VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

PREETI T KORISHETTAR
(1BM22CS208)

in partial fulfilment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
Dec 2023- March 2024

B. M. S. College of Engineering,
Bull Temple Road, Bangalore 560019
(Affiliated To Visvesvaraya Technological University, Belgaum)
Department of Computer Science and Engineering

This is to certify that the Lab work entitled **“DATA STRUCTURES”** carried out by Preeti T Korishettar **(1BM22CS208)**, who is a bonafied student of **B. M. S. College of Engineering**. It is in partial fulfilment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 202324. The Lab report has been approved as it satisfies the academic requirements in respect of Data structures Lab - **(23CS3PCDST) work** prescribed for the said degree.

Prof. Sneha S Bagalkot
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi S Nayak
Professor and Head
Department of CSE
BMSCE, Bengaluru

Index Sheet

Course outcomes:

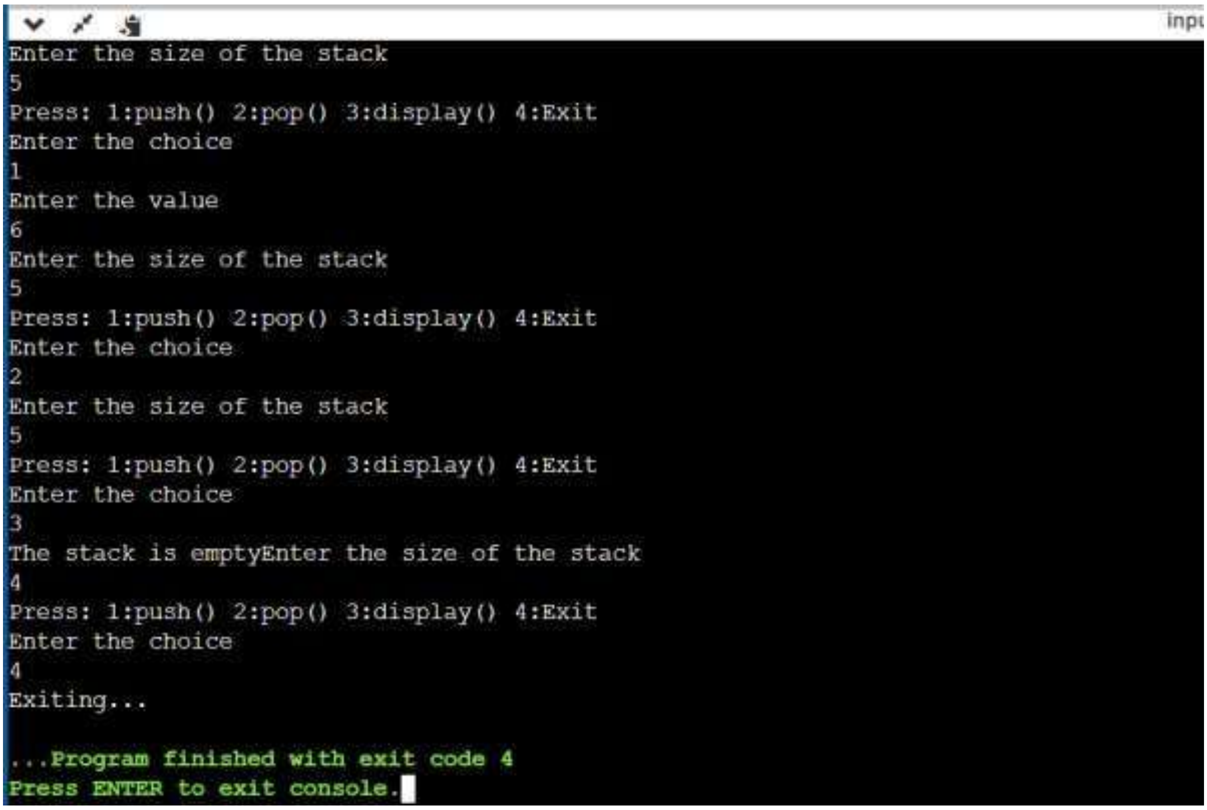
CO1	Apply the concept of linear and nonlinear data structures.
CO2	Analyze data structure operations for a given problem
CO3	Design and develop solutions using the operations of linear and nonlinear data structure for a given specification.
CO4	Conduct practical experiments for demonstrating the operations of different data structures.

Lab program 1:

Write a program to simulate the working of stack using an array with the following: a) Push
b) Pop
c) Display

```
#include <stdio.h> int
stack[100],i,n,ch=0,top=-1;
void push(); void pop(); void
display(); void main()
{
while(ch!=4)
{
printf("Enter the size of the stack\n");
scanf("%d",&n);
printf("Press:\n1:push()\n2:pop()\n3:display()\n4:Exit\n");
printf("Enter the choice\n"); scanf("%d",&ch); switch(ch)
{
case 1: push();
break; case 2:
pop(); break;
case 3:display();
break;
case 4:printf("Exiting...");
break;
default : printf("Press valid choice");
}
} }
void push()
{ if(top==n)
{
printf("The stack is full\n");
} else
{ int
value
;
printf
("Ent
er the
value
\n");
scanf
("%d
",&v
alue);
top=t
op+1
;
stack
[top]
=valu
e;
} } void pop() { if(top==-1)
{ printf("Underflow"); }
else { top=top-1; } } void
display() { if(top==-1) {
printf("The stack is empty");
} else { for(i=top;i>=0;i--)
printf("%d\n",stack[i]);
}
}
```

Output:



```
Enter the size of the stack
5
Press: 1:push() 2:pop() 3:display() 4:Exit
Enter the choice
1
Enter the value
6
Enter the size of the stack
5
Press: 1:push() 2:pop() 3:display() 4:Exit
Enter the choice
2
Enter the size of the stack
5
Press: 1:push() 2:pop() 3:display() 4:Exit
Enter the choice
3
The stack is emptyEnter the size of the stack
4
Press: 1:push() 2:pop() 3:display() 4:Exit
Enter the choice
4
Exiting...

...Program finished with exit code 4
Press ENTER to exit console.
```

Lab program 2:

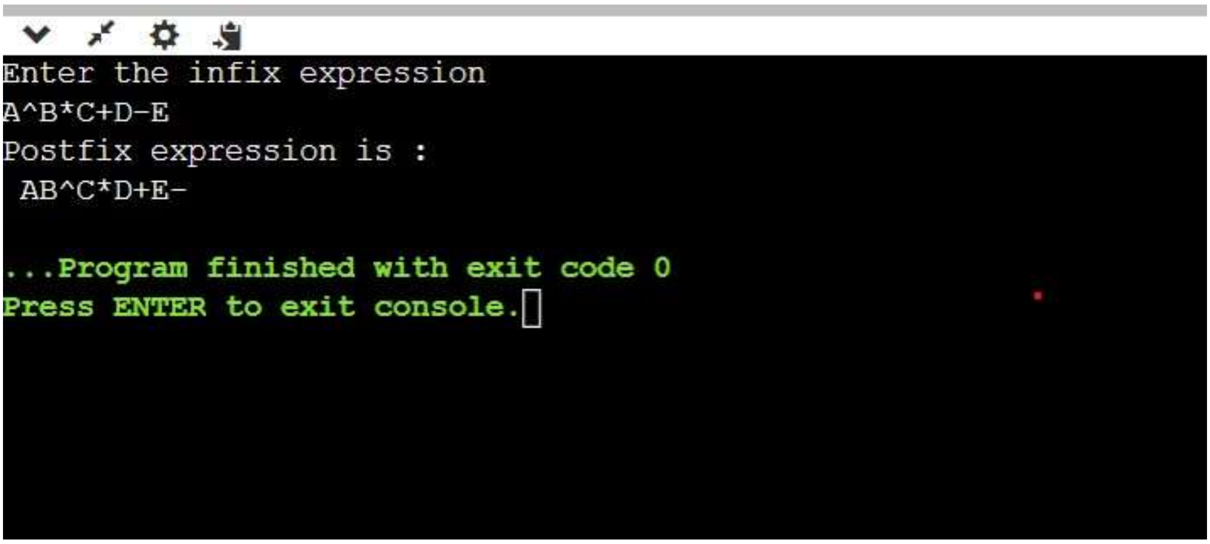
WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + (plus), - (minus), * (multiply) and /(divide)

```
#include <stdio.h>

#include <string.h>

#include <ctype.h> char

st[100];

#define max 100

int top=-1; char

ele; void push(char

ele)

{
if(top== (max)-1) {

printf("Stack is full");

} else {

top++; st[top]=ele; } }

char pop() { if(top==-1)

{ printf("Stack is

empty"); } else {

top=top-1;

return(st[top+1]); } }

char pre( char a) {

if(a=='^') { return(5);

} else if(a=='*') {

return(4); } else

if(a=='/') {

return(3); } else

if(a=='+') {
```

```
return(2); } else
if(a=='-') {
return(1); } else {
return(0); } } int
main() { char
postfix[100]; char
infix[100]; int
i=0; printf("Enter
the infix
expression\n");
scanf("%s",infix)
;
while(infix[i]!='\
0')
{ if(isalpha(infix[i]))
{ postfix[i]=infix[i];
} else
if(infix[i]=='^'||infix[i]=='*'||infix[i]=='/'||infix[i]=='+'||infix[i]=='-')
{ if(pre(st[top])>pre(infix[i]))
{ postfix[i]=pop();
push(infix[i]);
} else {
push(infix[i]);
} } i++; } postfix[i]=st[top];
printf(“Postfix expression is : \n”);
for(i=0;i<=strlen(infix)+1;i++)
```

```
{ printf("%c",postfix[i]);

}


}
```

Output:



```
Enter the infix expression
A^B*C+D-E
Postfix expression is :
 AB^C*D+E-

...Program finished with exit code 0
Press ENTER to exit console.
```

Output:

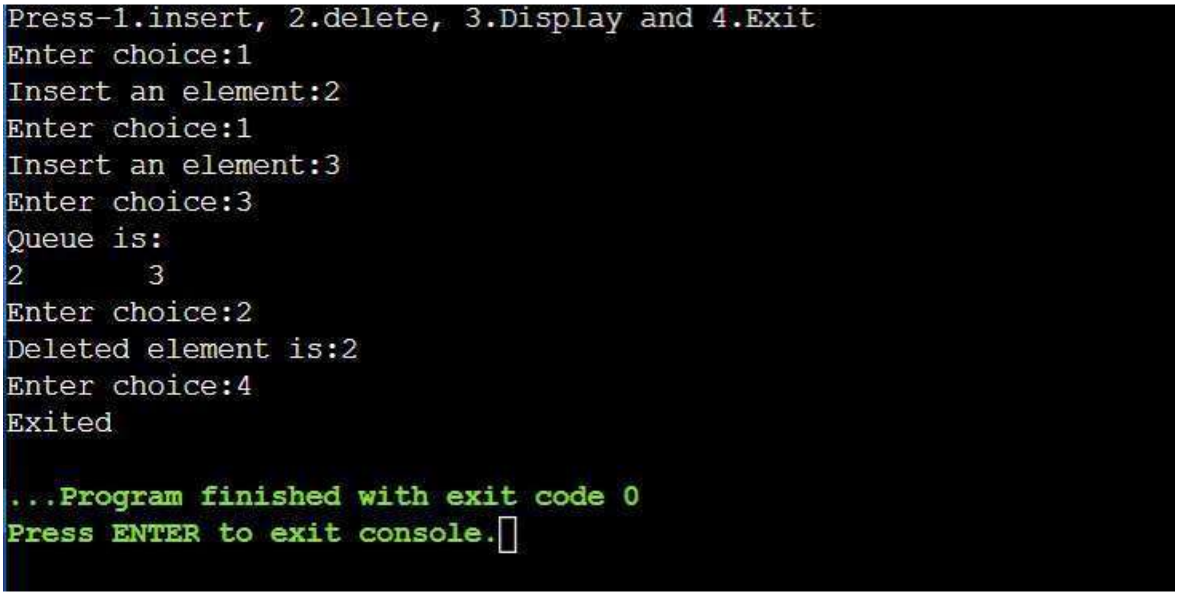
Lab program 3:

3a) WAP to simulate the working of a queue of integers using an array. Provide the following operations: Insert, Delete, Display The program should print appropriate messages for queue empty and queue overflow conditions

```
#include <stdio.h>
int q[50],rear=-1,front=-1,max=50;
void enqueue(); void dequeue();
void display();
void main()
{
int ch;
printf("Press-1.insert, 2.delete, 3.Display and 4.Exit\n"); while(ch!=4)
{
printf("Enter choice:");
scanf("%d",&ch);
switch(ch){ case 1:
enqueue(); break;
case 2:
```

```
dequeue();
break; case
3:
display();
break;
}
}
printf("Exited");
}
void enqueue()
{
int item; if(rear==max-1)
printf("Queue overflow\n");
else { if(front==-1) front=0;
printf("Insert an element:");
scanf("%d",&item);
rear+=1; q[rear]=item;
}
}
void dequeue()
{
if(front==-1||front>rear)
printf("Queue underflow\n");
else
{
printf("Deleted element is:%d\n",q[front]); front+=1;
}
}
void display()
{ int
i;
if(front==-1) printf("Queue
is empty");
else
{
printf("Queue is:\n");
for(i=front;i<=rear;i++)
printf("%d\t",q[i]);
printf("\n");
}
}
```

Output:



```
Press-1.insert, 2.delete, 3.Display and 4.Exit
Enter choice:1
Insert an element:2
Enter choice:1
Insert an element:3
Enter choice:3
Queue is:
2       3
Enter choice:2
Deleted element is:2
Enter choice:4
Exited

...Program finished with exit code 0
Press ENTER to exit console.
```

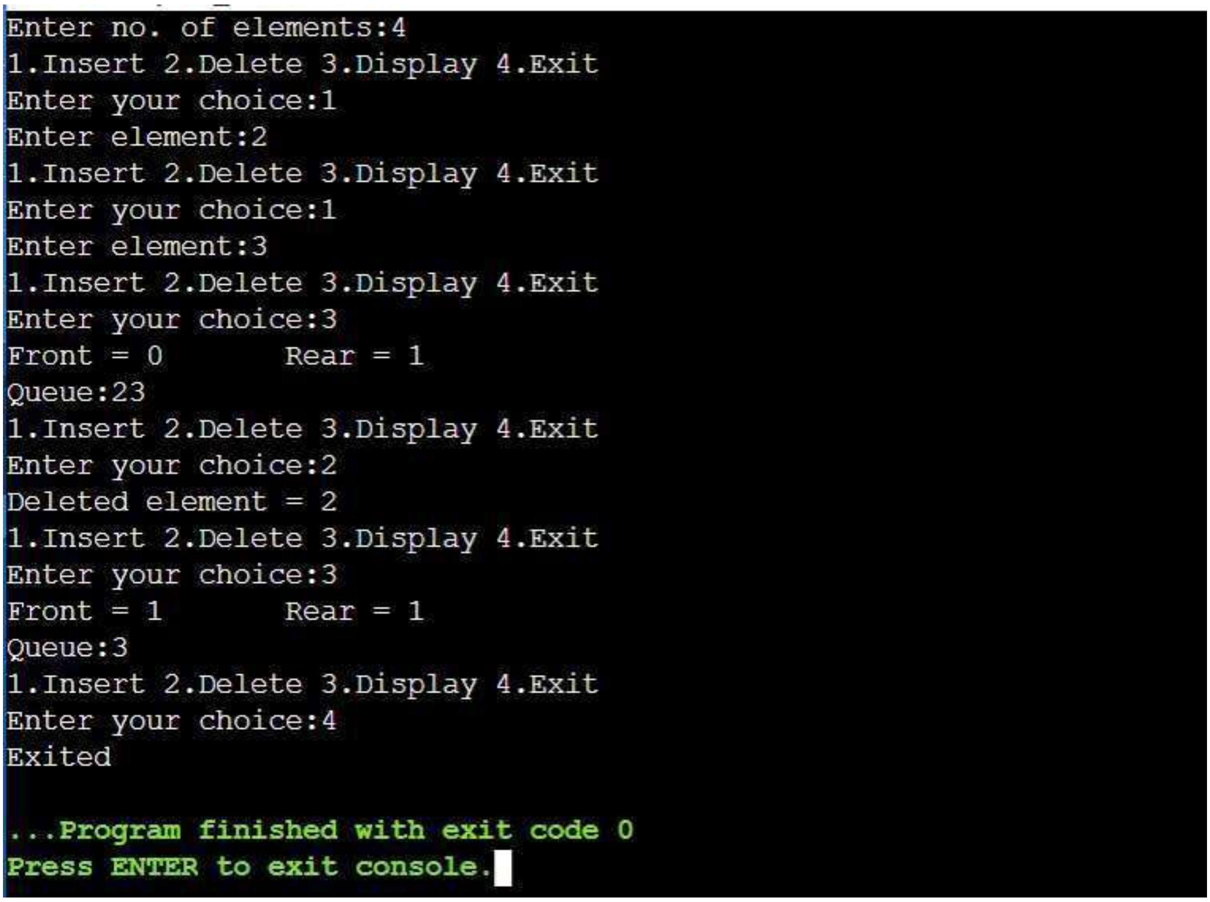
3b) WAP to simulate the working of a circular queue of integers using an array. Provide the following operations: Insert, Delete &amp; Display The program should print appropriate messages for queue empty and queue overflow conditions

```
#include <stdio.h> int
q[50],front=-1,rear=-1,size;
void display(); void enqueue();
void dequeue();
void main()
{
int ch;
printf("Enter no. of elements:");
scanf("%d",&size);
while(ch!=4)
{
printf("1.Insert 2.Delete 3.Display 4.Exit\n"); printf("Enter
your choice:");
scanf("%d",&ch);
switch (ch)
{
case 1:
enqueue();
break; case
2:
dequeue();
break; case
3:
```

```
display();
break;
}
}
printf("Exited");
}
void enqueue()
{
int item;
if((front == rear+1)||(front==0 && rear==size-1))
printf("ueue is full\n"); else
{
if(front == -1) front=0;
printf("Enter element:");
scanf("%d",&item);
rear=(rear + 1)%size;
q[rear] = item;
}
}
void dequeue()
{ int ele; if(front==-1)
printf("Queue is empty\n");
else
{
ele = q[front];
if(front==rear)
{
front = -1; rear
= -1;
}
else front = (front+1) %
size;
printf("Deleted element = %d\n",ele);
}
}
void display()
{
int i; if(front == -1)
printf("Queue is empty");
else
{
printf("Front = %d\t",front);
printf("Rear = %d\n",rear);
printf("Queue:");
```

```
for(i=front;i!=rear;i=(i+1)%size)
printf("%d",q[i]);
printf("%d\n",q[i]);
}
```

Output:



```
Enter no. of elements:4
1.Insert 2.Delete 3.Display 4.Exit
Enter your choice:1
Enter element:2
1.Insert 2.Delete 3.Display 4.Exit
Enter your choice:1
Enter element:3
1.Insert 2.Delete 3.Display 4.Exit
Enter your choice:3
Front = 0       Rear = 1
Queue:23
1.Insert 2.Delete 3.Display 4.Exit
Enter your choice:2
Deleted element = 2
1.Insert 2.Delete 3.Display 4.Exit
Enter your choice:3
Front = 1       Rear = 1
Queue:3
1.Insert 2.Delete 3.Display 4.Exit
Enter your choice:4
Exited

...Program finished with exit code 0
Press ENTER to exit console.
```

Lab program 4:

WAP to Implement Singly Linked List with following operations

a) Create a linked list.

b) Insertion of a node at first position, at any position and at end of list.

Display the contents of the linked list.

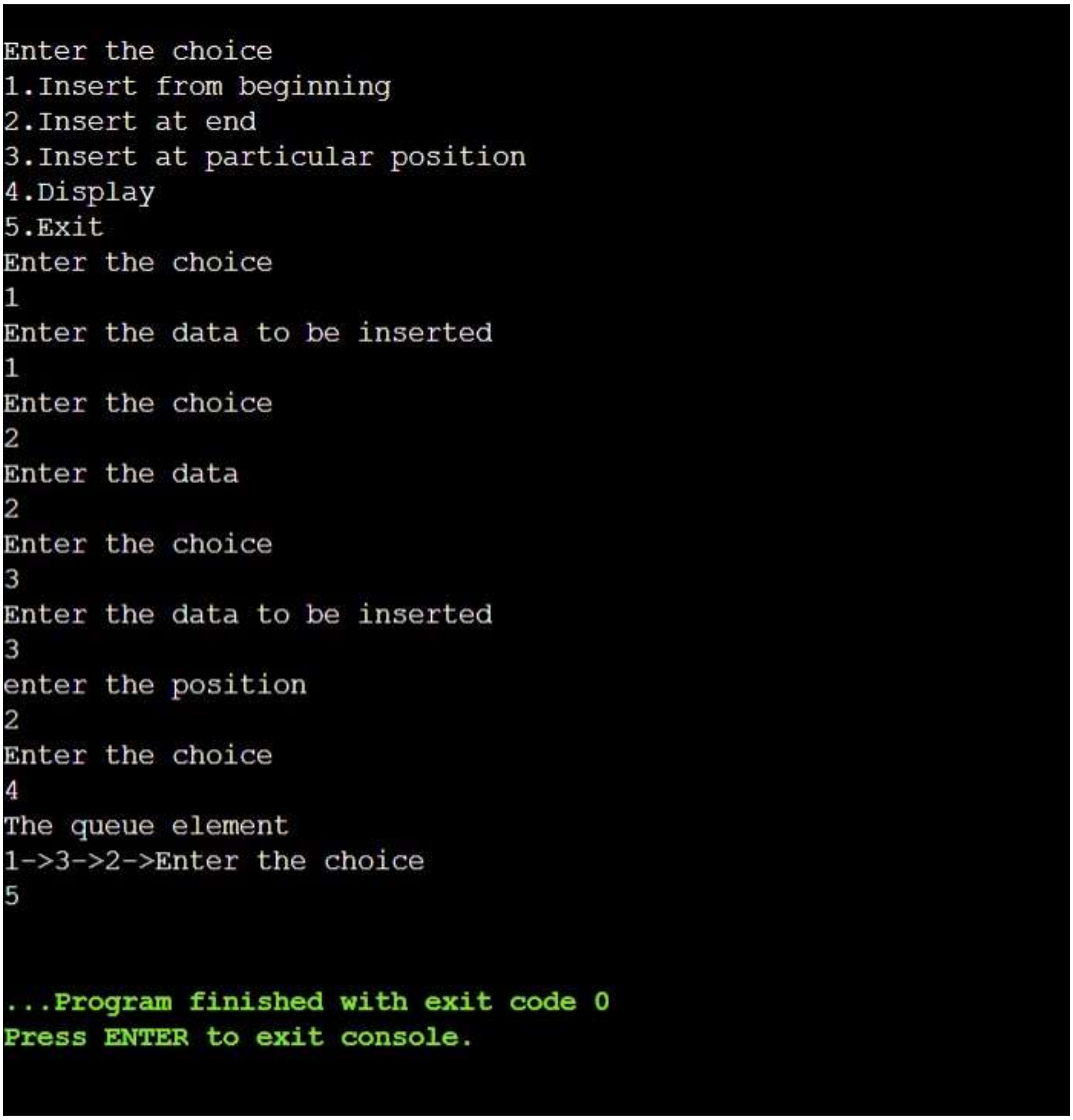
```
#include <stdio.h>
#include <stdlib.h> void
push();
```

```
void append(); void
display(); void
insert_at_pos();
struct node
{
int data; struct
node *next;
};
struct node *head=NULL;
void main()
{
int ch;
printf("\nEnter the choice\n1.Insert from beginning\n2.Insert at end\n3.Insert at particular
position\n4.Display\n5.Exit\n"); while(ch!=6)
{
printf("Enter the choice\n");
scanf("%d",&ch);
switch(ch)
{
case 1: push();
break; case 2:
append();
break; case 3:
insert_at_pos();
break; case 4:
display(); break;
case 5:exit(0);
default: printf("Invalid choise");
break;
}
}
}
void push()
{
int data; struct node
*new_node;
new_node=(struct node*)malloc(sizeof(struct node));
printf("Enter the data to be inserted\n");
scanf("%d",&data); new_node-
>data=data; new_node-
>next=head;
head=new_node;
}
void append()
{
int data; struct node
*last=head; struct node
*new_node;
```

```
new_node=(struct node*)malloc(sizeof(struct node));
printf("Enter the data\n");
scanf("%d",&data); new_node-
>data=data; new_node-
>next=NULL;
if(head==NULL)
{
head=new_node;
return;
}
while(last->next!=NULL)
{
last=last->next;
}
last->next=new_node;
}
void insert_at_pos()
{ int
data,i;
int pos; struct node *temp=head; struct node
*new_node; struct node *temp1; new_node = (struct
node*) malloc(sizeof(struct node)); printf("Enter the
data to be inserted\n");
scanf("%d",&data); new_node-
>data=data; printf("enter the
position\n"); scanf("%d",&pos);
if(pos==1)
{
new_node->next=temp;
head=new_node;
}
else
{
for(i=2;i<pos-1;i++)
{
temp=temp->next;
}
Temp1=temp->next; new_node-
>next=temp1;
temp->next=new_node;
}
}
void display()
{
struct node *p=head ;
printf("The queue element\n");
while(p!=NULL)
{
```

```
printf("%d->",p->data); p=p->next;
}
}
```

Output:



```
Enter the choice
1.Insert from beginning
2.Insert at end
3.Insert at particular position
4.Display
5.Exit
Enter the choice
1
Enter the data to be inserted
1
Enter the choice
2
Enter the data
2
Enter the choice
3
Enter the data to be inserted
3
enter the position
2
Enter the choice
4
The queue element
1->3->2->Enter the choice
5


...Program finished with exit code 0
Press ENTER to exit console.
```

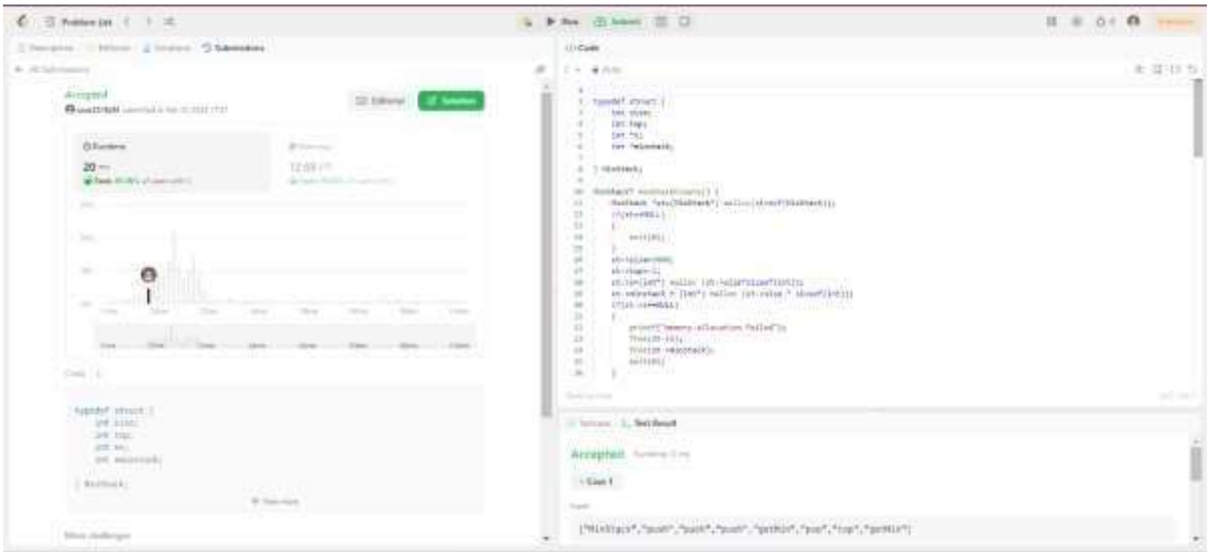
Program – Leetcode-1 platform(Minstack)

```
typedef struct {
   int size;     int
top;     int *s;
int *minstack;


} MinStack;


MinStack* minStackCreate() {
   MinStack *st=(MinStack*) malloc(sizeof(MinStack));
if(st==NULL)
   {
exit(0);
   }
   st->size=1000;     st->top=-1;     st->s=(int*) malloc
(st->size*sizeof(int));     st->minstack = (int*) malloc
(st->size * sizeof(int));     if(st->s==NULL)
   {
      printf("memory allocation failed");
free(st->s);         free(st->minstack);
exit(0);
   }
return st;
}
```

```
void minStackPush(MinStack* obj, int val) {     if(obj-

>top==obj->size-1)         printf("stack is overflow");     else{

obj->top++;         obj->s[obj->top]=val;         if (obj->top ==

0 || val < obj->minstack[obj->top - 1]) {             obj-

>minstack[obj->top] = val;

      } else {             obj->minstack[obj->top] = obj-

>minstack[obj->top - 1];

      }

   }

}


void minStackPop(MinStack* obj) {

   int value;     if(obj-

>top==-1)

printf("underflow");     else

   {

      value=obj->s[obj->top];
      obj->top--;

   }

}


int minStackTop(MinStack* obj) {

   int value=-1;     if(obj-

>top==-1)
```

```
   {
exit(0);    }
else
   {
        value=obj->s[obj->top];
return value;
   }
}


int minStackGetMin(MinStack* obj) {
if(obj->top==-1)
   {
exit(0);    }
else
   {
        return obj->minstack[obj->top];
   }
}


void minStackFree(MinStack* obj) {
   free(obj->s);    free(obj-
>minstack);    free(obj);
}
```

Output:



Accepted

Runtime

20 ms

```
typedef struct {
    int size;
    int top;
    int *s;
    int *minstack;

} MinStack;

MinStack* minStackCreate() {
    MinStack *st=(MinStack*) malloc(sizeof(MinStack));
    if(st==NULL)
    {
        exit(0);
    }
    st->size=1000;
    st->top=-1;
    st->s=(int*) malloc (st->size*sizeof(int));
    st->minstack = (int*) malloc (st->size * sizeof(int));
    if(st->s==NULL)
    {
        printf("memory allocation failed");
        free(st->s);
        free(st->minstack);
        exit(0);
    }
```

Accepted Runtime: 0 ms

Case 1

```
["MinStack","push","push","push","getMin","pop","top","getMin"]
```

Lab program 5:

WAP to Implement Singly Linked List with following operations

a) Create a linked list.

b) Deletion of first element, specified element and last element in the list.

c) Display the contents of the linked list.

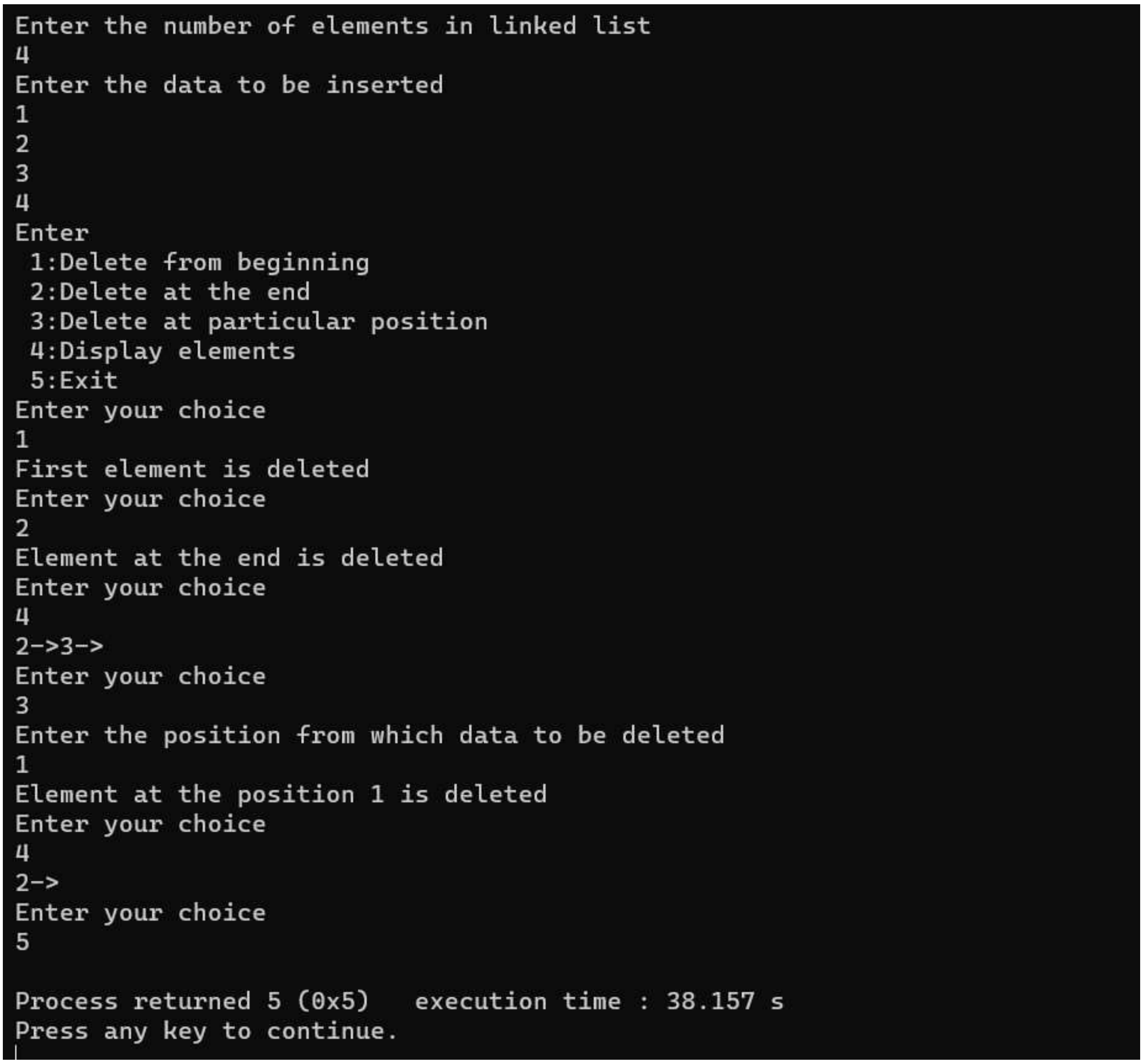
```
#include <stdio.h>

#include <stdlib.h> struct

node

{ int data; struct

node *next;

};
```

```
struct node *head=NULL;
void dbegin() { struct
node*ptr;
if(head==NULL)
{ printf("List is
empty\n"); } else {
ptr=head; head=head-
>next;
free(ptr); printf("First element is
deleted\n");
} } void dend() {
struct node*ptr;
struct node*ptr1;
if(head==NULL)
{ printf("List is
empty\n");
} else if(head-
>next==NULL)
{ free(head); } else {
ptr=head; while(ptr-
>next!=NULL)
{ ptr1=ptr;
ptr=ptr->next;
} free(ptr);
ptr1->next=NULL;
printf("Element at the end is deleted\n");
} } void dpos() {
struct node*ptr;
struct node*ptr1;
int pos,i;
printf("Enter the
position from
which data to be
deleted\n");
scanf("%d",&po
s); ptr=head;
if(head==NULL
)
{ printf("List is
empty\n");
} else if(head-
>next==NULL)
{ free(head); }
for(i=0;i<pos;i++)
{ ptr1=ptr; ptr=ptr->next; } ptr1->next=ptr->next;
free(ptr); printf("Element at the position %d is
deleted\n",pos);
} void display() { struct
node *node=head;
if(head==NULL)
```

```
{
printf("List is empty\n");
} else
{
while(node!=NULL)
{ printf("%d->",node->data);
node=node->next;
} printf("\n");
} } void main() { int n,i,data; printf("Enter the number
of elements in linked list\n"); scanf("%d",&n);
printf("Enter the data to be inserted\n");
for(i=0;i<n;i++) { struct node *last=head; struct node
*new_node; new_node=(struct
node*)malloc(sizeof(struct node));
scanf("%d",&data); new_node->data=data;
new_node->next=NULL; if(head==NULL)
{
head=new_node;
} else
{ while(last->next!=NULL)
{ last=last->next;
} last->next=new_node;
} } int ch; printf("Enter\n 1:Delete from beginning\n 2:Delete at the end\n 3:Delete
at particular position\n 4:Display elements\n 5:Exit\n"); while(ch!=5)
```

```
{ printf("Enter your
choice\n"); scanf("%d",&ch);
switch(ch) { case 1:dbegin();
break;
case 2:dend();
break; case
3:dpos(); break;
case 4:display();
break;
}
}
}
```

Ouput:



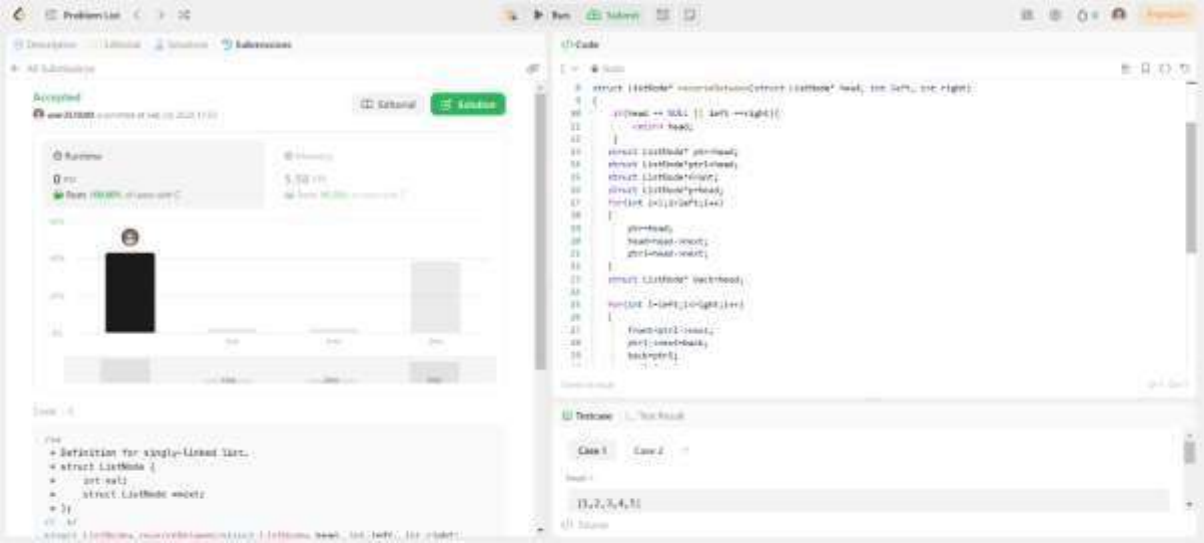
```
Enter the number of elements in linked list
4
Enter the data to be inserted
1
2
3
4
Enter
 1:Delete from beginning
 2:Delete at the end
 3:Delete at particular position
 4:Display elements
 5:Exit
Enter your choice
1
First element is deleted
Enter your choice
2
Element at the end is deleted
Enter your choice
4
2->3->
Enter your choice
3
Enter the position from which data to be deleted
1
Element at the position 1 is deleted
Enter your choice
4
2->
Enter your choice
5

Process returned 5 (0x5)   execution time : 38.157 s
Press any key to continue.
```

Program – Leetcode-2 platform(Reversing List)

```
/**
*     Definition for singly-linked list.
*     struct ListNode {
*     int val;
*     struct ListNode *next;
*     }; //  */ struct ListNode* reverseBetween(struct ListNode* head, int
left, int right)
{     if(head == NULL || left
==right){         return head;
    }    struct ListNode*
ptr=head;    struct
ListNode*ptr1=head;    struct
ListNode*front;    struct
ListNode*p=head;    for(int
i=1;i<left;i++)
```

```
   {        ptr=head;
head=head->next;
ptr1=head->next;
   }    struct ListNode*
back=head;
    for(int
i=left;i<right;i++)
   {        front=ptr1-
>next;        ptr1-
>next=back;
back=ptr1;
ptr1=front;
   }
if(left==1)    {
       p->next=ptr1->next;
ptr1->next=back;
p=ptr1;    }    else    {
ptr->next=back;
head->next=front;
   }
return p;
}
```

Output:



Accepted

Runtime

0 ms

Memory

5.58 MB

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
struct ListNode* reverseBetween(struct ListNode* head, int left, int right)
```

```
struct ListNode* reverseBetween(struct ListNode* head, int left, int right)
{
    if(head == NULL || left ==right){
        return head;
    }
    struct ListNode* ptr=head;
    struct ListNode*ptr1=head;
    struct ListNode*front;
    struct ListNode*p=head;
    for(int i=1;i<left;i++)
    {
        ptr=head;
        head=head->next;
        ptr1=head->next;
    }
    struct ListNode* back=head;

    for(int i=left;i<right;i++)
    {
        front=ptr1->next;
        ptr1->next=back;
        back=ptr1;
```

head =

[1,2,3,4,5]

Lab program 6:

6a) WAP to Implement Single Link List with following operations: Sort the linked list, Reverse the linked list, Concatenation of two linked lists.

#include <stdio.h>

#include <stdlib.h>

```
struct node
{     int data;
struct node *next;
};

struct node *head1=NULL; struct
node *head2=NULL;
void linklist1()
{     int
n,i,data;
printf("Enter
the number
of elements
in linked
list\n");
scanf("%d",
&n);
printf("Enter
the elements
to be inserted
for first
linked
list\n");
```

```
for(i=0;i<n;i

++)

   {

   struct node *last1=head1;     struct node *new_node1;

new_node1=(struct node*)malloc(sizeof(struct node));

scanf("%d",&data);     new_node1->data=data;

new_node1->next=NULL;     if(head1==NULL)

   {

      head1=new_node1;

   }

else

   {

   while(last1->next!=NULL)

   {        last1=last1-

>next;

   }

   last1->next=new_node1;

   }

   }

}
void linklist2()

{     int n,i,data;     printf("Enter the number of elements in linked

list\n");     scanf("%d",&n);     printf("Enter the elements to be

inserted for second linked list\n");     for(i=0;i<n;i++)

   {
```

```
   struct node *last2=head2;     struct node *new_node2;
new_node2=(struct node*)malloc(sizeof(struct node));
scanf("%d",&data);     new_node2->data=data;
new_node2->next=NULL;     if(head2==NULL)
   {
      head2=new_node2;
   }
else
   {
   while(last2->next!=NULL)
   {        last2=last2-
>next;
   }
   last2->next=new_node2;
   }
   }
} void sort() {     struct node
*curr = head1;     struct node
*ptr = NULL;     int temp;
while(curr != NULL)
   {        ptr = curr-
>next;         while(ptr
!=NULL)
      {
         if(curr->data > ptr->data)
```

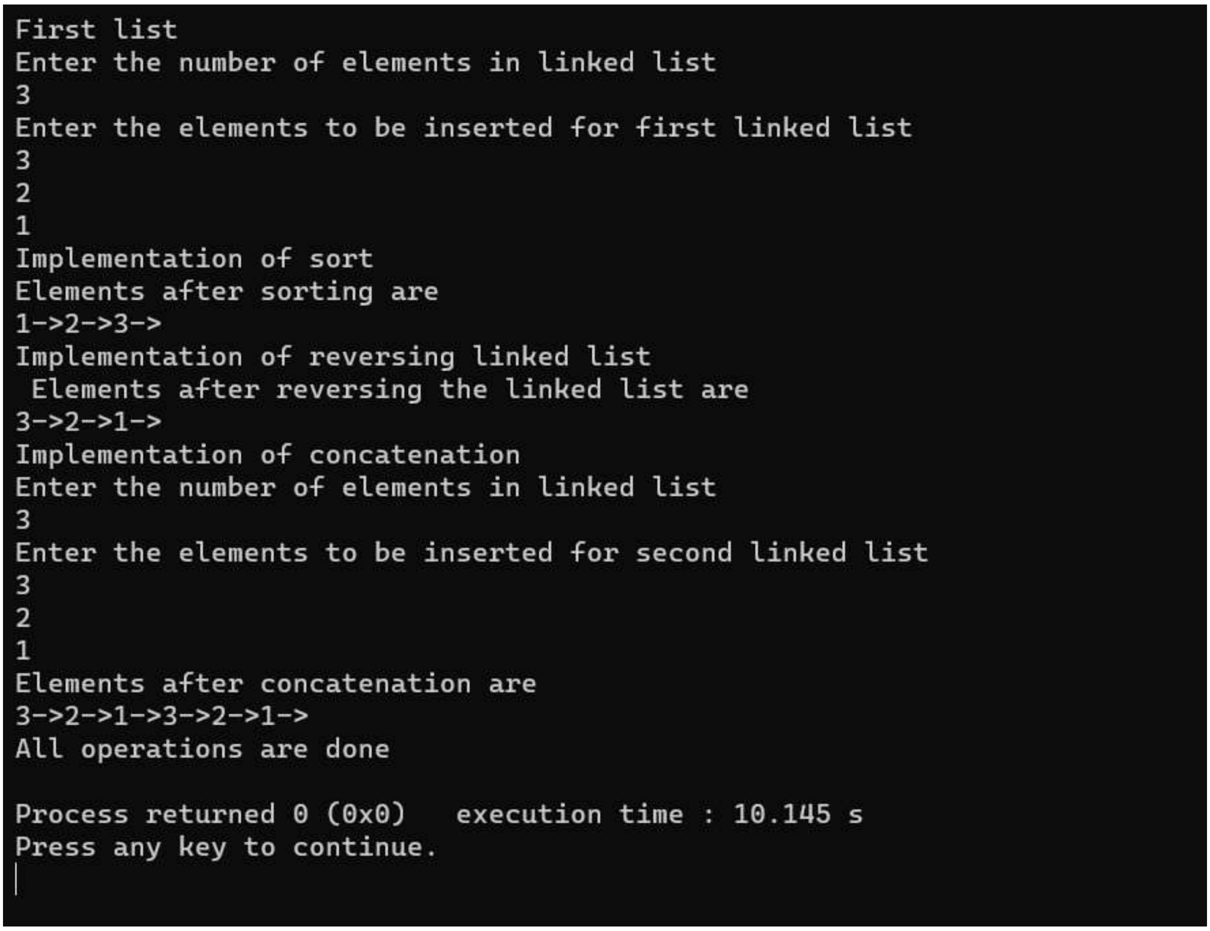
```
            {                  temp = curr-
>data;                  curr-
>data=ptr->data;                  ptr-
>data=temp;
            }


            ptr = ptr->next;
        }
        curr = curr->next;
    }
display(); }
void
reverse() {
struct node*
prev=NULL;
struct node*
ptr=NULL;
while(head1!
=NULL)
    {
        ptr=head1->next;
head1->next=prev;
prev=head1;         head1=ptr;
    }
```

```
   head1=prev;     display();

} void concate() {     struct

node*temp=head1;

if(head1==NULL)

   {

      struct node *node=head2;

while(node!=NULL)

      {

         printf("%d->",node->data);
node=node->next;

      }

printf("\n");     }

else

   {

      while(temp->next!=NULL)

      {

         temp=temp->next;

      }

      temp->next=head2;

   }

   struct node *node=head1;

while(node!=NULL)

      {

         printf("%d->",node->data);

node=node->next;
```

```
    }       printf("\n"); }
void display() {     struct
node *node=head1;
if(head1==NULL)
  {
    printf("List is empty\n");
  }
else
  {
    while(node!=NULL)
    {
      printf("%d->",node->data);
node=node->next;
    }
printf("\n");
  }
} void main() {     printf("First list\n");     linklist1();
printf("Implementation of sort\n");     printf("Elements
after sorting are\n");     sort();     printf("Implementation
of reversing linked list\n ");     printf("Elements after
reversing the linked list are\n");     reverse();
printf("Implementation of concatenation\n");
  linklist2();     printf("Elements after
concatenation are\n"); concate(); printf("All
operations are done\n");
```

```
}
```

Output:



```
First list
Enter the number of elements in linked list
3
Enter the elements to be inserted for first linked list
3
2
1
Implementation of sort
Elements after sorting are
1->2->3->
Implementation of reversing linked list
 Elements after reversing the linked list are
3->2->1->
Implementation of concatenation
Enter the number of elements in linked list
3
Enter the elements to be inserted for second linked list
3
2
1
Elements after concatenation are
3->2->1->3->2->1->
All operations are done

Process returned 0 (0x0)   execution time : 10.145 s
Press any key to continue.
```

6b) WAP to Implement Single Link List to simulate Stack &amp; Queue Operations.

Queue using LL:

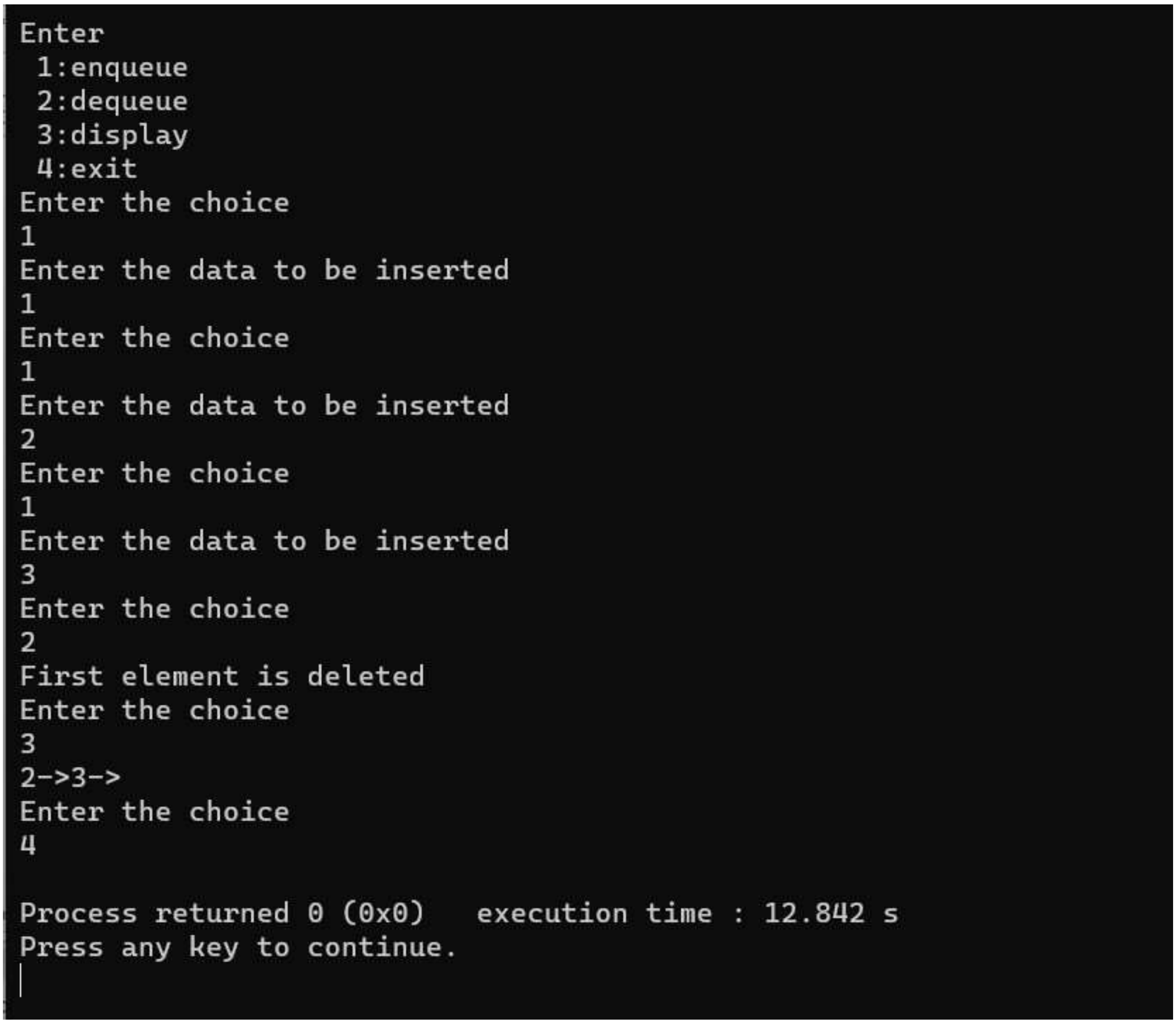
```
#include <stdio.h>

#include <stdlib.h>



struct node

{
   int data; struct

   node *next;

};
```

```
struct node *head=NULL; void enqueue() {     int
i,data;     printf("Enter the data to be inserted\n");
struct node *last=head;     struct node *new_node;
new_node=(struct node*)malloc(sizeof(struct node));
scanf("%d",&data);     new_node->data=data;
new_node->next=NULL;     if(head==NULL)
   {
      head=new_node;
   }
else
   {
   while(last->next!=NULL)
   {        last=last-
>next;
   }
   last->next=new_node;
   } } void
dequeue() {
struct node*ptr;
if(head==NULL)
   {        printf("List is
empty\n");
   }     else     {
ptr=head;
head=head->next;
```

```
        free(ptr);         printf("First element
is deleted\n");
    }
} void display() {     struct
node *node=head;
if(head==NULL)
    {         printf("List is
empty\n");
    }
    else
    {
while(node!=NULL)
        {
            printf("%d->",node->data);
node=node->next;
        }
printf("\n");
    }
}

void main() {     int ch;     printf("Enter\n 1:enqueue \n 2:dequeue
\n 3:display \n 4:exit\n");     while(ch!=4)
    {
```

```
   printf("Enter the choice\n");
scanf("%d",&ch);
switch(ch)
   {        case
1:enqueue();
break;        case
2:dequeue();
break;
      case 3:display();
break;        case
4:exit(0);
   }
   }
}
```

Output:



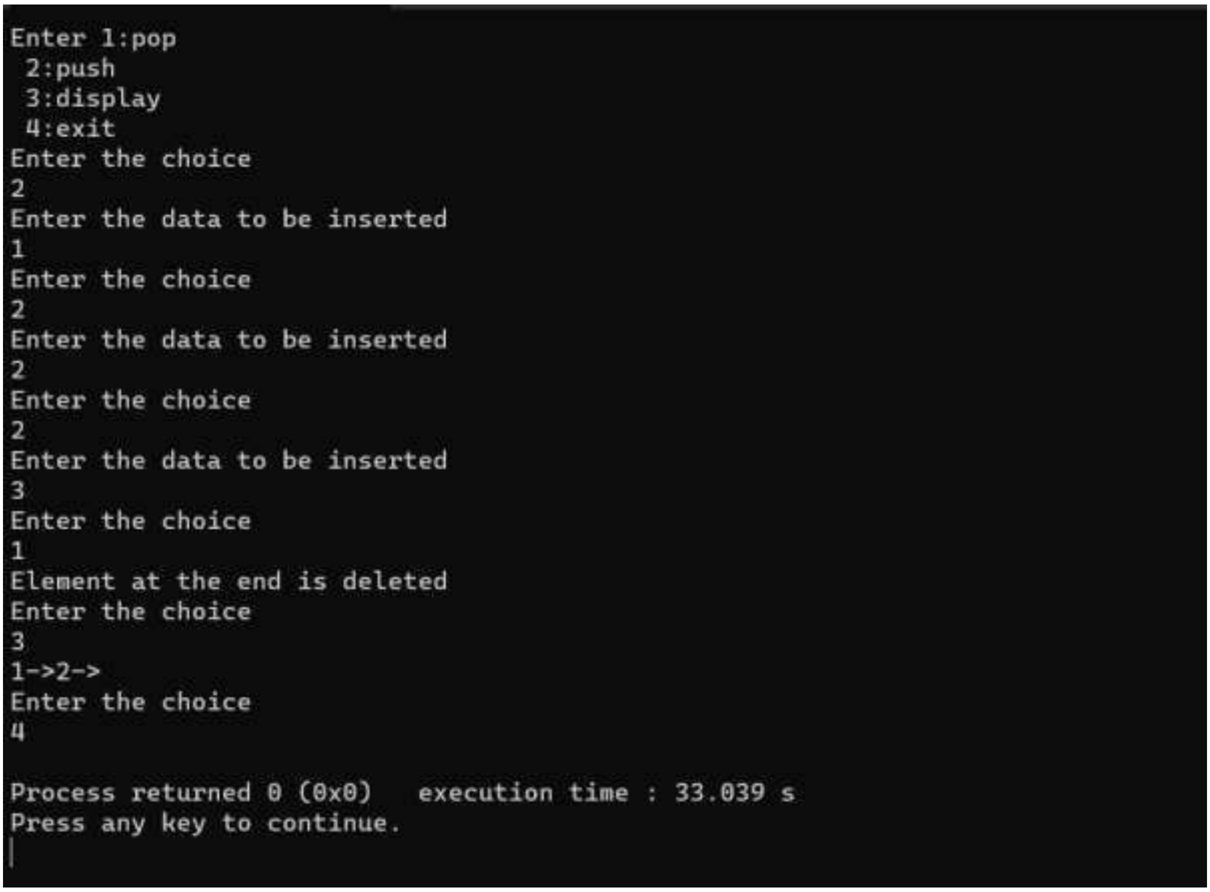
```
Enter
 1:enqueue
 2:dequeue
 3:display
 4:exit
Enter the choice
1
Enter the data to be inserted
1
Enter the choice
1
Enter the data to be inserted
2
Enter the choice
1
Enter the data to be inserted
3
Enter the choice
2
First element is deleted
Enter the choice
3
2->3->
Enter the choice
4

Process returned 0 (0x0)   execution time : 12.842 s
Press any key to continue.
```

Stack using LL:

```
#include <stdio.h>

#include <stdlib.h>



struct node

{    int data;    struct

node *next;

};
```

```
struct node *head=NULL; void

push()

{    int i,data;    printf("Enter the data to be

inserted\n");    struct node *last=head;    struct node

*new_node;    new_node=(struct

node*)malloc(sizeof(struct node));

scanf("%d",&data);    new_node->data=data;

new_node->next=NULL;    if(head==NULL)

  {

     head=new_node;
  }

  else

  {

  while(last->next!=NULL)

  {

     last=last->next;

  }

  last->next=new_node;

  }

}

void pop()

{
   struct node*ptr;
struct node*ptr1;
if(head==NULL)
   {
       printf("List is empty\n");
   }
   else if(head->next==NULL)
   {
       free(head);
   }
   else
   {
       ptr=head;        while(ptr-
>next!=NULL)
       {
           ptr1=ptr;
ptr=ptr->next;
       }
       free(ptr);        ptr1->next=NULL;
printf("Element at the end is deleted\n");
   }
}
void display()
{
   struct node *node=head;
if(head==NULL)
   {
      printf("List is empty\n");
   }
   else
   {
      while(node!=NULL)
      {
         printf("%d->",node->data);
node=node->next;
      }
      printf("\n");
   }
}

void main()
{    int ch;    printf("Enter 1:pop \n 2:push \n 3:display \n
4:exit\n");    while(ch!=4)
   {
   printf("Enter the choice\n");
scanf("%d",&ch);
switch(ch)
   {
        case 1: pop();

break;        case

2:push();        break;

case 3:display();

break;        case

4:exit(0);

   }

   }
}
```

Output:



```
Enter 1:pop
 2:push
 3:display
 4:exit
Enter the choice
2
Enter the data to be inserted
1
Enter the choice
2
Enter the data to be inserted
2
Enter the choice
2
Enter the data to be inserted
3
Enter the choice
1
Element at the end is deleted
Enter the choice
3
1->2->
Enter the choice
4

Process returned 0 (0x0)   execution time : 33.039 s
Press any key to continue.
```

Lab program 7:

WAP to Implement doubly link list with primitive operations

a) Create a doubly linked list.

b) Insert a new node to the left of the node.

c) Delete the node based on a specific value

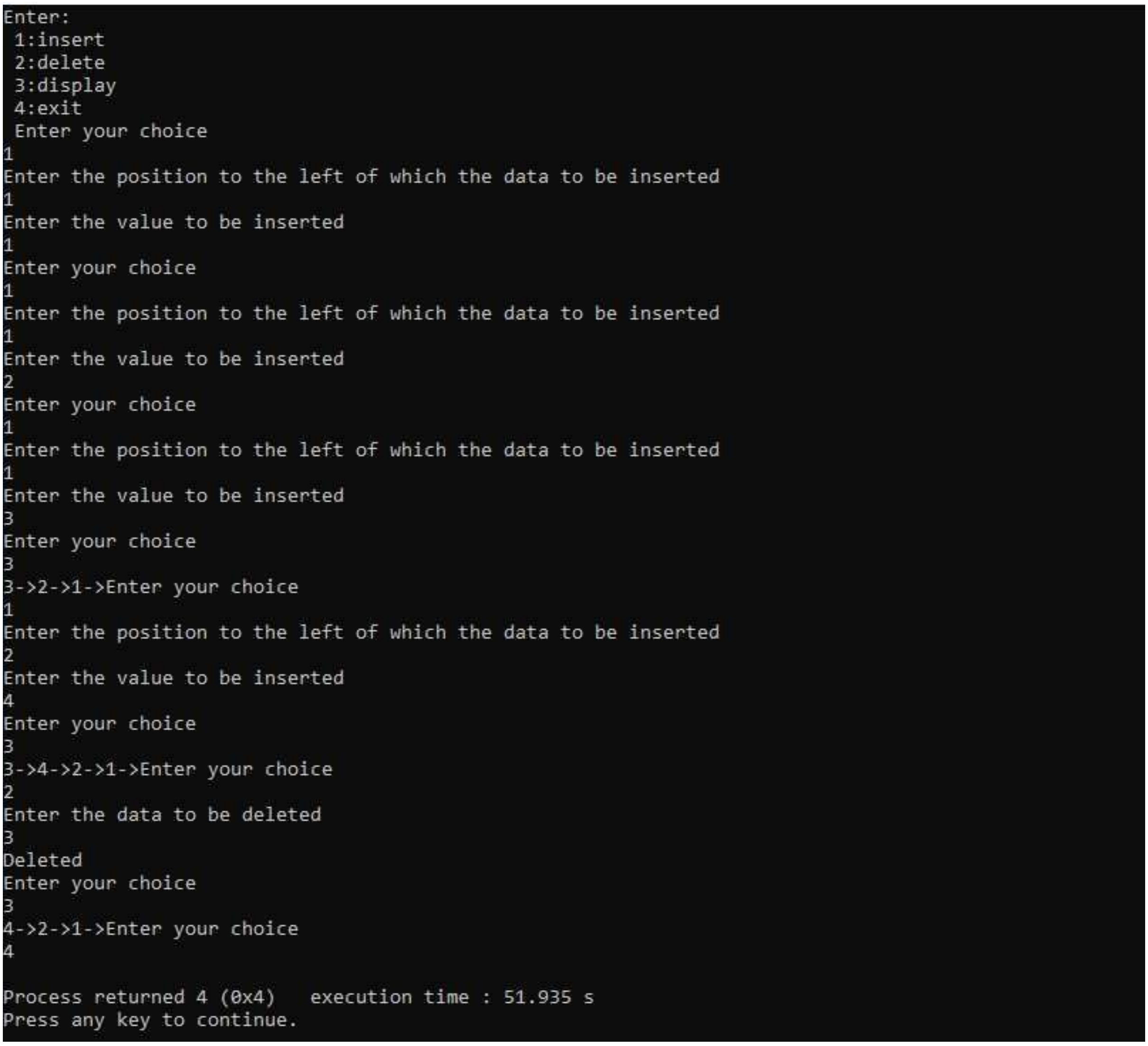
d) Display the contents of the list

```
#include <stdio.h>

#include <ctype.h> struct

node

{    int val;    struct

node * prev;    struct

node * next;

};

struct node *head=NULL;

void insert_left()

{   int pos;   struct node* ptr1=head;   printf("Enter the position to the

left of which the data to be inserted\n");   scanf("%d",&pos);   struct

node*new_node = (struct node*)malloc(sizeof(struct node));

printf("Enter the value to be inserted\n");   scanf("%d",&new_node-

>val);


  if(head==NULL)

  {

     head=new_node;       head-

>prev=NULL;       head->next=NULL;

  }

  else if(pos==1)

  {
      new_node->next=head;
      new_node->prev=NULL;       head-

>prev=new_node;       head=new_node;

  }

else

  {

  for(int i=0;i<pos-1;i++)

  {       ptr1=ptr1-

>next;

  }

  new_node->prev=ptr1->prev;    new_node-

>next=ptr1;    ptr1->prev->next=new_node;

ptr1->prev=new_node;

  } } void delete_pos() {    int data;

struct node* ptr=head;    printf("Enter

the data to be deleted\n");

scanf("%d",&data);    if(ptr->val==data)

  {

    head=head->next;
    free(ptr);

printf("Deleted\n");

return;    }    else

  {    while(ptr-

>val!=data)

  {       ptr=ptr->next;

if(ptr->next==NULL)
```

```
        {
            ptr->prev->next=NULL;
            free(ptr);
printf("deleted\n");            return;
        }
   }
   ptr->next->prev=ptr->prev;
ptr->prev->next=ptr->next;
head=ptr;    printf("Deleted");
  } } void
display() {
   struct node*p=head;
while(p!=NULL)
   {        printf("%d->",p-
>val);        p=p->next;
   } } void main() {     int ch;     printf("Enter:\n 1:insert\n
2:delete\n 3:display \n 4:exit\n ");     while(ch!=4)
    {         printf("Enter your
choice\n");         scanf("%d",&ch);
switch(ch)
        {             case
1:insert_left();
break;              case
2:delete_pos();
```

```
break;          case

3:display();          break;

      }

   }
}
```

Output:



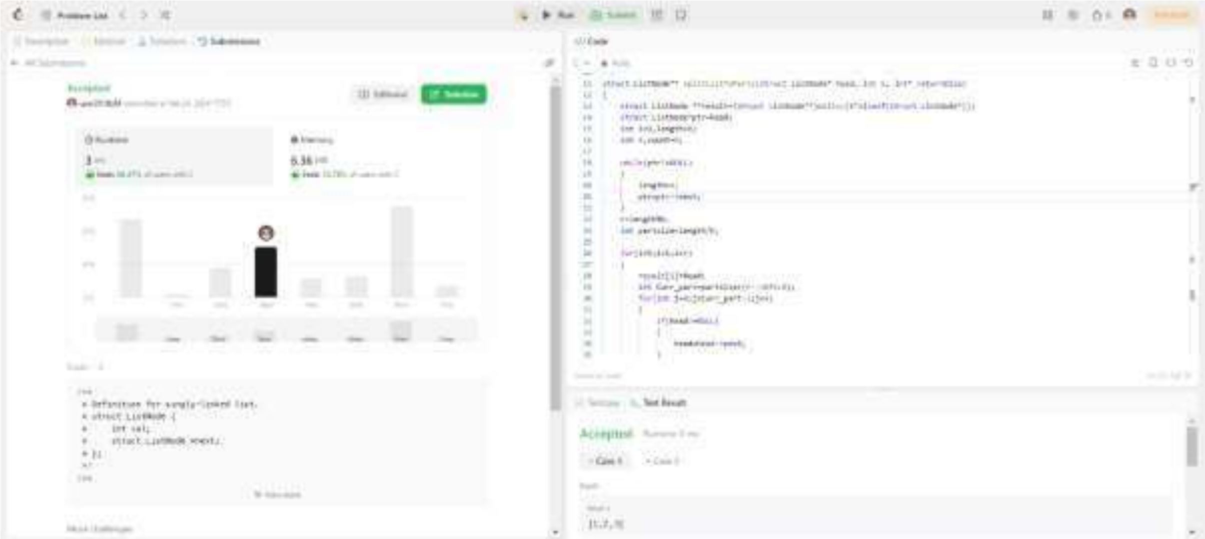
```
Enter:
 1:insert
 2:delete
 3:display
 4:exit
 Enter your choice
1
Enter the position to the left of which the data to be inserted
1
Enter the value to be inserted
1
Enter your choice
1
Enter the position to the left of which the data to be inserted
1
Enter the value to be inserted
2
Enter your choice
1
Enter the position to the left of which the data to be inserted
1
Enter the value to be inserted
3
Enter your choice
3
3->2->1->Enter your choice
1
Enter the position to the left of which the data to be inserted
2
Enter the value to be inserted
4
Enter your choice
3
3->4->2->1->Enter your choice
2
Enter the data to be deleted
3
Deleted
Enter your choice
3
4->2->1->Enter your choice
4

Process returned 4 (0x4)   execution time : 51.935 s
Press any key to continue.
```

Program – Leetcode-3 platform(Spliting Linked list)

```
**
*     Definition for singly-linked list.
*     struct ListNode {
*     int val;
*     struct ListNode *next;
*     };
```

```
 */
/**
*     Note: The returned array must be malloced, assume caller calls free().
 */ struct ListNode** splitListToParts(struct ListNode* head, int k, int*
returnSize)
{
    struct ListNode **result=(struct ListNode**)malloc(k*sizeof(struct
ListNode*));     struct ListNode*ptr=head;     int i=0,length=0;     int
r,count=0;

while(ptr!=NULL)
    {
length++;
ptr=ptr->next;
    }
r=length%k;
    int partsize=length/k;

for(i=0;i<k;i++)
    {         result[i]=head;         int
Curr_part=partsize+(r-->0?1:0);
for(int j=0;j<Curr_part-1;j++)
        {
if(head!=NULL)
            {
head=head->next;
            }
}
        if(head!=NULL)
        {             struct
ListNode*temp=head;
head=head->next;             temp-
>next=NULL;
        }
    }
    *returnSize=k;
return result;
```

Output:



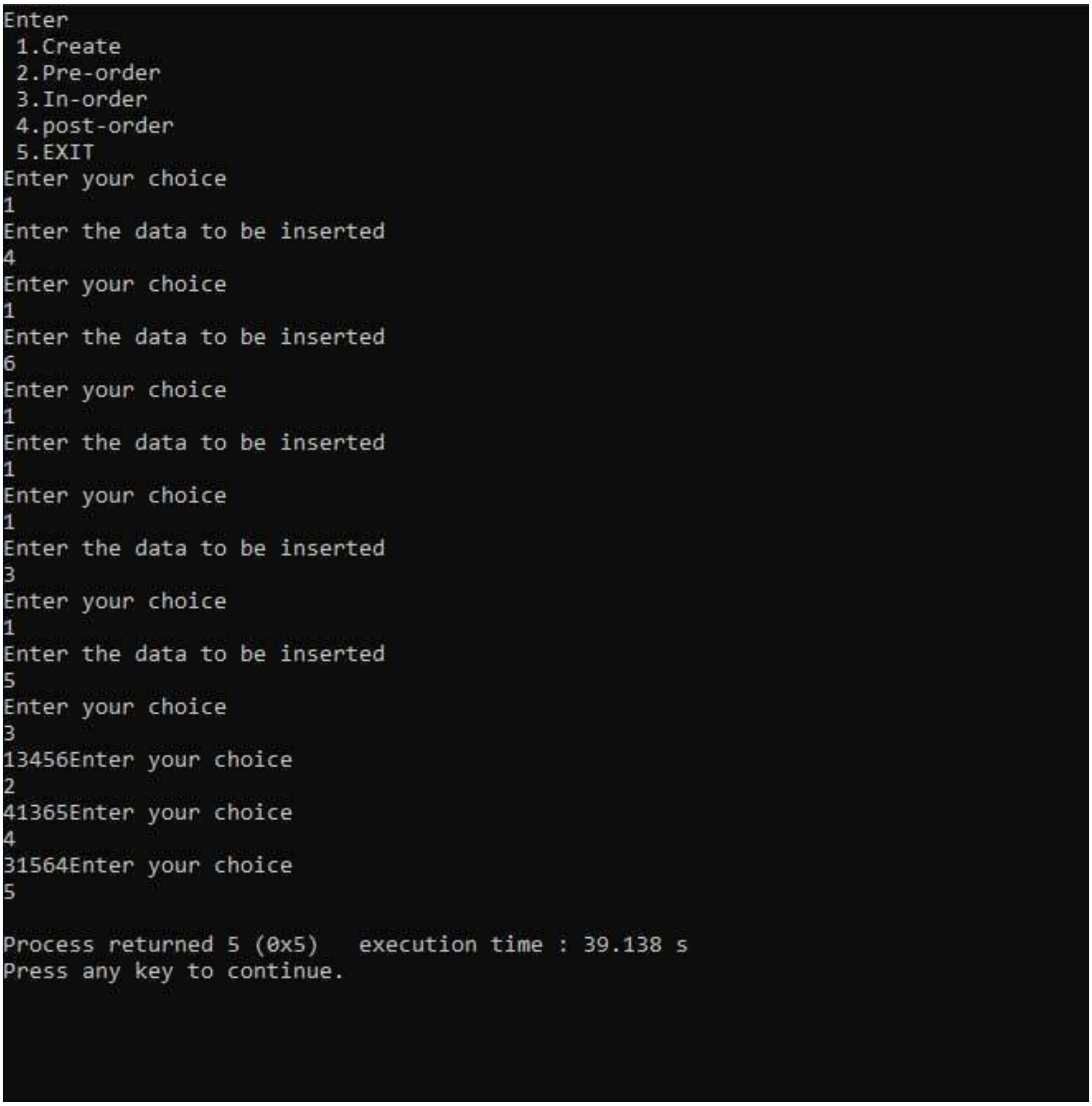
Lab program 8:

Write a program

a) To construct a binary Search tree.
b) To traverse the tree using all the methods i.e., in-order, preorder and post order
c) To display the elements in the tree.

```
#include <stdio.h>
#include <stdlib.h>

struct node
{     int data;
struct node* left;
struct node*right;
};
struct node*root=NULL;

void create()
{
   struct node* new_node=(struct node*)malloc(sizeof(struct node));
   printf("Enter the data to be inserted\n");
scanf("%d",&new_node->data);
   struct node*ptr=root;
struct node*ptr1;
new_node->left=NULL;
new_node->right=NULL;

   if(root==NULL)
   {
      root=new_node;
   }
else
   {
      while(ptr->left!=NULL || ptr->right!=NULL)
      {
         if(new_node->data > ptr->data)
         {
            if(ptr->right!=NULL)
            {
              ptr=ptr->right;
            }
else             {
break;
            }
}
else
         {
            if(ptr->left!=NULL)
            {
ptr=ptr->left;
            }
else             {
break;
            }
         }
      }
      if(new_node->data > ptr->data)
      {
        ptr->right=new_node;
      }
else
      {
         ptr->left=new_node;
      }
   }
}

void pre_order(struct node*ptr)
{    struct node*
trav=ptr;
   if(ptr!=NULL)
   {
        printf("%d",ptr->data);
pre_order(ptr->left);
        pre_order(ptr->right);
    }
}

void inorder(struct node*ptr)
{     struct node*
trav=ptr;
if(ptr!=NULL)
    {
        inorder(ptr->left);          printf("%d",ptr-
>data);          inorder(ptr->right);
    }
}

void post_order(struct node*ptr)
{     struct node*
trav=ptr;
    if(ptr!=NULL)
    {
        post_order(ptr->left);          post_order(ptr-
>right);
        printf("%d",ptr->data);
    }
}


void main()
{
    printf("Enter\n 1.Create\n 2.Pre-order\n 3.In-order\n 4.post-order\n 5.EXIT\n");
int ch;     do{
        printf("Enter your choice\n");
scanf("%d",&ch);
        switch(ch)
        {
            case 1:create();
            break;
            case 2:pre_order(root);
break;
            case 3:inorder(root);
break;
            case 4:post_order(root);
            break;
        }
    }while(ch!=5);
}
```

Output:



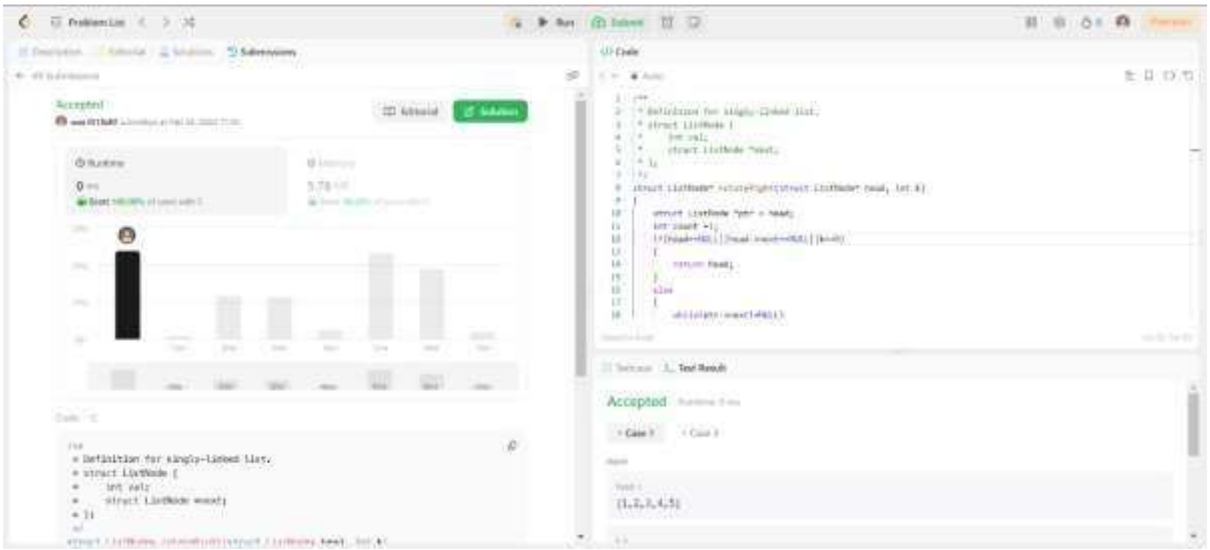
```
Enter
 1.Create
 2.Pre-order
 3.In-order
 4.post-order
 5.EXIT
Enter your choice
1
Enter the data to be inserted
4
Enter your choice
1
Enter the data to be inserted
6
Enter your choice
1
Enter the data to be inserted
1
Enter your choice
1
Enter the data to be inserted
3
Enter your choice
1
Enter the data to be inserted
5
Enter your choice
3
13456Enter your choice
2
41365Enter your choice
4
31564Enter your choice
5

Process returned 5 (0x5)   execution time : 39.138 s
Press any key to continue.
```

Program – Leetcode-4 platform(Rotating Linked list)

```
/**
*    Definition for singly-linked list.
*    struct ListNode {
*    int val;
*    struct ListNode *next;
*    };
```

```
 */ struct ListNode* rotateRight(struct ListNode* head,
int k)
{
    struct ListNode *ptr = head;     int count =1;
if(head==NULL||head->next==NULL||k==0)
    {
        return head;
    }
else
    {
        while(ptr->next!=NULL)
        {
            ptr=ptr->next;
count++;
        }
        if(k%count==0)
        {
            return head;
        }
        ptr->next=head;
        for(int i=k%count;i<count;i++)
        {
            ptr=ptr->next;
        }
        head=ptr->next;
ptr->next=NULL;         return
head;

}
}
```

Output:



Accepted

Runtime
0 ms

Memory
5.78 MB

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
struct ListNode* rotateRight(struct ListNode* head, int k)
{
    struct ListNode *ptr = head;
    int count =1;
    if(head==NULL||head->next==NULL||k==0)
    {
        return head;
    }
    else
    {
        while(ptr->next!=NULL)
```

Test Result

Accepted

Case 1 Case 2

[1,2,3,4,5]

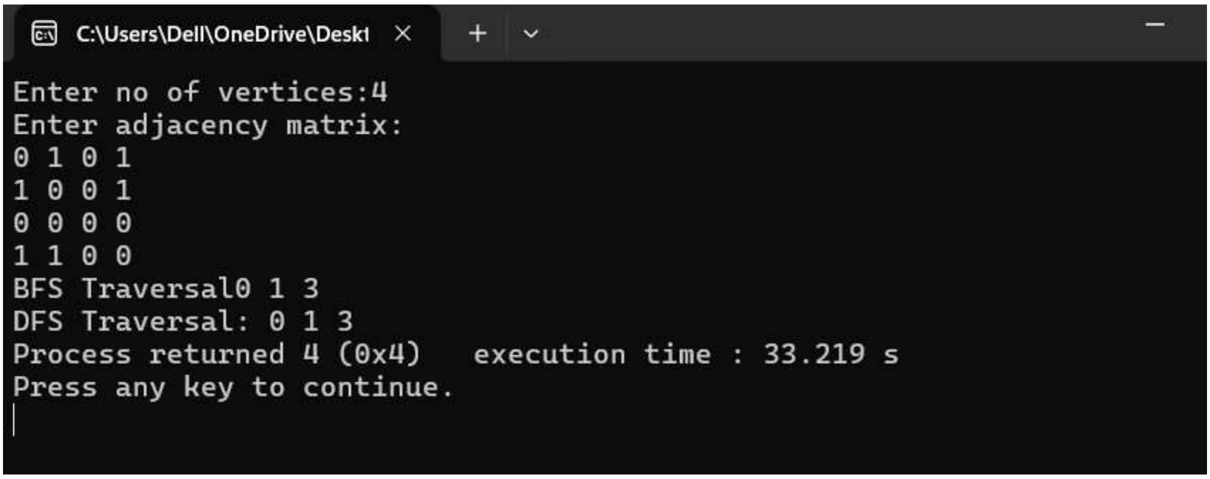
Lab program 9:

9a) Write a program to traverse a graph using BFS method.

```
#include<stdio.h>

int queue[100]; int
front=0,rear=0;
//int n=6;

int a[10][10]; void
enqueue(int var)
{
   queue[rear] = var;
rear++; }
void dequeue()
{
   front++;
}

void bfs(int n)
{
   int visited[10] = {0};
enqueue(0);     visited[0]
= 1;     while(front !=
rear)
   {
      int current = queue[front];
      printf("%d ", current);
dequeue();        for(int
i=0;i<n;i++)
      {
         if((a[current][i] == 1) && (visited[i] == 0))
         {
visited[i] = 1;
enqueue(i);
         }
      }
   }
}


void dfs(int a[][10], int n, int start , int visited2[10]) {

  visited2[start] = 1;
  printf("%d ", start);
```

```
   for(int i=0; i<n; i++) {
if(a[start][i] && !visited2[i]) {
dfs(a,n,i,visited2);
      }
   } } void main() {     int n;     int
start=0;     int visited2[10];
printf("Enter no of vertices:");
scanf("%d",&n);     printf("Enter
adjacency matrix:\n");     for(int
i=0;i<n;i++)
    {        for(int
j=0;j<n;j++)
            scanf("%d",&a[i][j]);
    }     bfs(n);    for(int i = 0;
i < 10; ++i) {       visited2[i]
= 0;
   }

   printf("\nDFS Traversal: ");
   dfs(a, n, start,visited2);

    //dfs(a,n,start);
}
```

Output:



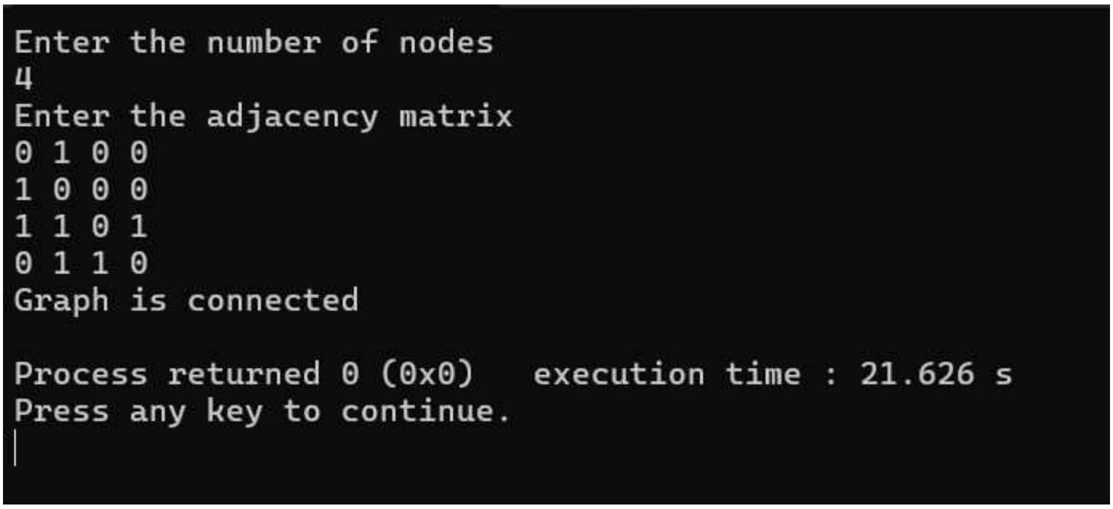
```
Enter no of vertices:4
Enter adjacency matrix:
0 1 0 1
1 0 0 1
0 0 0 0
1 1 0 0
BFS Traversal0 1 3
DFS Traversal: 0 1 3
Process returned 4 (0x4)   execution time : 33.219 s
Press any key to continue.
```

9b) Write a program to check whether given graph is connected or not using DFS method.

```
#include<stdio.h> int
a[1][10];
void dfs(int n, int cost[10][10], int u, int s[])
```

```
{     int v;
s[u]=1;
   for(v=0;v<n;v++)
   {
       if((cost[u][v]==1) && (s[v]==0))
       dfs(n,cost,v,s);
   }
}
void main()
{     int n,i,j,cost[10][10],s[10],con,flag;
printf("Enter the number of nodes\n");
scanf("%d", &n);
   printf("Enter the adjacency matrix\n");
for(i=0;i<n;i++)
   {
       for(j=0;j<n;j++)
scanf("%d", &cost[i][j]);
   }
con=0;
   for(j=0;j<n;j++)
   {
       for(i=0;i<n;i++)
s[i]=0;         dfs(n,cost,j,s);
flag=0;
for(i=0;i<n;i++)
         {
             if(s[i]==0)
flag=1;
         }
if(flag==0)
con=1;
   }
   if(con==1)
       printf("Graph is connected\n");
else
       printf("Graph is not connected\n");

}
```

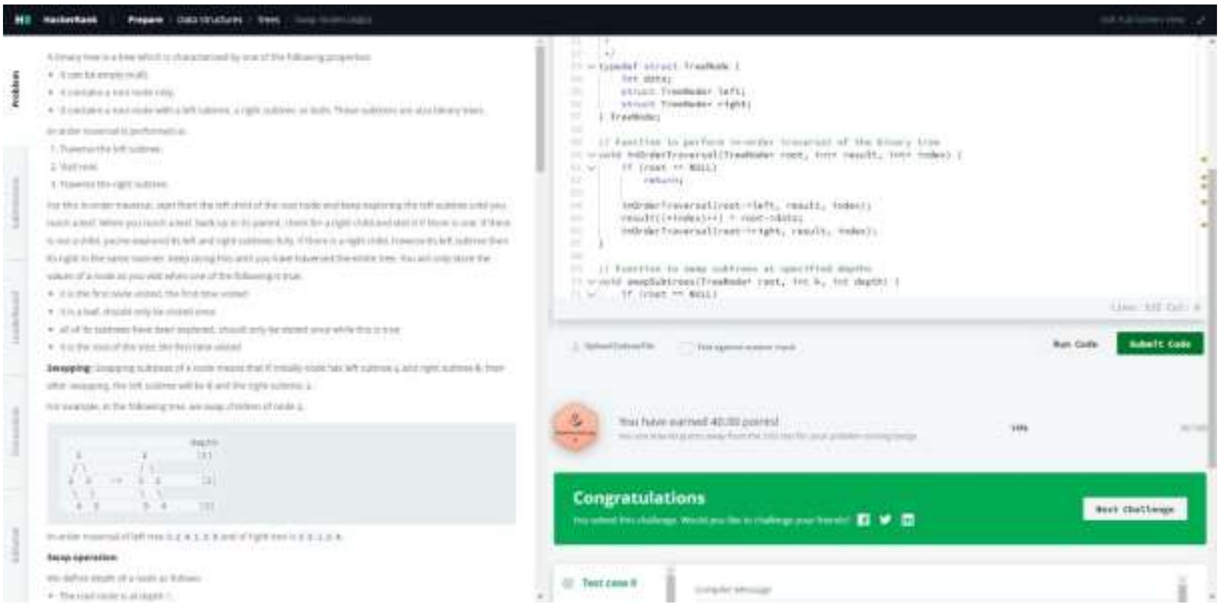
Output:



```
Enter the number of nodes
4
Enter the adjacency matrix
0 1 0 0
1 0 0 0
1 1 0 1
0 1 1 0
Graph is connected

Process returned 0 (0x0)   execution time : 21.626 s
Press any key to continue.
```

Hacker rank-1

```
void inOrderTraversal(TreeNode* root, int* result, int* index) {
if (root == NULL)
        return;

    inOrderTraversal(root->left, result, index);
result[(*index)++] = root->data;
    inOrderTraversal(root->right, result, index);
}

// Function to swap subtrees at specified depths void
swapSubtrees(TreeNode* root, int k, int depth) {     if
(root == NULL)
        return;

    if (depth % k == 0) {
        TreeNode* temp = root->left;         root-
>left = root->right;
        root->right = temp;
    }

    swapSubtrees(root->left, k, depth + 1);
swapSubtrees(root->right, k, depth + 1);
}

// Function to build the binary tree from the given indexes
TreeNode* buildTree(int indexes_rows, int indexes_columns, int** indexes) {
    TreeNode* root = (TreeNode*)malloc(sizeof(TreeNode));
root->data = 1;     root->left = NULL;
    root->right = NULL;
    TreeNode* nodes[indexes_rows + 1];
    nodes[1] = root;

    for (int i = 0; i < indexes_rows; i++) {
        TreeNode* curr = nodes[i + 1];

        if (indexes[i][0] != -1) {
            curr->left = (TreeNode*)malloc(sizeof(TreeNode));
            curr->left->data = indexes[i][0];
curr->left->left = NULL;             curr->left-
>right = NULL;
            nodes[indexes[i][0]] = curr->left;
        }

        if (indexes[i][1] != -1) {
            curr->right = (TreeNode*)malloc(sizeof(TreeNode));
            curr->right->data = indexes[i][1];
curr->right->left = NULL;             curr->right-
>right = NULL;
            nodes[indexes[i][1]] = curr->right;
        }
    }

    return root;
}

int** swapNodes(int indexes_rows, int indexes_columns, int** indexes, int queries_count,
int* queries, int* result_rows, int* result_columns) {
    int** result = (int**)malloc(queries_count * sizeof(int*));
    *result_rows = queries_count;
    *result_columns = indexes_rows;

    TreeNode* root = buildTree(indexes_rows, indexes_columns, indexes);

    for (int i = 0; i < queries_count; i++) {
        int k = queries[i];
        swapSubtrees(root, k, 1);

        int* traversal = (int*)malloc(indexes_rows * sizeof(int));
int index = 0;
        inOrderTraversal(root, traversal, &index);
        result[i] = traversal;
    }
    return result;
}
```



A binary tree is a tree which is characterized by one of the following properties:

- It can be empty (null).
- It contains a root node only.
- It contains a root node with a left subtree, a right subtree, or both. These subtrees are also binary trees.

In-order traversal is performed as

1. Traverse the left subtree.
2. Visit root.
3. Traverse the right subtree.

For this in-order traversal, start from the left child of the root node and keep exploring the left subtree until you reach a leaf. When you reach a leaf, back up to its parent, check for a right child and visit it if there is one. If there is not a child, you've explored its left and right subtrees fully. If there is a right child, traverse its left subtree then its right in the same manner. Keep doing this until you have traversed the entire tree. You will only store the values of a node as you visit when one of the following is true:

- it is the first node visited, the first time visited
- it is a leaf, should only be visited once
- all of its subtrees have been explored, should only be visited once while this is true
- it is the root of the tree, the first time visited

Swapping: Swapping subtrees of a node means that if initially node has left subtree L and right subtree R, then after swapping, the left subtree will be R and the right subtree, L.

For example, in the following tree, we swap children of node 1.

```
                                depth
    1               1            [1]
   / \             / \
  2   3     ->    3   2          [2]
   \   \           \   \
    4   5           5   4        [3]
```

In-order traversal of left tree is 2 4 1 3 5 and of right tree is 3 5 1 2 4.

Swap operation:

We define depth of a node as follows:

- The root node is at depth 1.

```
    }
    */
typedef struct TreeNode {
    int data;
    struct TreeNode* left;
    struct TreeNode* right;
} TreeNode;

// Function to perform in-order traversal of the binary tree
void inOrderTraversal(TreeNode* root, int* result, int* index) {
    if (root == NULL)
        return;

    inOrderTraversal(root->left, result, index);
    result[(*index)++] = root->data;
    inOrderTraversal(root->right, result, index);
}

// Function to swap subtrees at specified depths
void swapSubtrees(TreeNode* root, int k, int depth) {
    if (root == NULL)
```

You have earned 40.00 points!

Congratulations

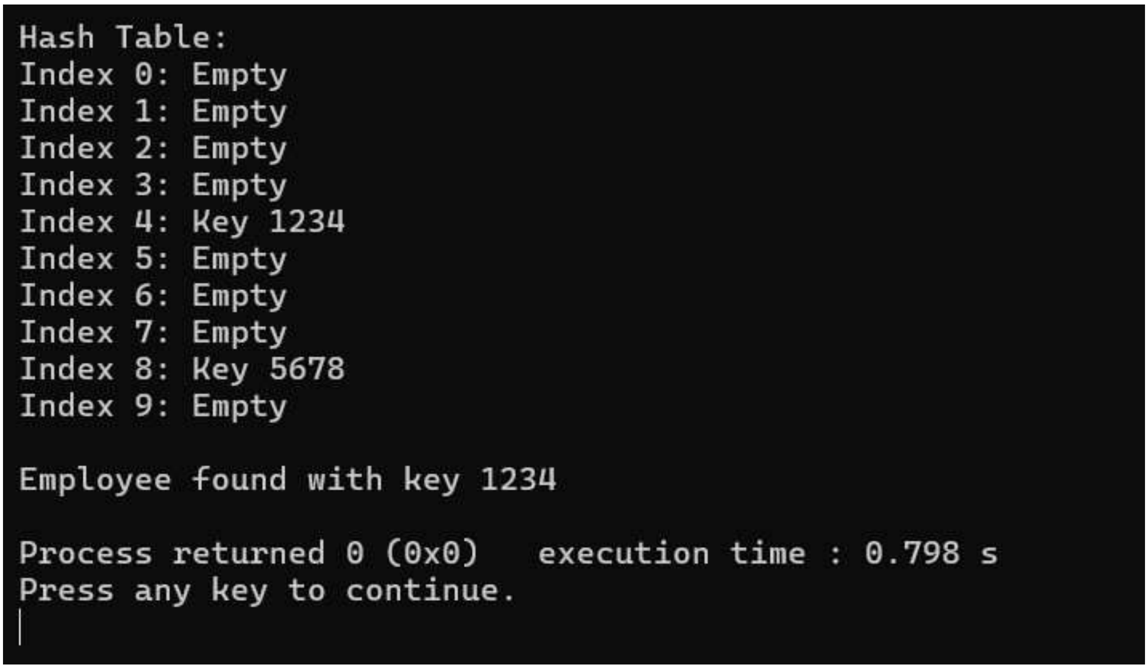
You solved this challenge. Would you like to challenge your friends?

Lab program 10:

Given a File of N employee records with a set K of Keys(4-digit) which uniquely determine the records in file F.
Assume that file F is maintained in memory by a Hash Table (HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT.
Let the keys in K and addresses in L are integers.
Design and develop a Program in C that uses Hash function H: K -&gt; L as H(K)=K mod m (remainder method), and implement hashing technique to map a given key K to the address space L.
Resolve the collision (if any) using linear probing.

```
#include <stdio.h>
#include <stdlib.h>
#define TABLE_SIZE 10 // Size of hash table
#define EMPTY -1 // Indicates empty cell in hash table
// Employee structure
struct Employee { int
key; // Unique key
// Add other employee data here
};
// Hash table structure struct
HashTable {
struct Employee* table[TABLE_SIZE];
};
// Hash function using remainder method
int hash(int key, int m) { return
key % m;
}
// Function to initialize hash table void
initHashTable(struct HashTable* ht) { for
(int i = 0; i < TABLE_SIZE; i++) { ht-
>table[i] = NULL;
}
}
// Function to insert employee record into hash table void
insert(struct HashTable* ht, struct Employee* emp) {
int index = hash(emp->key, TABLE_SIZE); //
Linear probing to resolve collisions
while (ht->table[index] != NULL && ht->table[index]->key != EMPTY) {
index = (index + 1) % TABLE_SIZE;
}
ht->table[index] = emp;
}
// Function to search for an employee record using key struct
Employee* search(struct HashTable* ht, int key) {
int index = hash(key, TABLE_SIZE);
while (ht->table[index] != NULL) { if
(ht->table[index]->key == key) {
return ht->table[index];
}
index = (index + 1) % TABLE_SIZE;
}
return NULL; // Employee not found
}
// Function to display hash table contents void
displayHashTable(struct HashTable* ht) { printf("Hash
Table:\n"); for (int i = 0; i < TABLE_SIZE; i++) { if (ht-
>table[i] != NULL && ht->table[i]->key != EMPTY) {
printf("Index %d: Key %d\n", i, ht->table[i]->key);
} else {
printf("Index %d: Empty\n", i);
}
} } int main() { struct HashTable ht;
initHashTable(&ht); // Example employee
records struct Employee emp1 = {1234}; //
Key: 1234 struct Employee emp2 = {5678}; //
Key: 5678 // Insert employee records into
hash table
insert(&ht, &emp1);
insert(&ht, &emp2); //
Display hash table
displayHashTable(&ht);
// Search for an employee
int keyToSearch = 1234;
struct Employee* foundEmp = search(&ht, keyToSearch); if
(foundEmp != NULL) {
printf("\nEmployee found with key %d\n", foundEmp->key);
} else {
printf("\nEmployee with key %d not found\n", keyToSearch); } return 0;
}
```

Output:



```
Hash Table:
Index 0: Empty
Index 1: Empty
Index 2: Empty
Index 3: Empty
Index 4: Key 1234
Index 5: Empty
Index 6: Empty
Index 7: Empty
Index 8: Key 5678
Index 9: Empty

Employee found with key 1234

Process returned 0 (0x0)   execution time : 0.798 s
Press any key to continue.
```