|  |  |
| --- | --- |
|  | **DEPARTMENT OF COMPUTER ENGINEERING** |

**Experiment No. 08**

|  |  |
| --- | --- |
| Semester | S.E. Semester III – Computer Engineering |
| Subject | Data Structures Lab (CSL301) |
| Subject Professor In-charge | Prof. Swapnil S. Sonawane |
| Assisting Teachers | Prof. Swapnil S. Sonawane |

|  |  |
| --- | --- |
| Student Name | Trisha Shah |
| Roll Number | 20102A0004 |

**Title:**

Implement Singly Linked List ADT.

**Objective:**

Students will be able to implement Linear and Non-Linear data structures.

**Explanation:**

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.

Linked List can be defined as collection of objects called nodes that are randomly stored in the memory.

A node contains two fields i.e. data stored at that particular address and the pointer which contains the address of the next node in the memory.

The last node of the list contains pointer to the null.

The list is not required to be contiguously present in the memory. The node can reside any where in the memory and linked together to make a list. This achieves optimized utilization of space.

List size is limited to the memory size and doesn't need to be declared in advance.

Empty node cannot be present in the linked list.

We can store values of primitive types or objects in the singly linked list.

**Program Code:**

#include<stdio.h>

#include<stdlib.h>

typedef struct node

{

int data;

struct node \*next;

}node;

typedef struct LL

{

node \*start;

}LL;

int count(LL \*l)

{

node \*p;

int c=0;

p=l->start;

while(p!=NULL)

{

c=c+1;

p=p->next;

}

return c;

}

void display(LL \*l)

{

node \*p;

if(l->start==NULL)

{

printf("\nList is empty...");

}

else

{

p=l->start;

while(p!=NULL)

{

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

p=p->next;

}

}

}

void insertbegin(LL \*l,int x)

{

node \*newrec;

newrec=(node \*)malloc(sizeof(node));

newrec->data=x;

newrec->next=NULL;

if(l->start==NULL)

{

l->start=newrec;

}

else

{

newrec->next=l->start;

l->start=newrec;

}

}

void insertend(LL \*l,int x)

{

node \*newrec,\*p;

newrec=(node \*)malloc(sizeof(node));

newrec->data=x;

newrec->next=NULL;

if(l->start==NULL)

{

l->start=newrec;

}

else

{

p=l->start;

while(p->next!=NULL)

{

p=p->next;

}

p->next=newrec;

}

}

void insertatpos(LL \*l,int x,int pos)

{

int i;

node \*newrec,\*p;

newrec=(node \*)malloc(sizeof(node));

newrec->data=x;

newrec->next=NULL;

if(pos>count(l)+1)

{

printf("\nInvalid Position...");

}

else if(pos==1)

{

newrec->next=l->start;

l->start=newrec;

}

else

{

p=l->start;

for(i=1;i<pos-1;i++)

{

p=p->next;

}

newrec->next=p->next;

p->next=newrec;

}

}

void deletebegin(LL \*l)

{

node \*p;

if(l->start==NULL)

{

printf("\nDeletion not possible...");

}

else

{

p=l->start;

l->start=l->start->next;

free(p);

}

}

void deleteend(LL \*l)

{

node \*p,\*q;

if(l->start==NULL)

{

printf("\nDeletion not possible...");

}

else if(l->start->next==NULL)//only 1 node

{

p=l->start;

l->start=NULL;

free(p);

}

else

{

q=l->start;

while(q->next->next!=NULL)

{

q=q->next;

}

p=q->next;

q->next=NULL;

free(p);

}

}

void deleteatpos(LL \*l,int pos)

{

int i;

node \*p,\*q;

if(pos>count(l))

{

printf("\nInvalid position...");

}

else if(pos==1)

{

p=l->start;

l->start=l->start->next;

free(p);

}

else

{

q=l->start;

for(i=1;i<pos-1;i++)

{

q=q->next;

}

p=q->next;

q->next=p->next;

free(p);

}

}

int search(LL \*l,int x)

{

node \*p;

p=l->start;

while(p!=NULL)

{

if(x==p->data)

{

return 1;

}

else

{

p=p->next;

}

}

return 0;

}

void sort(LL \*l)

{

int temp;

node \*i,\*j;

for(i=l->start;i->next!=NULL;i=i->next)

{

for(j=l->start;j->next!=NULL;j=j->next)

{

if(j->data>j->next->data)

{

temp=j->data;

j->data=j->next->data;

j->next->data=temp;

}

}

}

}

void reverse(LL \*l)

{

node \*p,\*q,\*r;

p=l->start;

q=NULL;

while(p!=NULL)

{

r=p->next;

p->next=q;

q=p;

p=r;

}

l->start=q;

}

int main()

{

int ch,x,pos,ele;

LL l;

l.start=NULL;

while(1)

{

printf("\nMenu:\n1-Insert at beginning\n2-Insert at end\n3-Insert at specific position\n4-Delete at beginning\n5-Delete at end\n6-Delete at position\n7-Count\n8-Search\n9-Sort\n10-Reverse\n11-Display\n12-EXIT\nEnter Choice=");

scanf("%d",&ch);

if(ch==12)

break;

switch(ch)

{

case 1:

{

printf("\nEnter element to be inserted=");

scanf("%d",&x);

insertbegin(&l,x);

display(&l);

}

break;

case 2:

{

printf("\nEnter element to be inserted=");

scanf("%d",&x);

insertend(&l,x);

display(&l);

}

break;

case 3:

{

printf("\nEnter element to be inserted and position=");

scanf("%d%d",&x,&pos);

insertatpos(&l,x,pos);

display(&l);

}

break;

case 4:

{

deletebegin(&l);

display(&l);

}

break;

case 5:

{

deleteend(&l);

display(&l);

}

break;

case 6:

{

printf("\nEnter position of element to be deleted=");

scanf("%d",&pos);

deleteatpos(&l,pos);

display(&l);

}

break;

case 7:

{

ele=count(&l);

printf("\nNumber of nodes=%d",ele);

}

break;

case 8:

{

printf("\nEnter element to be searched=");

scanf("%d",&x);

ele=search(&l,x);

if(ele==1)

{

printf("\nElement is present...");

}

else

{

printf("\nElement is not present...");

}

}

break;

case 9:

{

sort(&l);

display(&l);

}

break;

case 10:

{

reverse(&l);

display(&l);

}

break;

case 11:

{

display(&l);

}

break;

default:

{

printf("\nInvalid Choice...");

}

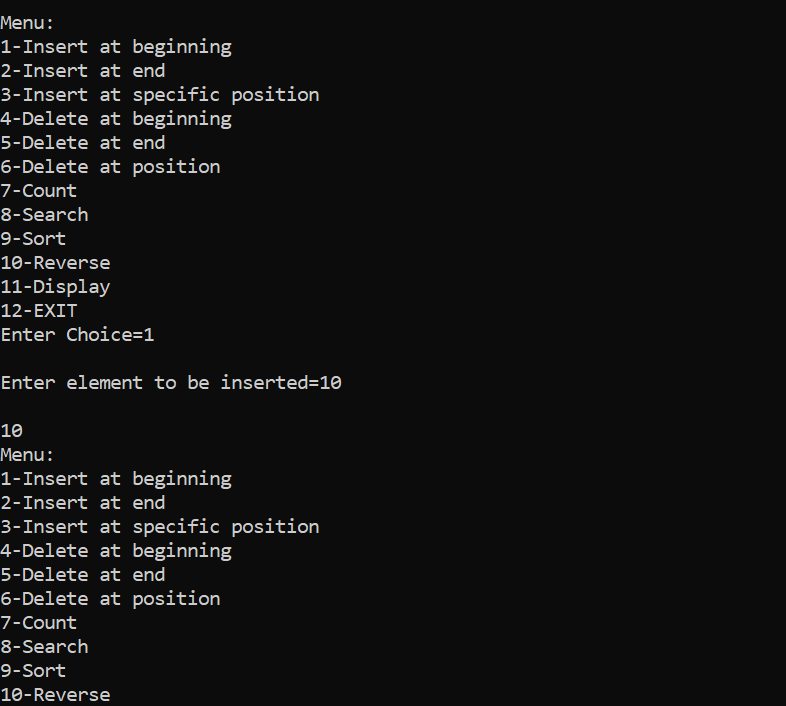
}

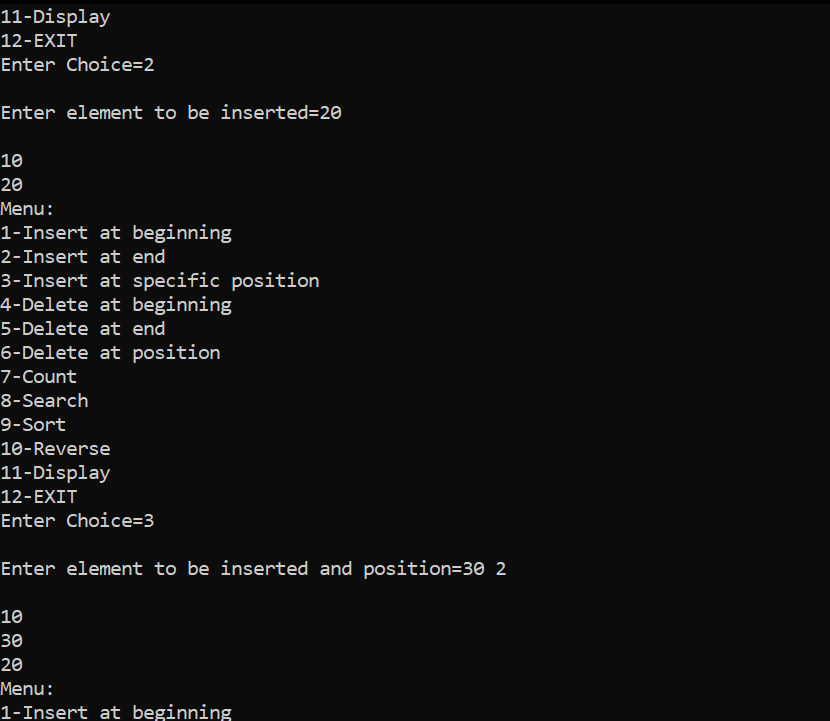
}

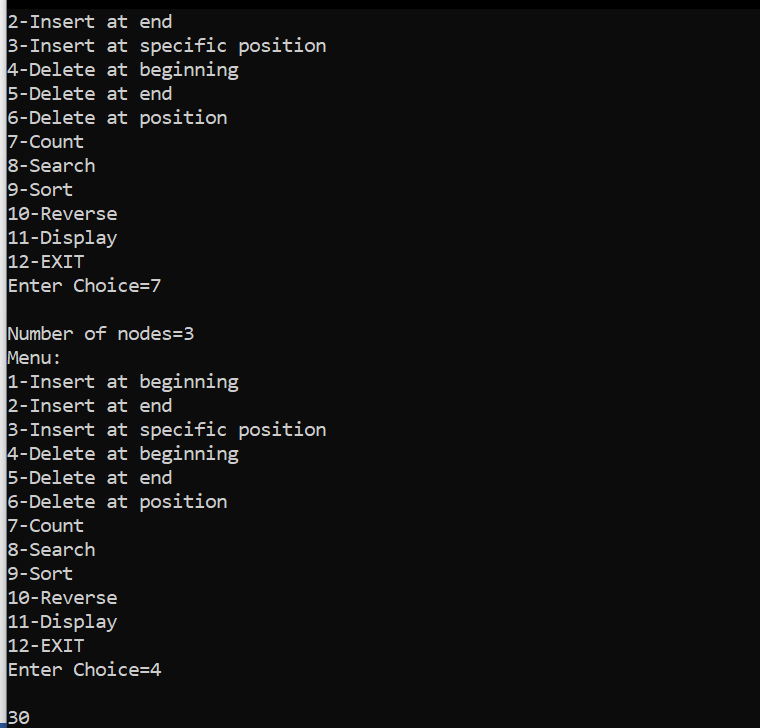
return 0;

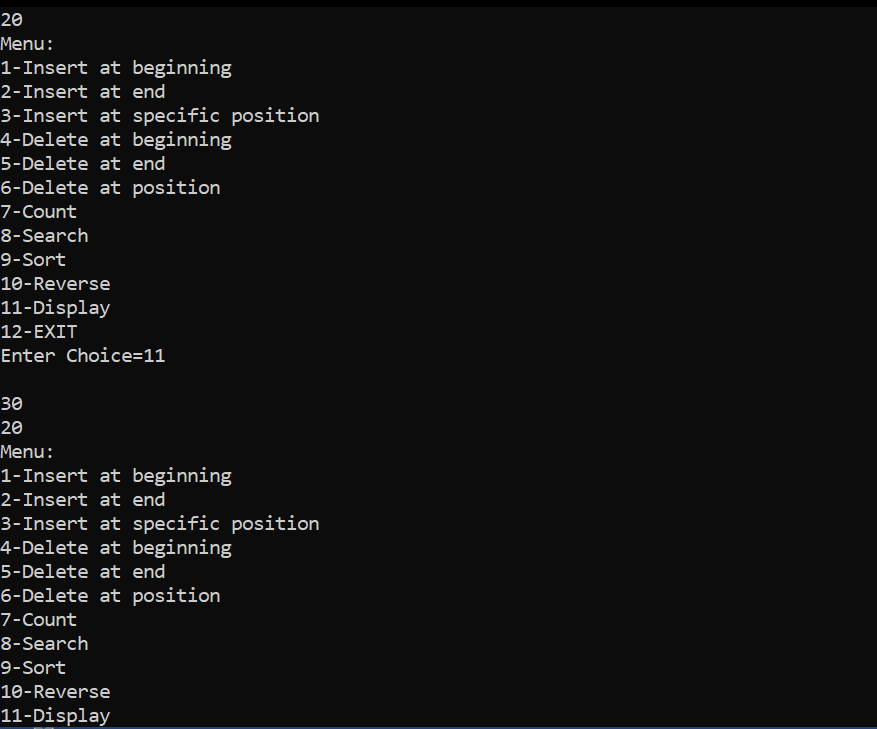
}

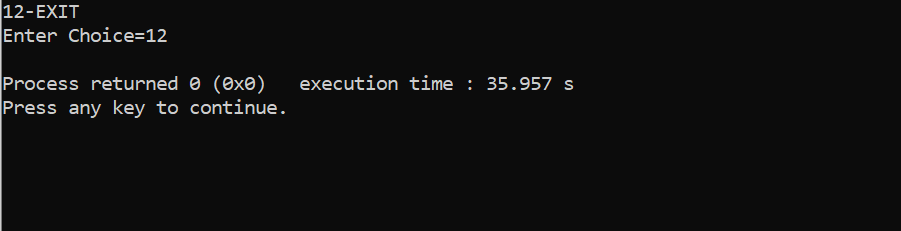
**Output:**



****

****

****

****

**Conclusion:**

Through this experiment, we could successfully implement a singly linkedlist.