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

**Experiment No. 10**

|  |  |
| --- | --- |
| 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 Doubly Linked List ADT.

**Objective:**

Students will be able to explain various data structures, related terminologies and its types.

**Explanation:**

A Doubly Linked List (DLL) contains an extra pointer, typically called previous pointer, together with next pointer and data which are there in singly linked list.

Doubly linked list is a complex type of linked list in which a node contains a pointer to the previous as well as the next node in the sequence. Therefore, in a doubly linked list, a node consists of three parts: node data, pointer to the next node in sequence (next pointer) pointer to the previous node (previous pointer). A sample node in a doubly linked list is shown in the figure.

The prev part of the first node and the next part of the last node will always contain null indicating end in each direction.

In a singly linked list, we could traverse only in one direction, because each node contains address of the next node and it doesn't have any record of its previous nodes. However, doubly linked list overcome this limitation of singly linked list. Due to the fact that, each node of the list contains the address of its previous node, we can find all the details about the previous node as well by using the previous address stored inside the previous part of each node.

**Program Code:**

#include<stdio.h>

#include<stdlib.h>

typedef struct node

{

int data;

struct node \*next,\*prev;

}node;

typedef struct LL

{

node \*start;

}LL;

int count(LL \*l)

{

int c=0;

node \*p;

p=l->start;

while(p!=NULL)

{

c++;

p=p->next;

}

return c;

}

void insertbegin(LL \*l,int x)

{

node \*newrec;

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

newrec->data=x;

newrec->next=NULL;

newrec->prev=NULL;

if(l->start==NULL)

{

l->start=newrec;

}

else

{

newrec->next=l->start;

l->start->prev=newrec;

l->start=newrec;

}

}

void insertend(LL \*l,int x)

{

node \*newrec,\*p;

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

newrec->data=x;

newrec->next=NULL;

newrec->prev=NULL;

if(l->start==NULL)

{

l->start=newrec;

}

else

{

p=l->start;

while(p->next!=NULL)

{

p=p->next;

}

p->next=newrec;

newrec->prev=p;

}

}

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

{

int i;

node \*newrec,\*p,\*q;

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

newrec->data=x;

newrec->next=NULL;

newrec->prev=NULL;

if(pos>count(l)+1)

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

else if(pos==1)

{

newrec->next=l->start;

l->start->prev=newrec;

l->start=newrec;

}

else

{

p=l->start;

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

{

p=p->next;

}

q=p->next;

p->next=newrec;

newrec->prev=p;

newrec->next=q;

q->prev=newrec;

}

}

void deletebegin(LL \*l)

{

node \*p;

if(l->start==NULL)

{

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

}

else

{

p=l->start;

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

l->start->prev=NULL;

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;

p->prev=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;

l->start->prev=NULL;

free(p);

}

else

{

q=l->start;

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

{

q=q->next;

}

p=q->next;

q->next=p->next;

p->next->prev=q;

free(p);

}

}

void display(LL \*l)

{

int c;

node \*p;

if(l->start==NULL)

{

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

}

else

{

printf("\n1-Forward Direction\n2-Backword Direction\nEnter Choice=");

scanf("%d",&c);

if(c==1)

{

p=l->start;

while(p!=NULL)

{

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

p=p->next;

}

}

else if(c==2)

{

p=l->start;

while(p->next!=NULL)

{

p=p->next;

}

while(p!=NULL)

{

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

p=p->prev;

}

}

else

{

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

}

}

}

int main()

{

int ch,x,pos;

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 specific position\n7-Display\n8-EXIT\nEnter Choice=");

scanf("%d",&ch);

if(ch==8)

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:

{

display(&l);

}

break;

default:

{

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

}

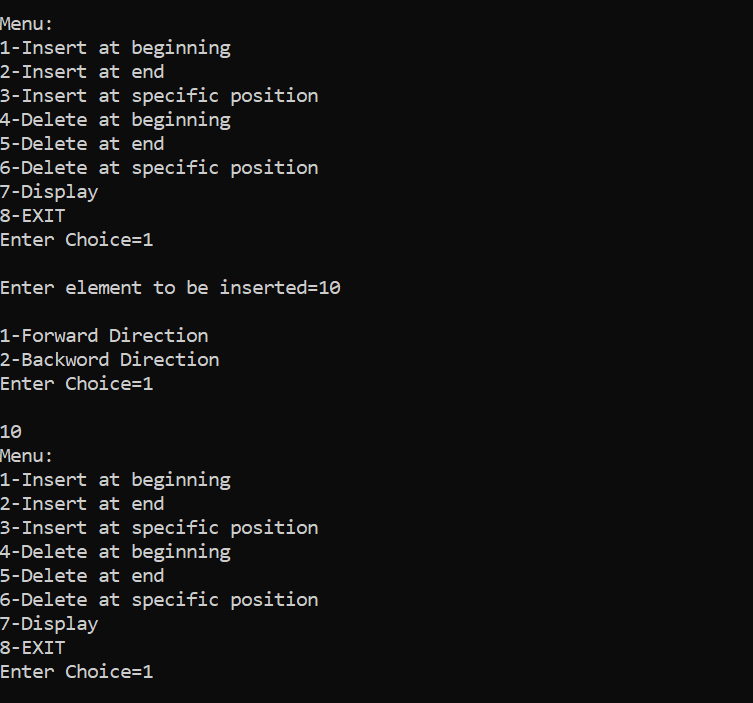
}

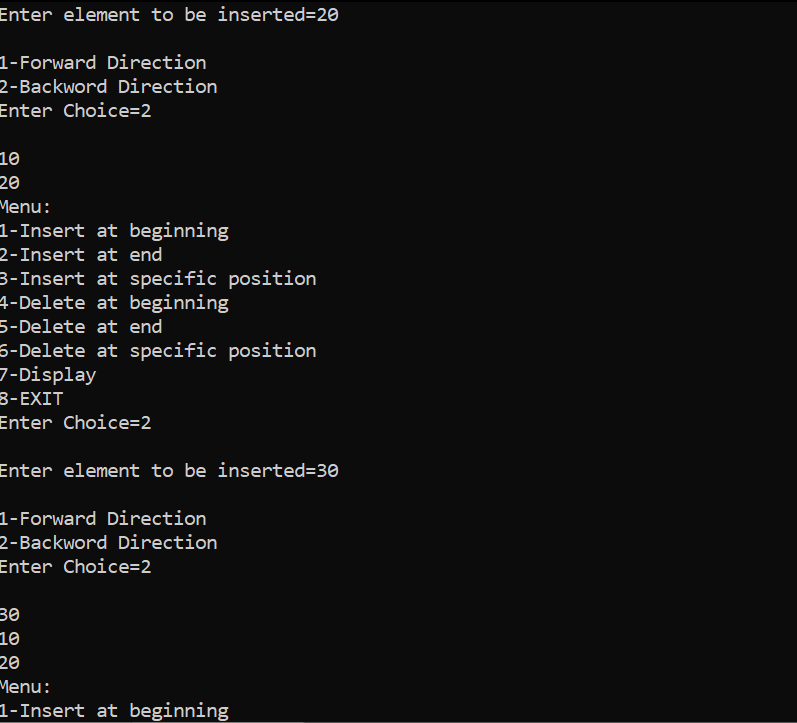
}

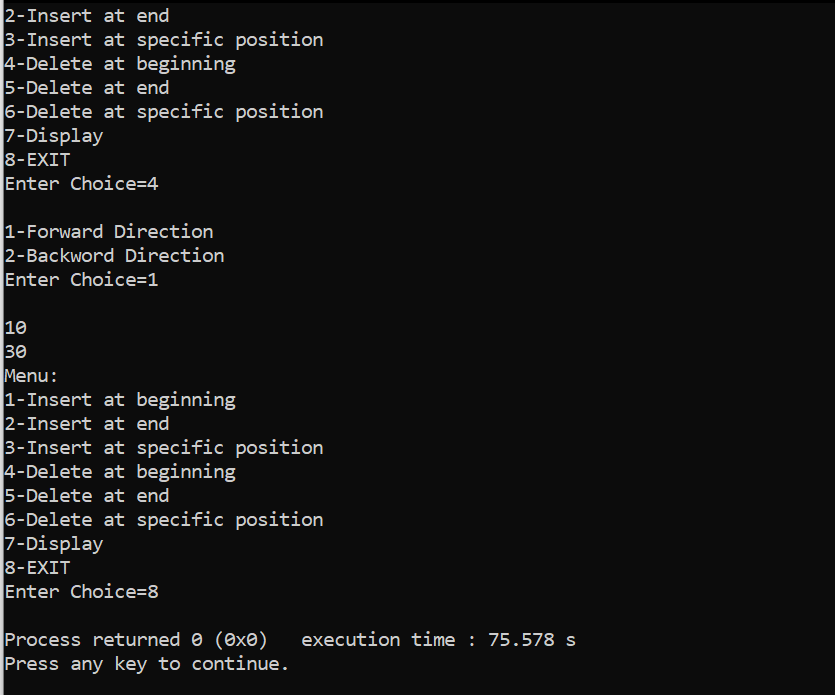
return 0;

}

**Output:**

****

****

****

**Conclusion:**

Through this experiment, students could successfully demonstrate doubly linked list.