**Linked List**

Like arrays, Linked List is a linear data structure. Unlike arrays, linked list elements are not stored at a contiguous location; the elements are linked using pointers.

**What is Linked List?**

When we want to work with an unknown number of data values, we use a linked list data structure to organize that data. The linked list is a linear data structure that contains a sequence of elements such that each element links to its next element in the sequence. Each element in a linked list is called "Node".



**Why Linked List?**   
Arrays can be used to store linear data of similar types, but arrays have the following limitations.

**1)** The size of the arrays is fixed: So we must know the upper limit on the number of elements in advance. Also, generally, the allocated memory is equal to the upper limit irrespective of the usage.   
**2)** Inserting a new element in an array of elements is expensive because the room has to be created for the new elements and to create room existing elements have to be shifted. 

For example, in a system, if we maintain a sorted list of IDs in an array id[].   
id[] = [1000, 1010, 1050, 2000, 2040].   
And if we want to insert a new ID 1005, then to maintain the sorted order, we have to move all the elements after 1000 (excluding 1000).   
Deletion is also expensive with arrays until unless some special techniques are used. For example, to delete 1010 in id[], everything after 1010 has to be moved.

**Advantages over arrays**   
**1)** Dynamic size   
**2)** Ease of insertion/deletion

**Drawbacks:**   
**1)** Random access is not allowed. We have to access elements sequentially starting from the first node. So we cannot do binary search with linked lists efficiently with its default implementation.   
**2)** Extra memory space for a pointer is required with each element of the list.   
**3)** Not cache friendly. Since array elements are contiguous locations, there is locality of reference which is not there in case of linked lists.

**Representation:**   
A linked list is represented by a pointer to the first node of the linked list. The first node is called the head. If the linked list is empty, then the value of the head is NULL.   
Each node in a list consists of at least two parts:   
1) data   
2) Pointer (Or Reference) to the next node   
In C, we can represent a node using structures. Below is an example of a linked list node with integer data.

**// A linked list node**

**struct Node {**

**int data;**

**struct Node\* next;**

**};**

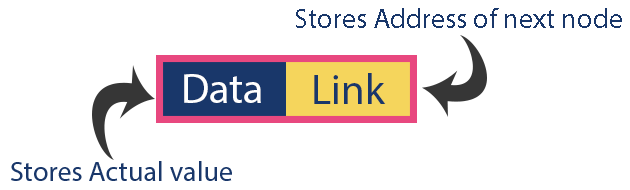
**Single Linked List**

Simply a list is a sequence of data, and the linked list is a sequence of data linked with each other.   
The formal definition of a single linked list is as follows...

**Definition**

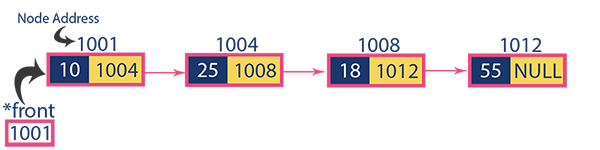
Single linked list is a sequence of elements in which every element has link to its next element in the sequence.

In any single linked list, the individual element is called as "Node". Every "Node" contains two fields, data field, and the next field. The data field is used to store actual value of the node and next field is used to store the address of next node in the sequence.  
The graphical representation of a node in a single linked list is as follows...



**Important Points to be Remembered**  
  In a single linked list, the address of the first node is always stored in a reference node known as "front" (Sometimes it is also known as "head").  
  Always next part (reference part) of the last node must be NULL.

Example



# Operations on Single Linked List

The following operations are performed on a Single Linked List

* **Insertion**
* **Deletion**
* **Display**

Before we implement actual operations, first we need to set up an empty list. First, perform the following steps before implementing actual operations.

* Step 1 - Include all the **header files** which are used in the program.
* Step 2 - Declare all the **user defined functions**.
* Step 3 - Define a **Node** structure with two members **data** and **next**
* Step 4 - Define a Node pointer **'head'** and set it to **NULL**.
* Step 5 - Implement the main method by displaying operations menu and make suitable function calls in the main method to perform user selected operation.

# Insertion

In a single linked list, the insertion operation can be performed in three ways. They are as follows...

1. Inserting At Beginning of the list
2. Inserting At End of the list
3. Inserting At Specific location in the list

# Inserting At Beginning of the list

We can use the following steps to insert a new node at beginning of the single linked list...

* Step 1 - Create a **newNode** with given value.
* Step 2 - Check whether list is **Empty** (**head** == **NULL**)
* Step 3 - If it is **Empty** then, set **newNode→next** = **NULL** and **head** = **newNode**.
* Step 4 - If it is **Not Empty** then, set **newNode→next** = **head** and **head** = **newNode**.

# Inserting At End of the list

We can use the following steps to insert a new node at end of the single linked list...

* Step 1 - Create a **newNode** with given value and **newNode → next** as **NULL**.
* Step 2 - Check whether list is **Empty** (**head** == **NULL**).
* Step 3 - If it is **Empty** then, set **head** = **newNode**.
* Step 4 - If it is **Not Empty** then, define a node pointer **temp** and initialize with **head**.
* Step 5 - Keep moving the **temp** to its next node until it reaches to the last node in the list (until **temp → next** is equal to **NULL**).
* Step 6 - Set **temp → next** = **newNode**.

# Inserting At Specific location in the list (After a Node)

We can use the following steps to insert a new node after a node in the single linked list...

* Step 1 - Create a **newNode** with given value.
* Step 2 - Check whether list is **Empty** (**head** == **NULL**)
* Step 3 - If it is **Empty** then, set **newNode → next** = **NULL** and **head** = **newNode**.
* Step 4 - If it is **Not Empty** then, define a node pointer **temp** and initialize with **head**.
* Step 5 - Keep moving the **temp** to its next node until it reaches to the node after which we want to insert the newNode (until **temp1 → data** is equal to **location**, here location is the node value after which we want to insert the newNode).
* Step 6 - Every time check whether **temp** is reached to last node or not. If it is reached to last node then display **'Given node is not found in the list!!! Insertion not possible!!!'** and terminate the function. Otherwise move the **temp** to next node.
* Step 7 - Finally, Set '**newNode → next** = **temp → next**' and '**temp → next** = **newNode**'

# Deletion

In a single linked list, the deletion operation can be performed in three ways. They are as follows...

1. Deleting from Beginning of the list
2. Deleting from End of the list
3. Deleting a Specific Node

# Deleting from Beginning of the list

We can use the following steps to delete a node from beginning of the single linked list...

* Step 1 - Check whether list is **Empty** (**head** == **NULL**)
* Step 2 - If it is **Empty** then, display **'List is Empty!!! Deletion is not possible'** and terminate the function.
* Step 3 - If it is **Not Empty** then, define a Node pointer **'temp'** and initialize with **head**.
* Step 4 - Check whether list is having only one node (**temp → next** == **NULL**)
* Step 5 - If it is **TRUE** then set **head** = **NULL** and delete **temp** (Setting **Empty** list conditions)
* Step 6 - If it is **FALSE** then set **head** = **temp → next**, and delete **temp**.

# Deleting from End of the list

We can use the following steps to delete a node from end of the single linked list...

* Step 1 - Check whether list is **Empty** (**head** == **NULL**)
* Step 2 - If it is **Empty** then, display **'List is Empty!!! Deletion is not possible'** and terminate the function.
* Step 3 - If it is **Not Empty** then, define two Node pointers **'temp1'** and '**temp2'** and initialize '**temp1**' with **head**.
* Step 4 - Check whether list has only one Node (**temp1 → next** == **NULL**)
* Step 5 - If it is **TRUE**. Then, set **head** = **NULL** and delete **temp1**. And terminate the function. (Setting **Empty** list condition)
* Step 6 - If it is **FALSE**. Then, set '**temp2 = temp1** ' and move **temp1** to its next node. Repeat the same until it reaches to the last node in the list. (until **temp1 → next** == **NULL**)
* Step 7 - Finally, Set **temp2 → next** = **NULL** and delete **temp1**.

# Deleting a Specific Node from the list

We can use the following steps to delete a specific node from the single linked list...

* Step 1 - Check whether list is **Empty** (**head** == **NULL**)
* Step 2 - If it is **Empty** then, display **'List is Empty!!! Deletion is not possible'** and terminate the function.
* Step 3 - If it is **Not Empty** then, define two Node pointers **'temp1'** and '**temp2**' and initialize '**temp1**' with **head**.
* Step 4 - Keep moving the **temp1** until it reaches to the exact node to be deleted or to the last node. And every time set '**temp2 = temp1**' before moving the '**temp1**' to its next node.
* Step 5 - If it is reached to the last node then display **'Given node not found in the list! Deletion not possible!!!'**. And terminate the function.
* Step 6 - If it is reached to the exact node which we want to delete, then check whether list is having only one node or not
* Step 7 - If list has only one node and that is the node to be deleted, then set **head** = **NULL** and delete **temp1** (**free(temp1)**).
* Step 8 - If list contains multiple nodes, then check whether **temp1** is the first node in the list (**temp1 == head**).
* Step 9 - If **temp1** is the first node then move the **head** to the next node (**head = head → next**) and delete **temp1**.
* Step 10 - If **temp1** is not first node then check whether it is last node in the list (**temp1 → next == NULL**).
* Step 11 - If **temp1** is last node then set **temp2 → next** = **NULL** and delete **temp1** (**free(temp1)**).
* Step 12 - If **temp1** is not first node and not last node then set **temp2 → next** = **temp1 → next** and delete **temp1** (**free(temp1)**).

# Displaying a Single Linked List

We can use the following steps to display the elements of a single linked list...

* Step 1 - Check whether list is **Empty** (**head** == **NULL**)
* Step 2 - If it is **Empty** then, display **'List is Empty!!!'** and terminate the function.
* Step 3 - If it is **Not Empty** then, define a Node pointer **'temp'** and initialize with **head**.
* Step 4 - Keep displaying **temp → data** with an arrow (**--->**) until **temp** reaches to the last node
* Step 5 - Finally display **temp → data** with arrow pointing to **NULL** (**temp → data ---> NULL**).

**Implementation of Single Linked List using C Programming**

**#include<stdio.h>**

**#include<conio.h>**

**#include<stdlib.h>**

**void insertAtBeginning(int);**

**void insertAtEnd(int);**

**void insertBetween(int,int,int);**

**void display();**

**void removeBeginning();**

**void removeEnd();**

**void removeSpecific(int);**

**struct Node**

**{**

**int data;**

**struct Node \*next;**

**}\*head = NULL;**

**void main()**

**{**

**int choice,value,choice1,loc1,loc2;**

**clrscr();**

**while(1){**

**mainMenu: printf("\n\n\*\*\*\*\*\* MENU \*\*\*\*\*\*\n1. Insert\n2. Display\n3. Delete\n4. Exit\nEnter your choice: ");**

**scanf("%d",&choice);**

**switch(choice)**

**{**

**case 1: printf("Enter the value to be insert: ");**

**scanf("%d",&value);**

**while(1){**

**printf("Where you want to insert: \n1. At Beginning\n2. At End\n3. Between\nEnter your choice: ");**

**scanf("%d",&choice1);**

**switch(choice1)**

**{**

**case 1: insertAtBeginning(value);**

**break;**

**case 2: insertAtEnd(value);**

**break;**

**case 3: printf("Enter the two values where you wanto insert: ");**

**scanf("%d%d",&loc1,&loc2);**

**insertBetween(value,loc1,loc2);**

**break;**

**default: printf("\nWrong Input!! Try again!!!\n\n");**

**goto mainMenu;**

**}**

**goto subMenuEnd;**

**}**

**subMenuEnd:**

**break;**

**case 2: display();**

**break;**

**case 3: printf("How do you want to Delete: \n1. From Beginning\n2. From End\n3. Spesific\nEnter your choice: ");**

**scanf("%d",&choice1);**

**switch(choice1)**

**{**

**case 1: removeBeginning();**

**break;**

**case 2: removeEnd();**

**break;**

**case 3: printf("Enter the value which you wanto delete: ");**

**scanf("%d",&loc2);**

**removeSpecific(loc2);**

**break;**

**default: printf("\nWrong Input!! Try again!!!\n\n");**

**goto mainMenu;**

**}**

**break;**

**case 4: exit(0);**

**default: printf("\nWrong input!!! Try again!!\n\n");**

**}**

**}**

**}**

**void insertAtBeginning(int value)**

**{**

**struct Node \*newNode;**

**newNode = (struct Node\*)malloc(sizeof(struct Node));**

**newNode->data = value;**

**if(head == NULL)**

**{**

**newNode->next = NULL;**

**head = newNode;**

**}**

**else**

**{**

**newNode->next = head;**

**head = newNode;**

**}**

**printf("\nOne node inserted!!!\n");**

**}**

**void insertAtEnd(int value)**

**{**

**struct Node \*newNode;**

**newNode = (struct Node\*)malloc(sizeof(struct Node));**

**newNode->data = value;**

**newNode->next = NULL;**

**if(head == NULL)**

**head = newNode;**

**else**

**{**

**struct Node \*temp = head;**

**while(temp->next != NULL)**

**temp = temp->next;**

**temp->next = newNode;**

**}**

**printf("\nOne node inserted!!!\n");**

**}**

**void insertBetween(int value, int loc1, int loc2)**

**{**

**struct Node \*newNode;**

**newNode = (struct Node\*)malloc(sizeof(struct Node));**

**newNode->data = value;**

**if(head == NULL)**

**{**

**newNode->next = NULL;**

**head = newNode;**

**}**

**else**

**{**

**struct Node \*temp = head;**

**while(temp->data != loc1 && temp->data != loc2)**

**temp = temp->next;**

**newNode->next = temp->next;**

**temp->next = newNode;**

**}**

**printf("\nOne node inserted!!!\n");**

**}**

**void removeBeginning()**

**{**

**if(head == NULL)**

**printf("\n\nList is Empty!!!");**

**else**

**{**

**struct Node \*temp = head;**

**if(head->next == NULL)**

**{**

**head = NULL;**

**free(temp);**

**}**

**else**

**{**

**head = temp->next;**

**free(temp);**

**printf("\nOne node deleted!!!\n\n");**

**}**

**}**

**}**

**void removeEnd()**

**{**

**if(head == NULL)**

**{**

**printf("\nList is Empty!!!\n");**

**}**

**else**

**{**

**struct Node \*temp1 = head,\*temp2;**

**if(head->next == NULL)**

**head = NULL;**

**else**

**{**

**while(temp1->next != NULL)**

**{**

**temp2 = temp1;**

**temp1 = temp1->next;**

**}**

**temp2->next = NULL;**

**}**

**free(temp1);**

**printf("\nOne node deleted!!!\n\n");**

**}**

**}**

**void removeSpecific(int delValue)**

**{**

**struct Node \*temp1 = head, \*temp2;**

**while(temp1->data != delValue)**

**{**

**if(temp1 -> next == NULL){**

**printf("\nGiven node not found in the list!!!");**

**goto functionEnd;**

**}**

**temp2 = temp1;**

**temp1 = temp1 -> next;**

**}**

**temp2 -> next = temp1 -> next;**

**free(temp1);**

**printf("\nOne node deleted!!!\n\n");**

**functionEnd:**

**}**

**void display()**

**{**

**if(head == NULL)**

**{**

**printf("\nList is Empty\n");**

**}**

**else**

**{**

**struct Node \*temp = head;**

**printf("\n\nList elements are - \n");**

**while(temp->next != NULL)**

**{**

**printf("%d --->",temp->data);**

**temp = temp->next;**

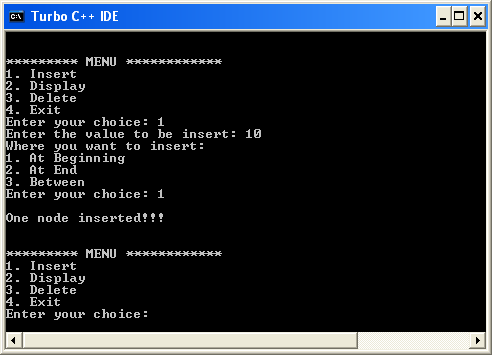
**}**

**printf("%d --->NULL",temp->data);**

**}**

**}**

**Output**



**Double Linked List**

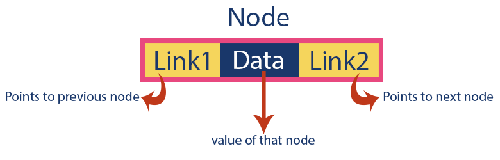
# What is Double Linked List?

In a single linked list, every node has a link to its next node in the sequence. So, we can traverse from one node to another node only in one direction and we can not traverse back. We can solve this kind of problem by using a double linked list. A double linked list can be defined as follows...

**Definition**

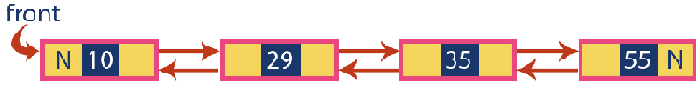
Double linked list is a sequence of elements in which every element has links to its previous element and next element in the sequence.

In a double linked list, every node has a link to its previous node and next node. So, we can traverse forward by using the next field and can traverse backward by using the previous field. Every node in a double linked list contains three fields and they are shown in the following figure...



Here, **'link1'** field is used to store the address of the previous node in the sequence, **'link2'** field is used to store the address of the next node in the sequence and **'data'** field is used to store the actual value of that node.

Example



**Important Points to be Remembered**  
  In double linked list, the first node must be always pointed by head.  
  Always the previous field of the first node must be NULL.  
  Always the next field of the last node must be NULL.

# Operations on Double Linked List

In a double linked list, we perform the following operations...

1. Insertion
2. Deletion
3. Display

# Insertion

In a double linked list, the insertion operation can be performed in three ways as follows...

1. Inserting At Beginning of the list
2. Inserting At End of the list
3. Inserting At Specific location in the list

# Inserting At Beginning of the list

We can use the following steps to insert a new node at beginning of the double linked list...

* Step 1 - Create a **newNode** with given value and **newNode → previous** as **NULL**.
* Step 2 - Check whether list is **Empty** (**head** == **NULL**)
* Step 3 - If it is **Empty** then, assign **NULL** to **newNode → next** and **newNode** to **head**.
* Step 4 - If it is **not Empty** then, assign **head** to **newNode → next** and **newNode** to **head**.

# Inserting At End of the list

We can use the following steps to insert a new node at end of the double linked list...

* Step 1 - Create a **newNode** with given value and **newNode → next** as **NULL**.
* Step 2 - Check whether list is **Empty** (**head** == **NULL**)
* Step 3 - If it is **Empty**, then assign **NULL** to **newNode → previous** and **newNode** to **head**.
* Step 4 - If it is **not Empty**, then, define a node pointer **temp** and initialize with **head**.
* Step 5 - Keep moving the **temp** to its next node until it reaches to the last node in the list (until **temp → next** is equal to **NULL**).
* Step 6 - Assign **newNode** to **temp → next** and **temp** to **newNode → previous**.

# Inserting At Specific location in the list (After a Node)

We can use the following steps to insert a new node after a node in the double linked list...

* Step 1 - Create a **newNode** with given value.
* Step 2 - Check whether list is **Empty** (**head** == **NULL**)
* Step 3 - If it is **Empty** then, assign **NULL** to both **newNode → previous** & **newNode → next** and set **newNode** to **head**.
* Step 4 - If it is **not Empty** then, define two node pointers **temp1** & **temp2** and initialize **temp1** with **head**.
* Step 5 - Keep moving the **temp1** to its next node until it reaches to the node after which we want to insert the newNode (until **temp1 → data** is equal to **location**, here location is the node value after which we want to insert the newNode).
* Step 6 - Every time check whether **temp1** is reached to the last node. If it is reached to the last node then display **'Given node is not found in the list!!! Insertion not possible!!!'** and terminate the function. Otherwise move the **temp1** to next node.
* Step 7 - Assign **temp1 → next** to **temp2**, **newNode** to **temp1 → next**, **temp1** to **newNode → previous**, **temp2** to **newNode → next** and **newNode** to **temp2 → previous**.

# Deletion

In a double linked list, the deletion operation can be performed in three ways as follows...

1. Deleting from Beginning of the list
2. Deleting from End of the list
3. Deleting a Specific Node

# Deleting from Beginning of the list

We can use the following steps to delete a node from beginning of the double linked list...

* Step 1 - Check whether list is **Empty** (**head** == **NULL**)
* Step 2 - If it is **Empty** then, display **'List is Empty!!! Deletion is not possible'** and terminate the function.
* Step 3 - If it is not Empty then, define a Node pointer **'temp'** and initialize with **head**.
* Step 4 - Check whether list is having only one node (**temp → previous** is equal to **temp → next**)
* Step 5 - If it is **TRUE**, then set **head** to **NULL** and delete **temp** (Setting **Empty** list conditions)
* Step 6 - If it is **FALSE**, then assign **temp → next** to **head**, **NULL** to **head → previous** and delete **temp**.

# Deleting from End of the list

We can use the following steps to delete a node from end of the double linked list...

* Step 1 - Check whether list is **Empty** (**head** == **NULL**)
* Step 2 - If it is **Empty**, then display **'List is Empty!!! Deletion is not possible'** and terminate the function.
* Step 3 - If it is not Empty then, define a Node pointer **'temp'** and initialize with **head**.
* Step 4 - Check whether list has only one Node (**temp → previous** and **temp → next** both are **NULL**)
* Step 5 - If it is **TRUE**, then assign **NULL** to **head** and delete **temp**. And terminate from the function. (Setting **Empty** list condition)
* Step 6 - If it is **FALSE**, then keep moving **temp** until it reaches to the last node in the list. (until **temp → next** is equal to **NULL**)
* Step 7 - Assign **NULL** to **temp → previous → next** and delete **temp**.

# Deleting a Specific Node from the list

We can use the following steps to delete a specific node from the double linked list...

* Step 1 - Check whether list is **Empty** (**head** == **NULL**)
* Step 2 - If it is **Empty** then, display **'List is Empty!!! Deletion is not possible'** and terminate the function.
* Step 3 - If it is not Empty, then define a Node pointer **'temp'** and initialize with **head**.
* Step 4 - Keep moving the **temp** until it reaches to the exact node to be deleted or to the last node.
* Step 5 - If it is reached to the last node, then display **'Given node not found in the list! Deletion not possible!!!'** and terminate the fuction.
* Step 6 - If it is reached to the exact node which we want to delete, then check whether list is having only one node or not
* Step 7 - If list has only one node and that is the node which is to be deleted then set **head** to **NULL** and delete **temp** (**free(temp)**).
* Step 8 - If list contains multiple nodes, then check whether **temp** is the first node in the list (**temp == head**).
* Step 9 - If **temp** is the first node, then move the **head** to the next node (**head = head → next**), set **head** of **previous** to **NULL** (**head → previous = NULL**) and delete **temp**.
* Step 10 - If **temp** is not the first node, then check whether it is the last node in the list (**temp → next == NULL**).
* Step 11 - If **temp** is the last node then set **temp** of **previous** of **next** to **NULL** (**temp → previous → next = NULL**) and delete **temp** (**free(temp**)).
* Step 12 - If **temp** is not the first node and not the last node, then set **temp** of **previous** of **next** to **temp** of **next** (**temp → previous → next = temp → next**), **temp** of **next** of **previous** to **temp** of **previous** (**temp → next → previous = temp → previous**) and delete **temp** (**free(temp)**).

# Displaying a Double Linked List

We can use the following steps to display the elements of a double linked list...

* Step 1 - Check whether list is **Empty** (**head** == **NULL**)
* Step 2 - If it is **Empty**, then display **'List is Empty!!!'** and terminate the function.
* Step 3 - If it is not Empty, then define a Node pointer **'temp'** and initialize with **head**.
* Step 4 - Display **'NULL <--- '**.
* Step 5 - Keep displaying **temp → data** with an arrow (**<===>**) until **temp** reaches to the last node
* Step 6 - Finally, display **temp → data** with arrow pointing to **NULL** (**temp → data ---> NULL**).

**Implementation of Double Linked List using C Programming**

**#include<stdio.h>**

**#include<conio.h>**

**void insertAtBeginning(int);**

**void insertAtEnd(int);**

**void insertAtAfter(int,int);**

**void deleteBeginning();**

**void deleteEnd();**

**void deleteSpecific(int);**

**void display();**

**struct Node**

**{**

**int data;**

**struct Node \*previous, \*next;**

**}\*head = NULL;**

**void main()**

**{**

**int choice1, choice2, value, location;**

**clrscr();**

**while(1)**

**{**

**printf("\n\*\*\*\*\*\*\*\*\*\*\* MENU \*\*\*\*\*\*\*\*\*\*\*\*\*\n");**

**printf("1. Insert\n2. Delete\n3. Display\n4. Exit\nEnter your choice: ");**

**scanf("%d",&choice1);**

**switch()**

**{**

**case 1: printf("Enter the value to be inserted: ");**

**scanf("%d",&value);**

**while(1)**

**{**

**printf("\nSelect from the following Inserting options\n");**

**printf("1. At Beginning\n2. At End\n3. After a Node\n4. Cancel\nEnter your choice: ");**

**scanf("%d",&choice2);**

**switch(choice2)**

**{**

**case 1: insertAtBeginning(value);**

**break;**

**case 2: insertAtEnd(value);**

**break;**

**case 3: printf("Enter the location after which you want to insert: ");**

**scanf("%d",&location);**

**insertAfter(value,location);**

**break;**

**case 4: goto EndSwitch;**

**default: printf("\nPlease select correct Inserting option!!!\n");**

**}**

**}**

**case 2: while(1)**

**{**

**printf("\nSelect from the following Deleting options\n");**

**printf("1. At Beginning\n2. At End\n3. Specific Node\n4. Cancel\nEnter your choice: ");**

**scanf("%d",&choice2);**

**switch(choice2)**

**{**

**case 1: deleteBeginning();**

**break;**

**case 2: deleteEnd();**

**break;**

**case 3: printf("Enter the Node value to be deleted: ");**

**scanf("%d",&location);**

**deleteSpecic(location);**

**break;**

**case 4: goto EndSwitch;**

**default: printf("\nPlease select correct Deleting option!!!\n");**

**}**

**}**

**EndSwitch: break;**

**case 3: display();**

**break;**

**case 4: exit(0);**

**default: printf("\nPlease select correct option!!!");**

**}**

**}**

**}**

**void insertAtBeginning(int value)**

**{**

**struct Node \*newNode;**

**newNode = (struct Node\*)malloc(sizeof(struct Node));**

**newNode -> data = value;**

**newNode -> previous = NULL;**

**if(head == NULL)**

**{**

**newNode -> next = NULL;**

**head = newNode;**

**}**

**else**

**{**

**newNode -> next = head;**

**head = newNode;**

**}**

**printf("\nInsertion success!!!");**

**}**

**void insertAtEnd(int value)**

**{**

**struct Node \*newNode;**

**newNode = (struct Node\*)malloc(sizeof(struct Node));**

**newNode -> data = value;**

**newNode -> next = NULL;**

**if(head == NULL)**

**{**

**newNode -> previous = NULL;**

**head = newNode;**

**}**

**else**

**{**

**struct Node \*temp = head;**

**while(temp -> next != NULL)**

**temp = temp -> next;**

**temp -> next = newNode;**

**newNode -> previous = temp;**

**}**

**printf("\nInsertion success!!!");**

**}**

**void insertAfter(int value, int location)**

**{**

**struct Node \*newNode;**

**newNode = (struct Node\*)malloc(sizeof(struct Node));**

**newNode -> data = value;**

**if(head == NULL)**

**{**

**newNode -> previous = newNode -> next = NULL;**

**head = newNode;**

**}**

**else**

**{**

**struct Node \*temp1 = head, temp2;**

**while(temp1 -> data != location)**

**{**

**if(temp1 -> next == NULL)**

**{**

**printf("Given node is not found in the list!!!");**

**goto EndFunction;**

**}**

**else**

**{**

**temp1 = temp1 -> next;**

**}**

**}**

**temp2 = temp1 -> next;**

**temp1 -> next = newNode;**

**newNode -> previous = temp1;**

**newNode -> next = temp2;**

**temp2 -> previous = newNode;**

**printf("\nInsertion success!!!");**

**}**

**EndFunction:**

**}**

**void deleteBeginning()**

**{**

**if(head == NULL)**

**printf("List is Empty!!! Deletion not possible!!!");**

**else**

**{**

**struct Node \*temp = head;**

**if(temp -> previous == temp -> next)**

**{**

**head = NULL;**

**free(temp);**

**}**

**else{**

**head = temp -> next;**

**head -> previous = NULL;**

**free(temp);**

**}**

**printf("\nDeletion success!!!");**

**}**

**}**

**void deleteEnd()**

**{**

**if(head == NULL)**

**printf("List is Empty!!! Deletion not possible!!!");**

**else**

**{**

**struct Node \*temp = head;**

**if(temp -> previous == temp -> next)**

**{**

**head = NULL;**

**free(temp);**

**}**

**else{**

**while(temp -> next != NULL)**

**temp = temp -> next;**

**temp -> previous -> next = NULL;**

**free(temp);**

**}**

**printf("\nDeletion success!!!");**

**}**

**}**

**void deleteSpecific(int delValue)**

**{**

**if(head == NULL)**

**printf("List is Empty!!! Deletion not possible!!!");**

**else**

**{**

**struct Node \*temp = head;**

**while(temp -> data != delValue)**

**{**

**if(temp -> next == NULL)**

**{**

**printf("\nGiven node is not found in the list!!!");**

**goto FuctionEnd;**

**}**

**else**

**{**

**temp = temp -> next;**

**}**

**}**

**if(temp == head)**

**{**

**head = NULL;**

**free(temp);**

**}**

**else**

**{**

**temp -> previous -> next = temp -> next;**

**free(temp);**

**}**

**printf("\nDeletion success!!!");**

**}**

**FuctionEnd:**

**}**

**void display()**

**{**

**if(head == NULL)**

**printf("\nList is Empty!!!");**

**else**

**{**

**struct Node \*temp = head;**

**printf("\nList elements are: \n");**

**printf("NULL <--- ");**

**while(temp -> next != NULL)**

**{**

**printf("%d <===> ",temp -> data);**

**}**

**printf("%d ---> NULL", temp -> data);**

**}**

**}**

**Output**

