (curr_seq_len > max_seq_len)
                        max_seq_len = curr_seq_len;
                    curr_seq_len = 1;
                }

                lst = lst.GetNext();

            }
```

```

            return max_seq_len;
        }

        public static Node<int> MergeSortedLists(Node<int> l1, Node<int> l2)
        {

            Node<int> merged = new Node<int>(0);
            Node<int> head = null;
            Node<int> last = null;

            while (l1 != null && l2 != null)
            {
                if (l1.GetValue() < l2.GetValue())
                {
                    merged.SetValue(l1.GetValue());
                    if (head == null)
                        head = merged;
                    l1 = l1.GetNext();
                }

                else
                {
                    merged.SetValue(l2.GetValue());
                    if (head == null)
                        head = merged;
                    l2 = l2.GetNext();
                }

                merged.SetNext(new Node<int>(0));
                last = merged;
                merged = merged.GetNext();

            }

            if (l1 != null)
                last.SetNext(l1);

            else if (l2 != null)
                last.SetNext(l2);

            return head;
        }

        public static Node<int> MergeSortedLists2(Node<int> l1, Node<int> l2)
        {
            Node<int> next_l1, next_l2 = null;
```

```
            Node<int> head = null;

            while(l1 != null && l2 != null)
            {
                if (l1.GetValue() > l2.GetValue())
                {
                    if (head == null)
                        head = l2;

                    next_l2 = l2.GetNext();
                    l2.SetNext(l1);
                    l2 = next_l2;
                }

                else
                {
                    if (head == null)
                        head = l1;

                    next_l1 = l1.GetNext();
                    l1.SetNext(l2);
                    l1 = next_l1;
                }
            }


            return head;
        }

        public static void AddFollowingNum(Node<int> lst, int val)
        {
            while(lst.GetValue() != val)
            {
                lst = lst.GetNext();
            }

            lst.SetNext(new Node<int>(val + 1, lst.GetNext()));
        }

        public static Node<int> ListsIntersection(Node<int> l1, Node<int> l2)
        {
            Node<int> head = null;
            Node<int> save_head = null;


            while(l1 != null)
            {
                if (IsExist(l2, l1.GetValue()))
```

```
                {
                    if (head == null)
                        head = new Node<int>(l1.GetValue());
                    else
                    {
                        head.SetNext(new Node<int>(l1.GetValue()));
                        head = head.GetNext();
                    }

                    if (save_head == null)
                        save_head = head;
                }

                l1 = l1.GetNext();
            }

            return save_head;
        }

        public static bool Compare(Node<int> left, Node<int> right)
        {
            bool equals = true;

            while(left.GetNext() != null && right.GetNext() != null)
            {
                equals = left.GetValue() == right.GetValue();

                left = left.GetNext();
                right = right.GetNext();
            }

            return equals;
        }

        public static Node<T> GetMiddle<T>(Node<T> lst)
        {
            Node<T> slow = lst;
            Node<T> fast = lst;

            while(fast.GetNext() != null)
            {
                slow = slow.GetNext();
                fast = fast.GetNext().GetNext();
            }

            return slow;
        }

        public static bool IsPalindrome(Node<char> lst)
```

```
        {
            Node<char> rlst = ReverseList(GetMiddle(lst));

            bool pali = true;
            while (pali && lst != rlst && lst != null)
            {
                pali = lst.GetValue() == rlst.GetValue();
                lst = lst.GetNext();
                rlst = rlst.GetNext();
            }

            return pali;

        }
    }
}

```