* Problem **A**

**Singly Linked List**

**AIM :-** To write a ‘c’ Program to perform operations on **Singly Linked List** of Insert, Delete and Display**.**

**Theory :-** A **Singly Linked List** is also a **Linear Data Structure** which Stores the data. The **Data** and **Address** of the next list is stored in the block of memory called **NODE** which is **Dynamically Allocated**. The **NODES** are created anywhere in the memory block which are linked with their respective **Addresses.**

**Algorithms :-**

Declare a Structure node and create there members of int type ‘data’ and of struct type ‘next’ pointer.

Declare struct type Global Pointer Variables struct node **\*head** and struct node **\*tail** pointing to **NULL**.

**UDF(User Defined Function) start\_insert()**

Step 1:- Start.

Step 2:- Declare a pointer variable ‘new\_node’ of struct node datatype and dynamically allocate it **new\_node=(struct node\*)malloc(sizeof(struct node))**.

Step 3:- Read value for ‘new\_node->data’.

Step 4:- Assign **new\_node->next=head** and **head=new\_node**.

Step 5:- Stop.

**UDF last\_insert ()**

Step 1:- Start.

Step 2:- Declare a pointer variable ‘new\_node’ and ‘temp’ of struct node datatype and dynamically allocate it **new\_node=(struct node\*)malloc(sizeof(struct node))**.

Step 3:- Read value for ‘new\_node->data’.

Step 4:- Check **IF** Condition **head==tail**?, if true go to step 5 otherwise step 6.

Step 5:- Assign **new\_node->next=tail** and **head=new\_node**.

Step 6:- Else.

Step 7:- Assign **temp=head**.

Step 8:- Check **while** condition temp->next!=tail, if true go to step 9 otherwise step 10.

Step 9:- Assign **temp->next=new\_node** and **new\_node->next=tail**.

Step 10:- Stop.

**UDF random\_insert ()**

Step 1:- Start.

Step 2:- Declare integer variables ‘i’, ‘pos’.

Step 3:- Declare a pointer variable ‘new\_node’ and ‘temp’ of struct node datatype and dynamically allocate it **new\_node=(struct node\*)malloc(sizeof(struct node))**.

Step 4:- Check **IF** condition new\_node== NULL?, if true go to step 5 otherwise step 6.

Step 5:- Print OVERFLOW.

Step 6:- Else.

Step 7:- Read value in ‘new\_node->data’.

Step 8:- Read value for ‘Pos’.

Step 9:- Assign temp=head.

Step 10:- Check **for** loop condition, if i<pos go to step 11 otherwise step .

Step 11:- Assign **temp=temp->next**.

Step 12:- Check **if condition**, if temp=tail go to step 13 otherwise step 14.

Step 13:- Print Can’t insert.

Step 14:- Assign **new\_node->next=temp->next** and **temp->next=new\_node**.

Step 15:- Stop.

**UDF start\_delete ()**

Step 1:- Start.

Step 2:- Declare a pointer variable ‘new\_node’ of struct node datatype.

Step 3:- Check **if** condition, if **head==tail** go to step 4 otherwise go to step 5.

Step 4:- Print “List is Empty”.

Step 5:- Else.

Step 6:- Assign **new\_node=head** and **head=new\_node->next**.

Step 7:- Delete the node using free function **free(new\_node)**.

Step 8:- Stop.

**UDF last\_delete ()**

Step 1:- Start.

Step 2:- Declare a pointer variable ‘new\_node’ and ‘temp’ of struct node datatype.

Step 3:- Assign **new\_node=head**.

Step 4:- Check **if** condition, if **head==tail** is true go to step 5 otherwise step 6.

Step 5:- Print “List is Empty”.

Step 6:- Check **else if** condition, if **new\_node->next==tail** go to step 7 otherwise step 9.

Step 7:- Assign **head=tail**.

Step 8:- Delete the node using free function **free(new\_node)**.

Step 9:- Else.

Step 10:- Check **while** condition, if **new\_node->next!=tail** go to step 11 otherwise step 12.

Step 11:- Assign **temp=new\_node** and **new\_node=new\_node->next**.

Step 12:- Assign **temp->next=tail**.

Step 13:- Delete the node using free function **free(new\_node)**.

Step 14:- Stop.

**UDF random\_delete ()**

Step 1:- Start.

Step 2:- Declare a pointer variable ‘**new\_node**’ and ‘**temp**’ of struct node datatype.

Step 3:- Declare integer variables ‘**pos**’ and ‘**i**’.

Step 4:- Read value for ‘**pos**’.

Step 5:- Assign **new\_node=head**.

Step 6:- Check **for loop** condition, if **i<pos** go to step 7 otherwise step 10.

Step 7:- Assign **temp=new\_node** and **new\_node=new\_node->next**.

Step 8:- Check **if** condition, if **new\_node ==NULL** go to step 9 otherwise step 10.

Step 9:- Print “Can’t Delete”.

//End of **for loop**

Step 10:-Assign **temp->next=new\_node->next**

**Step 11:-** Delete the node using free function **free(new\_node)**.

Step 12:- Stop.

**UDF display ()**

Step 1:- Start.

Step 2:- Declare a pointer variable ‘**temp**’ of struct node datatype.

Step 3:- Assign **temp=head**.

Step 4:- Check **if** condition, if **temp==tail** go to step 5 otherwise step 6.

Step 5:- Print “Nothing to Print”.

Step 6:- Else.

Step 7:- Print values using **While loop**.

Step 8:- Check **while loop** condition, if **temp!=tail** go to step 9 otherwise step 11.

Step 9:- Print the values of **‘temp->data’**.

Step 10:- Assign **temp=temp->next.**

Step 11:- Stop.

**Main()**

Step 1:- Start.

Step 2:- Declare ‘**choice**’ integer variable.

Step 3:- Check **while loop** condition, if choice!=8 go to step 4 otherwise step .

Step 4:- Read Value for ‘choice’ variable from 1 to 8 for different operations.

Step 5:- Check **if** condition, if choice==1 go to step 6 otherwise step 7.

Step 6:- Call “**start\_insert()**” function.

Step 7:- Check **else if** condition, if choice==2 go to step 8 otherwise step 9.

Step 8:- Call “**last\_insert()**” function.

Step 9:- Check **else if** condition, if choice==3 go to step 10 otherwise step 11.

Step 10:- Call “**random\_insert()**” function.

Step 11:- Check **else if** condition, if choice==4 go to step 12 otherwise step 13.

Step 12:- Call “**start\_delete()**” function.

Step 13:- Check **else if** condition, if choice==5 go to step 14 otherwise step 15.

Step 14:- Call “**last\_delete()**” function.

Step 15:- Check **else if** condition, if choice==6 go to step 16 otherwise step 17.

Step 16:- Call “**random\_delete()**” function.

Step 17:- Check **else if** condition, if choice==7 go to step 18 otherwise step 19.

Step 18:- Call “**display()**” function.

Step 19:- Check **else if** condition, if choice==8 go to step 20 otherwise step 21.

Step 20:- Terminate the loop using ‘**break’ statement**.

Step 21:- **Else**.

Step 22:- Print “Invalid Value!!!”.

//End loop

Step 6:- Stop.

**Program :-**

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*next;

};

struct node \*head;

struct node \*tail;

void start\_insert()

{

struct node \*new\_node;

new\_node=(struct node\*)malloc(sizeof(struct node));

printf("\n\nEnter the value in the node??\n");

scanf("%d",&new\_node->data);

new\_node->next=head;

head=new\_node;

printf("Node Inserted\n");

}

void last\_insert()

{

struct node \*new\_node, \*temp;

new\_node=(struct node\*)malloc(sizeof(struct node));

printf("\n\nEnter the value in the node??\n");

scanf("%d",&new\_node->data);

if(head==tail)

{

new\_node->next=tail;

head=new\_node;

printf("\nNode inserted");

}

else

{

temp=head;

while(temp->next!=tail)

{

temp=temp->next;

}

temp->next=new\_node;

new\_node->next=tail;

printf("Node Inserted\n");

}

}

void random\_insert()

{

int i,pos,value;

struct node \*ptr, \*temp;

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

if(ptr==NULL)

{

printf("\nOVERFLOW");

}

else

{

printf("\nEnter Data value\n");

scanf("%d",&value);

ptr->data=value;

printf("\nEnter the location after which you want to insert???\n");

scanf("%d",&pos);

temp=head;

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

{

temp=temp->next;

if(temp==NULL)

{

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

return;

}

}

ptr->next=temp->next;

temp->next=ptr;

printf("\nNode inserted");

}

}

void start\_delete()

{

struct node \*ptr;

if(head==tail)

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

else

{

ptr=head;

head=ptr->next;

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

free(ptr);

}

}

void last\_delete()

{

struct node \*ptr,\*temp;

ptr=head;

if(head==tail)

{

printf("\nlist is empty");

}

else if(ptr->next==tail)

{

head=tail;

free(ptr);

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

}

else

{

while(ptr->next!=tail)

{

temp=ptr;

ptr=ptr->next;

}

temp->next=tail;

free(ptr);

printf("\nDeleted Node from the last ...\n");

}

}

void random\_delete()

{

struct node \*ptr,\*temp;

int pos,i;

printf("\nEnter the location of the node after which you want to perform deletion \n");

scanf("%d",&pos);

ptr=head;

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

{

temp=ptr;

ptr=ptr->next;

if(ptr==NULL)

{

printf("\nCan't delete");

return;

}

}

temp->next=ptr->next;

free(ptr);

printf("\nDeleted node from %d location",pos+1);

}

void display()

{

struct node \*temp;

temp=head;

if(temp==tail)

printf("\n\nNothing to print");

else

{

printf("\n\nprinting values . . . . .\n");

while(temp!=tail)

{

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

temp=temp->next;

}

}

}

int main()

{

int choice;

while(choice!=6)

{

printf("\n\nChoose one option from the following list ...\n");

printf("\n1.To Insert at Start\n2.To Insert at Last\n3.To Insert at Random Location\n4.To Delete a Node from Start\n5.To Delete a Node from Last\n6.To Delete a Node from Random Location\n7.To Display and Traverse\n8.To Exit\n\n");

scanf("%d",&choice);

if(choice==1)

start\_insert();

else if(choice==2)

last\_insert();

else if(choice==3)

random\_insert();

else if(choice==4)

start\_delete();

else if(choice==5)

last\_delete();

else if(choice==6)

random\_delete();

else if(choice==7)

display();

else if(choice==8)

break;

else

printf("\n\nInvalid value!!!!\n");

}

return 0;

}

**Output :-**

Choose one option from the following list ...

1.To Insert at Start

2.To Insert at Last

3.To Insert at Random Location

4.To Delete a Node from Start

5.To Delete a Node from Last

6.To Delete a Node from Random Location

7.To Display and Traverse

8.To Exit

2

Enter the value in the node??

1

Node inserted

Tell Your choice from above

2

Enter the value in the node??

2

Node Inserted

Tell Your choice from above

2

Enter the value in the node??

3

Node Inserted

Tell Your choice from above

2

Enter the value in the node??

4

Node Inserted

Tell Your choice from above

2

Enter the value in the node??

5

Node Inserted

Tell Your choice from above

2

Enter the value in the node??

6

Node Inserted

Tell Your choice from above

5

Deleted Node from the last ...

Tell Your choice from above

4

1 Data

Node Deleted

Tell Your choice from above

7

printing values . . . . .

2 3 4 5

Tell Your choice from above

6

Enter the location of the node on which you want to perform deletion

2

Deleted node from 2 location

Tell Your choice from above

7

printing values . . . . .

2 3 5

Tell Your choice from above

8

**Observation :-** After performing the experiment we observed that Values can be stored in a memory in any location and we were able to link them, so they can be accessed easily one after the other. Insertion and Deletion of Values can happen from anywhere like from starting or Ending or Randomly. It took 0.16sec Compilation Time.