Aim:- Implementation of linked list

Algorithm:-

Inserting at the beginning

Step 1: Create a new node with the data to be inserted

Step 2: Set the next pointer of the new node to the current head of the list

Step 3: Update the head of the list to the new node

Inserting at a random position

Step 1: Create a new node with the data to be inserted

Step 2: Traverse the list to find the node that comes immediately before the desired position

Step 3: Set the next pointer of the new node to the next pointer of the node found in step 2

Step 4: Set the next pointer of the node found in step 2 to the new node

Inserting at the end

Step 1: Create a new node with the data to be inserted

Step 2: Traverse the list to find the last node

Step 3: Set the next pointer of the last node to the new node

Deleting at the beginning

Step 1: Store the current head of the list in a temporary variable

Step 2: Update the head of the list to the next node

Step 3: Free the memory allocated to the temporary node

Deleting at a random position

1) Traverse the list to find the node that comes immediately before the node to be deleted

2) Store the next pointer of the node to be deleted in a temporary variable

3) Set the next pointer of the previous node to the next pointer of the node to be deleted

4) Free the memory allocated to the node to be deleted

Deleting at the end

1) Traverse the list to find the second-to-last node

2) Set the next pointer of the second-to-last node to null

3) Free the memory allocated to the last node

Program:-

#include<stdio.h>//standard input output header file//

#include<stdlib.h>//standard library header file//

struct node//node creation//

{

int data;

struct node \*next;

};

struct node \*head;

void begin\_insert()//function to insert the element in the beginning//

{

struct node \*ptr;

int item;

ptr=(struct node \*)malloc(sizeof(struct node \*));

if(ptr==NULL)//checking if the linked list is full//

printf("Overflow");

else

{

printf("Enter the value : ");

scanf("%d",&item);

ptr->data=item;

ptr->next=head;

head=ptr;

printf("Node inserted\n");

}

}

void last\_insert()//function to insert the element from the last//

{

struct node \*ptr,\*temp;

int item;

ptr=(struct node \*)malloc(sizeof(struct node \*));

if(ptr==NULL)

printf("Overflow");//checking if the linked list is full//

else

{

printf("Enter the value : ");

scanf("%d",&item);

ptr->data=item;

if(head==NULL)

{

ptr->next=NULL;

head=ptr;

printf("Node insertes\n");

}

else

{

temp=head;

while(temp->next!=NULL)

{

temp=temp->next;

}

temp->next=ptr;

ptr->next=NULL;

printf("Node inserted\n");

}

}

}

void random\_insert()//function to insert the element at the position you want//

{

struct node \*ptr,\*temp;

int item,i,loc;

ptr=(struct node \*)malloc(sizeof(struct node \*));

if(ptr==NULL)

{

printf("Overflow");

}

else

{

printf("Enter value : ");

scanf("%d",&item);

ptr->data=item;

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

scanf("%d",&loc);

temp=head;

for(i=0;i<loc;i++)

{

temp=temp->next;

if(temp==NULL)

{

printf("Can't insert\n");

return;

}

}

ptr->next=temp->next;

temp->next=ptr;

printf("Node inserted\n");

}

}

void begin\_delete()//function to delete the element in the beginning//

{

struct node \*ptr;

if(head==NULL)

printf("LIST IS EMPTY ");

else

{

ptr=head;

head=ptr->next;

free(ptr);

printf("Node deleted from beginning\n");

}

}

void last\_delete()//function to delete the element from the last//

{

struct node \*ptr,\*ptr1;

if(head==NULL)

printf("list is empty");

else if(head->next==NULL)

{

head=NULL;

free(head);

printf("Only node of the list deleted \n");

}

else

{

ptr=head;

while(ptr->next!=NULL)

{

ptr1=ptr;

ptr=ptr->next;

}

ptr1->next=NULL;

free(ptr);

printf("Deleted node from the last");

}

}

void delete\_random()//function to delete the element at the position you want//

{

struct node \*ptr,\*ptr1;

int loc,i;

printf("Enter the location of the node after which it has to be deleted : ");

scanf("%d",&loc);

ptr=head;

for(i=0;i<loc;i++)

{

ptr1=ptr;

ptr=ptr->next;

if(ptr==NULL)

{

printf("Cant delete");

return ;

}

}

ptr1->next=ptr->next;

free(ptr);

printf("Deleted Node %d",loc++);

}

void display()//function to display//

{

struct node \*ptr;

ptr=head;

if(ptr==NULL)//checking if queue is empty//

{

printf("Underflow");

return ;

}

else

{

printf("Printing elements \n");

while(ptr!=NULL)

{

printf("%d\n",ptr->data);

ptr=ptr->next;

}

}

}

int search(struct node\* head, int x)//function to search for the element//

{

struct node\* current = head;

while (current != NULL)

{

if (current->data == x)

return 1;

current = current->next;

}

return 0;

}

int main()//driver function//

{

int choice,val;

while(1)

{

printf("Select the operation\n");

printf("1.Insert at beginning \n2.Insert at last\n3.Insert at random location\n4.Delete at beginning\n5.Delete at last\n6.Delete at random\n7.Display\n8.Search\n9.Exit\n");

scanf("%d",&choice);

switch(choice)

{

case 1: begin\_insert();//calling function//

break;

case 2: last\_insert();//calling function//

break;

case 3: random\_insert();//calling function//

break;

case 4: begin\_delete();//calling function//

break;

case 5: last\_delete();//calling function//

break;

case 6: delete\_random();//calling function//

break;

case 7: display();//calling function//

break;

case 8: printf("Enter the value you want to search: ");

scanf("%d",&val);

int check = search(head,val);

if(check==1)

printf("Element is present\n");

else

printf("Element not found\n");

break;

case 9: exit(0);

default: printf("Invalid choice");

break;

}

}

return 0;

}

Screenshot of the output:-



