**Task No. 1:** Write a program to create a linked list and Perform.

1) Traversing

2) Insertion

3) Deletion

**Solution:**

using System;

namespace dOUBLY\_LINKLIST

{

class node

{

public int data; public node next; public node previous;

public node(int a)

{

data = a; next = null; previous = null; } }

class DLL

{ public node head;

public void DlistPrint()

{

node temp = head;

node last = null;

Console.WriteLine("Traversal in FORWARD Direction");

while (temp != null)

{

Console.WriteLine($" {temp.data}");

last = temp;

temp = temp.next;

}

Console.WriteLine("");

Console.WriteLine("Traversal in REVERSE direction");

while (last != null)

{

Console.WriteLine($" {last.data}");

last = last.previous;

}

Console.WriteLine();

}

// INSERT NODE CODDING

public void InsertAtFront(int new\_data)

{

node temp = head;

node new\_node = new node(new\_data);

new\_node.previous = null;

new\_node.next = head;

if (head != null)

{

head.previous = new\_node;

}

head = new\_node;

}

public void InsertAfterANode(node current\_node, int new\_data)

{

if (current\_node == null)

{

Console.WriteLine("The given Current node cannot be NULL ");

return;

}

node new\_node = new node(new\_data);

new\_node.next = current\_node.next;

new\_node.previous = current\_node;

current\_node.next = new\_node;

if (new\_node.next != null)

{

new\_node.next.previous = new\_node;

}

}

public void InsertBeforeANode(node current\_node, int new\_data)

{

if (current\_node == null)

{

Console.WriteLine("The given Current node cannot be NULL ");

return;

}

node new\_node = new node(new\_data);

new\_node.previous = current\_node.previous;

new\_node.next = current\_node;

if (current\_node.previous != null)

{

current\_node.previous.next = new\_node;

}

if (new\_node.previous != null)

{

current\_node.previous = new\_node;

}

else

{

Console.WriteLine("Can't insert Because Previous Node Is Null");

}

//node new\_node = new node(new\_data);

///\* 4. Make next of new node as next of prev\_node \*/

//new\_node.next = current\_node.next;

///\* 5. Make the next of prev\_node as new\_node \*/

//current\_node.next = new\_node;

///\* 6. Make prev\_node as previous of new\_node \*/

//new\_node.previous = current\_node;

///\* 7. Change previous of new\_node's next node \*/

//if (new\_node.next != null)

//{

// new\_node.next.previous = new\_node;

//}

}

public void InsertAtEnd(int new\_data)

{

node new\_node = new node(new\_data);

node last = head;

new\_node.next = null;

if (head == null)

{

new\_node.previous = null;

head = new\_node;

return;

}

while (last.next != null)

{

last = last.next;

}

last.next = new\_node;

new\_node.previous = last;

}

// DELETE NODE CODDING

public void DeleteFirstNode()

{

node temp = head;

if (temp == null)

{

Console.WriteLine("The given Current node cannot be NULL ");

return;

}

head = head.next;

if (head != null)

{

head.previous = null;

}

}

public void DeleteLastNode()

{

node last = head;

if (last == null)

{

Console.WriteLine("The given Current node cannot be NULL ");

return;

}

while (last.next != null)

{

last = last.next;

}

if (last.previous != null)

{

last.previous.next = null;

}

}

public void DeleteMiddleNode(node delete\_node)

{

if (delete\_node == null)

{

Console.WriteLine("The given Current node cannot be NULL ");

return;

}

if (delete\_node.next != null)

{

delete\_node.next.previous = delete\_node.previous;

}

delete\_node.previous.next = delete\_node.next;

delete\_node = null;

}

public void DelteAtValue(int value)

{

node temp = head;

while (temp.data != value && temp.next != null)

{

temp = temp.next;

}

if (temp.next != null)

{

temp.next.previous = temp.previous;

}

else

{

Console.WriteLine("Can't Delete Value because value Is Not Present");

return;

}

if (temp.previous != null)

{

temp.previous.next = temp.next;

}

head = temp.next;

}

}

class Program

{

static void Main(string[] args)

{

DLL dlist = new DLL();

dlist.head = new node(1);

node second = new node(2);

node third = new node(3);

dlist.head.next = second;

second.next = third;

third.next = null;

third.previous = second;

second.previous = dlist.head;

dlist.head.previous = null;

byte res;

do

{

Console.WriteLine("Please Choose An Option Below ");

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

Console.WriteLine(" 2) ADD Element");

Console.WriteLine(" 3) DELETE Element");

Console.Write("Enter : ");

res = byte.Parse(Console.ReadLine());

switch (res)

{

case 1:

dlist.DlistPrint();

break;

case 2:

byte res1;

Console.WriteLine("Please Choose An Option Below ");

Console.WriteLine(" 1) ADD STARTING Of The Node");

Console.WriteLine(" 2) ADD AFTER a Given Node");

Console.WriteLine(" 3) ADD BEFORE a Given Node");

Console.WriteLine(" 4) ADD END of the Node");

Console.Write("Enter : ");

res1 = byte.Parse(Console.ReadLine());

if (res1 == 1)

{

Console.Write("Enter Element You Want To Insert In Front : ");

int data = int.Parse(Console.ReadLine());

dlist.InsertAtFront(data);

}

else if (res1 == 2)

{

Console.Write("Enter Element You Want To Insert : ");

int data = int.Parse(Console.ReadLine());

dlist.InsertAfterANode(second, data);

}

else if (res1 == 3)

{

Console.Write("Enter Element You Want To Insert : ");

int data = int.Parse(Console.ReadLine());

dlist.InsertBeforeANode(second, data);

}

else if (res1 == 4)

{

Console.Write("Enter Element You Want To Insert : ");

int data = int.Parse(Console.ReadLine());

dlist.InsertAtEnd(data);

}

break;

case 3:

Console.WriteLine("Please Choose An Option Below ");

Console.WriteLine(" 1) Delete At First");

Console.WriteLine(" 2) Delete At Middle Node");

Console.WriteLine(" 3) Delete At Last");

Console.WriteLine(" 4) Delete The Value Node");

Console.Write("Enter : ");

byte res2 = byte.Parse(Console.ReadLine());

if (res2 == 1)

{

dlist.DeleteFirstNode();

Console.WriteLine("Node Is SuccessFully Deleted");

}

else if (res2 == 2)

{

dlist.DeleteMiddleNode(second);

Console.WriteLine("Node Is SuccessFully Deleted");

}

else if (res2 == 3)

{

dlist.DeleteLastNode();

Console.WriteLine("Node Is SuccessFully Deleted");

}

else if (res2 == 4)

{

Console.Write("Enter Value You Want To Delete : ");

dlist.DelteAtValue(int.Parse(Console.ReadLine()));

//Console.WriteLine("Node Is SuccessFully Deleted");

}

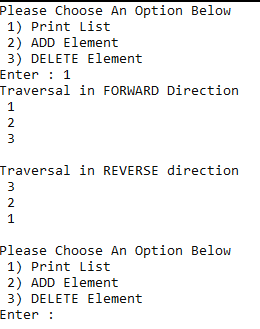
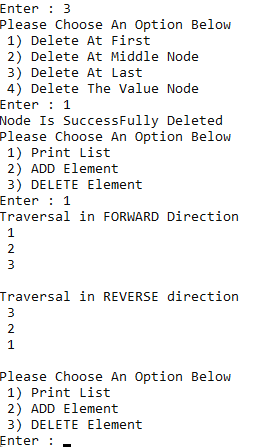
break;

default:

break;

}

} while (res != 1 || res != 2 || res != 3);}}}

**Output:**

**Graphical user interface, text

Description automatically generated**

**Text

Description automatically generated**