In a single linked list we perform the following operations...

* Insertion
* Deletion
* Display

Before we implement actual operations, first we need to setup 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 4: 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

**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.

**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");**

**}**

**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.

**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");**

**}**

**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'

**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");**

**}**

**Deletion**

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

**Deleting from Beginning of the list**

**Deleting from End of the list**

**Deleting a Specific Node**

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.

**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");**

**}**

**}**

**}**

**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.

**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");**

**}**

**}**

**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)).

**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:**

**}**

**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).

**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);**

**}**

**}**

**Complete Program**

#include<stdio.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;

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 want to 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(value);

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");

}

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!!!");

}

temp2 = temp1;

temp1 = temp1 -> next;

}

temp2 -> next = temp1 -> next;

free(temp1);

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

}

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);

}

}