class Node

{

public int info;

public Node link;

public Node(int i)

{

info = i;

link = null;

}

}

}

class SingleLinkedList

{

private Node start;

public SingleLinkedList()

{

start = null;

}

public void DisplayList()

{

Node p;

if (start == null)

{

Console.WriteLine("List is Empty");

return;

}

Console.Write("List is : ");

p = start;

while (p != null)

{

Console.Write(p.info + " ");

p = p.link;

}

Console.WriteLine();

}

public void CountNodes()

{

int n = 0;

Node p = start;

while (p != null)

{

n++;

p = p.link;

}

Console.WriteLine("Number of nodes in the list = " + n);

}

public bool Search(int x)

{

int position = 1;

Node p = start;

while (p != null)

{

if (p.info == x)

break;

position++;

p = p.link;

}

if (p == null)

{

Console.WriteLine(x + "not found in list");

return false;

}

else

{

Console.WriteLine(x + "is at position " + position);

return true;

}

}

public void InsertInBeginning(int data)

{

Node temp = new Node(data);

temp.link = start;

start = temp;

}

public void InsertAtEnd(int data)

{

Node p;

Node temp = new Node(data);

if (start == null)

{

start = temp;

return;

}

p = start;

while (p.link != null)

p = p.link;

p.link = temp;

}

public void CreateList()

{

int i, n, data;

Console.Write("Enter the number of nodes : ");

n = Convert.ToInt32(Console.ReadLine());

if (n == 0)

return;

for (i = 1; i <= n; i++)

{

Console.Write("Enter the element to be inserted : ");

data = Convert.ToInt32(Console.ReadLine());

InsertAtEnd(data);

}

}

public void InsertAfter(int data, int x)

{

Node p = start;

while (p != null)

{

if (p.info == x)

break;

p = p.link;

}

if (p == null)

{

Console.WriteLine(x + "not present in the list : ");

}

else

{

Node temp = new Node(data);

temp.link = p.link;

p.link = temp;

}

}

public void InsertBefore(int data, int x)

{

Node temp;

//if list is empty//

if (start == null)

{

Console.WriteLine("List is empty");

return;

}

//x is in first node, new node is to be inserted before first node //

if (x == start.info)

{

temp = new Node(data);

temp.link = start;

start = temp;

return;

}

// find reference to predecessor on node conataining x

Node p = start;

while (p.link != null)

{

if (p.link.info == x)

break;

p = p.link;

}

if (p.link == null)

{

Console.WriteLine(x + "not present in the list");

}

else

{

temp = new Node(data);

temp.link = p.link;

p.link = temp;

}

}

public void InsertAtPosition(int data, int k)

{

Node temp;

int i;

if (k == 1)

{

temp = new Node(data);

temp.link = start;

start = temp;

return;

}

Node p = start;

for (i = 1; i < k - 1 && p != null; i++) //find a reference to k-1 node

p = p.link;

if (p == null)

Console.WriteLine("You can only insert upto" + i + "th position");

else

{

temp = new Node(data);

temp.link = p.link;

p.link = temp;

}

}

public void DeleteFirstNode()

{

if (start == null)

return;

start = start.link;

}

public void DeleteLastNode()

{

if (start == null)

return;

if (start.link == null)

{

start = null;

return;

}

Node p = start;

while (p.link.link != null)

p = p.link;

p.link = null;

}

public void DeleteNode(int x)

{

if (start == null)

{

Console.WriteLine("List is empty\n");

return;

}

//Deletion of first node

if (start.info == x)

{

start = start.link;

return;

}

// Deletion in between or at the end

Node p = start;

while (p.link != null)

{

if (p.link.info == x)

break;

p = p.link;

}

if (p.link == null)

Console.WriteLine("Element" + x + "not in the list");

else

p.link = p.link.link;

}

public void ReverseList()

{

Node prev, p, next;

prev = null;

p = start;

while (p != null)

{

next = p.link;

p.link = prev;

prev = p;

p = next;

}

start = prev;

}

public void BubbleSortExData()

{

Node end, p, q;

for (end = null; end != start.link; end = p)

{

for (p = start; p.link != end; p = p.link)

{

q = p.link;

if (p.info > q.info)

{

int temp = p.info;

p.info = q.info;

q.info = temp;

}

}

}

}

public void BubbleSortExLinks()

{

Node end, r, p, q, temp;

for (end = null; end != start.link; end = p)

{

for (r = p = start; p.link != end; r = p, p = p.link)

{

q = p.link;

if (p.info > q.info)

{

p.link = q.link;

q.link = p;

if (p != start)

r.link = q;

else

start = q;

temp = p;

p = q;

q = temp;

}

}

}

}

public SingleLinkedList Merge1(SingleLinkedList list)

{

SingleLinkedList mergeList = new SingleLinkedList();

mergeList.start = Merge1(start, list.start);

return mergeList;

}

private Node Merge1(Node p1, Node p2)

{

Node startM;

if (p1.info <= p2.info)

{

startM = new Node(p1.info);

p1 = p1.link;

}

else

{

startM = new Node(p2.info);

p2 = p2.link;

}

Node pM = startM;

while (p1 != null && p2 != null)

{

if (p1.info <= p2.info)

{

pM.link = new Node(p1.info);

p1 = p1.link;

}

else

{

pM.link = new Node(p2.info);

p2 = p2.link;

}

pM = pM.link;

}

//if second list has finished and elements left in first list

while (p1 != null)

{

pM.link = new Node(p1.info);

p1 = p1.link;

pM = pM.link;

}

//if first list has finished and elements left in second list

while (p2 != null)

{

pM.link = new Node(p2.info);

p2 = p2.link;

pM = pM.link;

}

return startM;

}

public SingleLinkedList Merge2(SingleLinkedList list)

{

SingleLinkedList mergeList = new SingleLinkedList();

mergeList.start = Merge2(start, list.start);

return mergeList;

}

private Node Merge2(Node p1, Node p2)

{

Node startM;

if (p1.info <= p2.info)

{

startM = p1;

p1 = p1.link;

}

else

{

startM = p2;

p2 = p2.link;

}

Node pM = startM;

while (p1 != null && p2 != null)

{

if (p1.info <= p2.info)

{

pM.link = p1;

pM = pM.link;

p1 = p1.link;

}

else

{

pM.link = p2;

pM = pM.link;

p2 = p2.link;

}

}

if (p1 == null)

pM.link = p2;

else

pM.link = p1;

return startM;

}

public void MergeSort()

{

start = MergeSortRec(start);

}

private Node MergeSortRec(Node listStart)

{

if (listStart == null || listStart.link == null) // if list is empty or has one element

return listStart;

//if list has more than one elements

Node start1 = listStart;

Node start2 = DivideList(listStart);

start1 = MergeSortRec(start1);

start2 = MergeSortRec(start2);

Node startM = Merge2(start1, start2);

return startM;

}

private Node DivideList(Node p)

{

Node q = p.link.link;

while (q != null && q.link != null)

{

p = p.link;

q = q.link.link;

}

Node start2 = p.link;

p.link = null;

return start2;

}

public bool HasCycle()

{

if (FindCycle() == null)

return false;

else

return true;

}

private Node FindCycle()

{

if (start == null || start.link == null)

return null;

Node slowR = start, fastR = start;

while (fastR != null && fastR.link != null)

{

slowR = slowR.link;

fastR = fastR.link.link;

if (slowR == fastR)

return slowR;

}

return null;

}

public void RemoveCycle()

{

Node c = FindCycle();

if (c == null)

return;

Console.WriteLine("Node at which cycle was detected is " + c.info);

Node p = c, q = c;

int lenCycle = 0;

do

{

lenCycle++;

q = q.link;

} while (p != q);

Console.WriteLine("Length of cycle is : " + lenCycle);

int lenRemList = 0;

p = start;

while (p != q)

{

lenRemList++;

p = p.link;

q = q.link;

}

Console.WriteLine("Number of nodes not included in cycle are " + lenRemList);

int lengthList = lenCycle + lenRemList;

Console.WriteLine("Length of the list is " + lengthList);

p = start;

for (int i = 1; i <= lengthList - 1; i++)

p = p.link;

p.link = null;

}

public void InsertCycle(int x)

{

if (start == null)

return;

Node p = start, px = null, prev = null;

while (p != null)

{

if (p.info == x)

px = p;

prev = p;

p = p.link;

}

if (px != null)

prev.link = px;

else

Console.WriteLine(x + "not present in the list");

}

}

}

class Program

{

static void Main(string[] args)

{

int choice, data, k, x;

SingleLinkedList list = new SingleLinkedList();

list.CreateList();

while (true)

{

Console.WriteLine("1.Display List");

Console.WriteLine("2.Count the number of nodes");

Console.WriteLine("3.search for an element");

Console.WriteLine("4.Insert in empty list/ Insert in beginning of list");

Console.WriteLine("5.Insert a node at the end of list");

Console.WriteLine("6.Insert a node after a specified node");

Console.WriteLine("7.Insert a node befor a specified node");

Console.WriteLine("8.Insert a node at a given position");

Console.WriteLine("9.Delete first node");

Console.WriteLine("10.Delete last node");

Console.WriteLine("11.Delete any node");

Console.WriteLine("12.Reverse the list");

Console.WriteLine("13.Bubble sort by exchanging data");

Console.WriteLine("14.Bubble sort by exchanging links");

Console.WriteLine("15.Merge Sort");

Console.WriteLine("16.Insert Cycle");

Console.WriteLine("17.Detect Cycle");

Console.WriteLine("18.Remove Cycle");

Console.WriteLine("19.Quit");

Console.Write("Enter your choice : ");

choice = Convert.ToInt32(Console.ReadLine());

if (choice == 19)

break;

switch (choice)

{

case 1:

list.DisplayList();

break;

case 2:

list.CountNodes();

break;

case 3:

Console.WriteLine("Enter the element to be searched : ");

data = Convert.ToInt32(Console.ReadLine());

list.Search(data);

break;

case 4:

Console.Write("Enter the element to be inserted : ");

data = Convert.ToInt32(Console.ReadLine());

list.InsertInBeginning(data);

break;

case 5:

Console.Write("Enter the element to be inserted : ");

data = Convert.ToInt32(Console.ReadLine());

list.InsertAtEnd(data);

break;

case 6:

Console.Write("Enter the element to be inserted : ");

data = Convert.ToInt32(Console.ReadLine());

Console.Write("Enter the element after which to insert : ");

x = Convert.ToInt32(Console.ReadLine());

list.InsertAfter(data, x);

break;

case 7:

Console.Write("Enter the element to be inserted : ");

data = Convert.ToInt32(Console.ReadLine());

Console.Write("Enter the element before which to insert : ");

x = Convert.ToInt32(Console.ReadLine());

list.InsertBefore(data, x);

break;

case 8:

Console.Write("Enter the element to be inserted : ");

data = Convert.ToInt32(Console.ReadLine());

Console.Write("Enter the position at which to insert : ");

k = Convert.ToInt32(Console.ReadLine());

list.InsertBefore(data, k);

break;

case 9:

list.DeleteFirstNode();

break;

case 10:

list.DeleteLastNode();

break;

case 11:

Console.Write("Enter the element to be deleted : ");

data = Convert.ToInt32(Console.ReadLine());

list.DeleteNode(data);

break;

case 12:

list.ReverseList();

break;

case 13:

list.BubbleSortExData();

break;

case 14:

list.BubbleSortExLinks();

break;

case 15:

list.MergeSort();

break;

case 16:

Console.Write("Enter the element at which the cycle has to be inserted : ");

data = Convert.ToInt32(Console.ReadLine());

list.InsertCycle(data);

break;

case 17: //Detect cycle

if (list.HasCycle())

Console.WriteLine("List has a cycle");

else

Console.WriteLine("List has no cycle");

break;

case 18: //RemoveCycle

list.RemoveCycle();

break;

default:

Console.WriteLine("Wrong choice");

break;

}

}

}

}

}