VISVESVARAYATECHNOLOGICALUNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

Harsha B (1BM23CS107)

in partial fulfillment 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

September 2024-January 2025

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 Harsha B (1BM23CS107) , who is bonafide student of **B. M. S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2024- 25. 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.

Dr. Selva kumar S	**Dr. Kavitha Sooda**
Associate Professor	Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	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

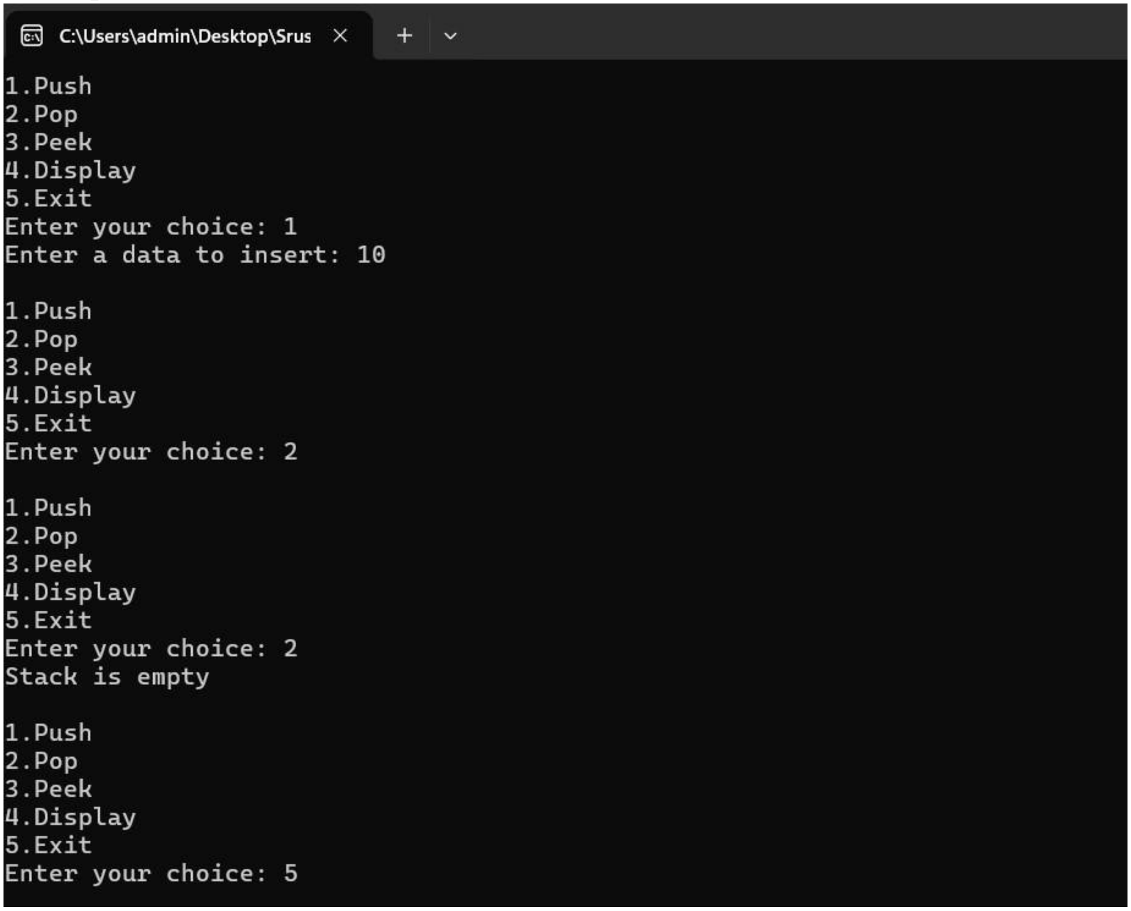
c) Display

The program should print appropriate messages for stack overflow, stack underflow.

```
#include<stdio.h>
#include<stdlib.h
> #define max 4
int stack[max];
int top=-1;

void push(int x){
   if(top==max-1){ printf("Stack
      is full\n");
   }
   else{
      top++;
      stack[top]=x;
   }

void       pop(){       if(top==-1){
printf("Stack is empty\n");
   }
   else{  top--
      ;
   }
}

void peek(){ if(top!=-1){
   printf("%d",stack[top]);
   }
}

void    display(){    if(top==-1){
   printf("Stack is empty\n");
   }
   else{ for(int i=top;i>=0;i--){
      printf("%d\n",stack[i]);
      }
   }
}

void   main(){   int
   choice,data;
   while(1){
   printf("\n1.Pus
   h         \n2.Pop
   \n3.Peek
```

```
          \n4.Display
          \n5.Exit\n");
            printf("Enter your choice: ");
            scanf("%d",&choice);
            switch(choice){
              case 1:printf("Enter a data to insert:
                  "); scanf("%d",&data); push(data);
                  break;
              case 2:pop(); break;
              case 3:peek(); break;
              case 4:display(); break;
              case 5:exit(0);
                  break;
              default:printf("Invalid choice.");
            }
          }
        }
```

Output:



```
1.Push
2.Pop
3.Peek
4.Display
5.Exit
Enter your choice: 1
Enter a data to insert: 10

1.Push
2.Pop
3.Peek
4.Display
5.Exit
Enter your choice: 2

1.Push
2.Pop
3.Peek
4.Display
5.Exit
Enter your choice: 2
Stack is empty

1.Push
2.Pop
3.Peek
4.Display
5.Exit
Enter your choice: 5
```

Lab program 2a:

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)

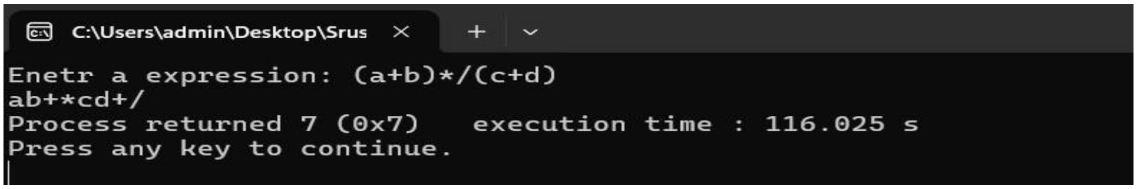
```
#include<stdio.h>
```

```
#include<string.h>
#define  max  30  char
s1[max];//postfix char
s2[max];//infix       int
t1=-1,t2=-1;

int precedence(char x){
   if(x=='*'   ||   x=='/'){
   return 2;
   }
   else if(x=='+' || x=='-'){ return
      1;
   }

   else if(x=='^'){ return
      3;
   }
   return 0;
}

void push1(char x){
   if(t1==max-1){
      printf("Stack is full\n");
      return;
   }
   else{ t1++;
      s1[t1]=x
      ;
   }
}

void push2(char x){
   if(t2==max-1){
      printf("Stack is full\n");
      return;
   }
   else{
      t2++;
      s2[t2]=x;
   }
}
```

```
char pop1(){ if(t1==-1){
   printf("Stack is empty\n");
      return '\0';
   }
   return s1[t1--];
}

char      pop2(){      if(t2==-1){
   printf("Stack   is   empty\n");
   return '\0';
   }
   return s2[t2--];
}

char peek2(){ if(t2!=-
   1){

      return s2[t2];
   }
   return '\0';
}

void main(){ char
   str[max];
   printf("Enetr a expression: ");
   scanf("%s",str);

   for(int i=0;i<strlen(str);i++){ char
      ch=str[i];

      if(ch=='('){
         push2(ch);
      }
      else if(ch==')'){
         while(peek2()!='('){
         push1(pop2());
         }
         pop2();
      }
      else  if(ch=='+'||  ch=='-'||  ch=='*'   ||   ch=='/'   ||   ch=='^'){
         while(t2>-1 && precedence(peek2())>=precedence(ch)){
         push1(pop2());
         }
         push2(ch);
      }
```

```
        else if(ch>='a' && ch<='z'){ push1(ch);
        }
    }

    while(t2>-1){
        push1(pop2());
    }
    for(int                    i=0;i<=t1;i++){
        printf("%c",s1[i]);
    }
```

Output:



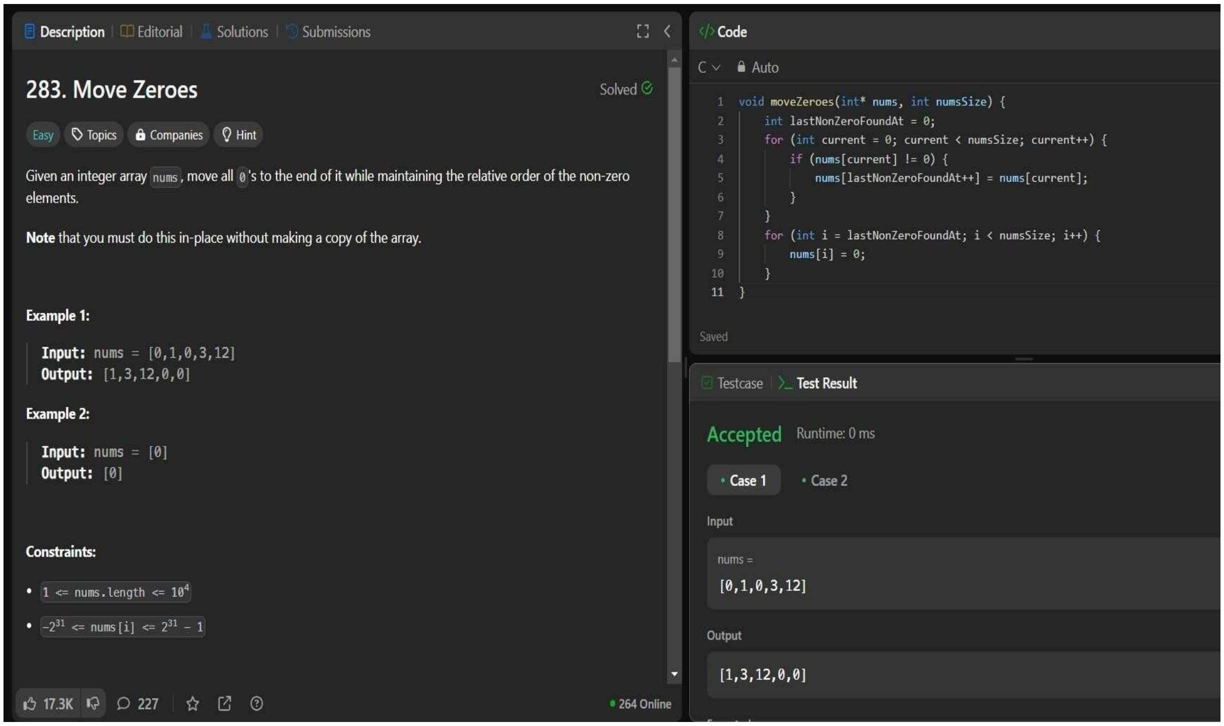
```
Enetr a expression: (a+b)*/(c+d)
ab++cd+/
Process returned 7 (0x7)   execution time : 116.025 s
Press any key to continue.
```

Lab program 2b:

Demonstration of account creation on LeetCode platform Program - Leetcode platform.

```
void moveZeroes(int* nums, int
numsSize) { int lastNonZeroFoundAt = 0; for (int current = 0; current <
numsSize; current++) { if (nums[current] != 0) {
nums[lastNonZeroFoundAt++] = nums[current];
                }
        }
        for (int i = lastNonZeroFoundAt; i < numsSize; i++) {
                nums[i] = 0;
        }
}
```

Output:



283. Move Zeroes

Given an integer array `nums`, move all `0`'s to the end of it while maintaining the relative order of the non-zero elements.

Note that you must do this in-place without making a copy of the array.

Example 1:

```
Input: nums = [0,1,0,3,12]
Output: [1,3,12,0,0]
```

Example 2:

```
Input: nums = [0]
Output: [0]
```

Constraints:

- `1 <= nums.length <=` 10^4
- -2^{31} `<= nums[i] <=` $2^{31} - 1$

```
void moveZeroes(int* nums, int numsSize) {
    int lastNonZeroFoundAt = 0;
    for (int current = 0; current < numsSize; current++) {
        if (nums[current] != 0) {
            nums[lastNonZeroFoundAt++] = nums[current];
        }
    }
    for (int i = lastNonZeroFoundAt; i < numsSize; i++) {
        nums[i] = 0;
    }
}
```

Accepted Runtime: 0 ms

Input

```
nums =
[0,1,0,3,12]
```

Output

```
[1,3,12,0,0]
```

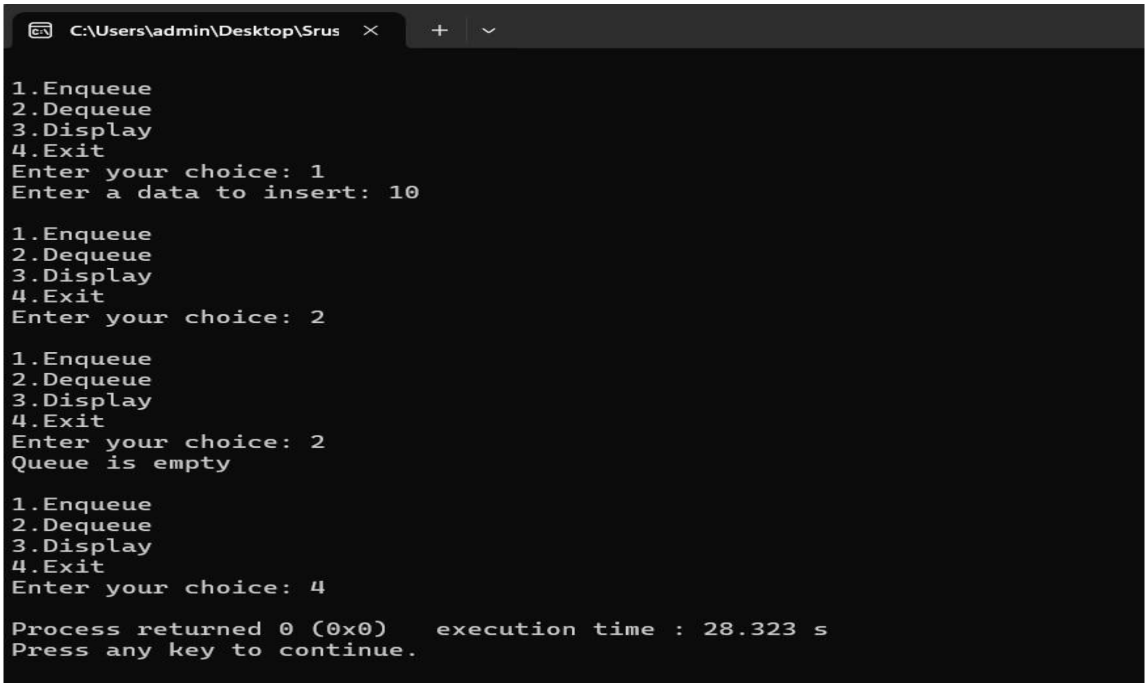
Lab program 3a:

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>
#include<stdlib.h>
#define    n    4    int
queue[n];            int
front=-1,rear=-1;

void        enqueue(int        x){
   if(rear==n-1){
   printf("Queue is full\n");
   }
   else if(rear==-1 && front==-1){ rear=front=0;
      queue[rear]=x;
   } else{
   rear++;
      queue[rear]=x;
   }
}

void dequeue(){ if(rear==-1 &&
   front==-1){  printf("Queue  is
   empty\n");
   }
   else if(front==rear){ front=rear=-1;
   }
   else{
      front++;
   }
}

void display(){  if(rear==-1  &&
   front==-1){  printf("Queue  is
   empty\n");
   } else{ for(int
   i=front;i<=rear;i++){
   printf("%d\n",queue[i]);
      }
   } }
```

```
void main(){ int choice,data; while(1){ printf("\n1.Enqueue
  \n2.Dequeue \n3.Display \n4.Exit\n"); printf("Enter your
  choice: "); scanf("%d",&choice); switch(choice){ case
  1:printf("Enter a data to insert: "); scanf("%d",&data);
  enqueue(data); break;
        case 2:dequeue(); break;
        case 3:display(); break;
        case 4:exit(0); break;
        default:printf("Invalid choice.");
    }
  }
}
```

Output:



```
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice: 1
Enter a data to insert: 10

1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice: 2

1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice: 2
Queue is empty

1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice: 4

Process returned 0 (0x0)   execution time : 28.323 s
Press any key to continue.
```

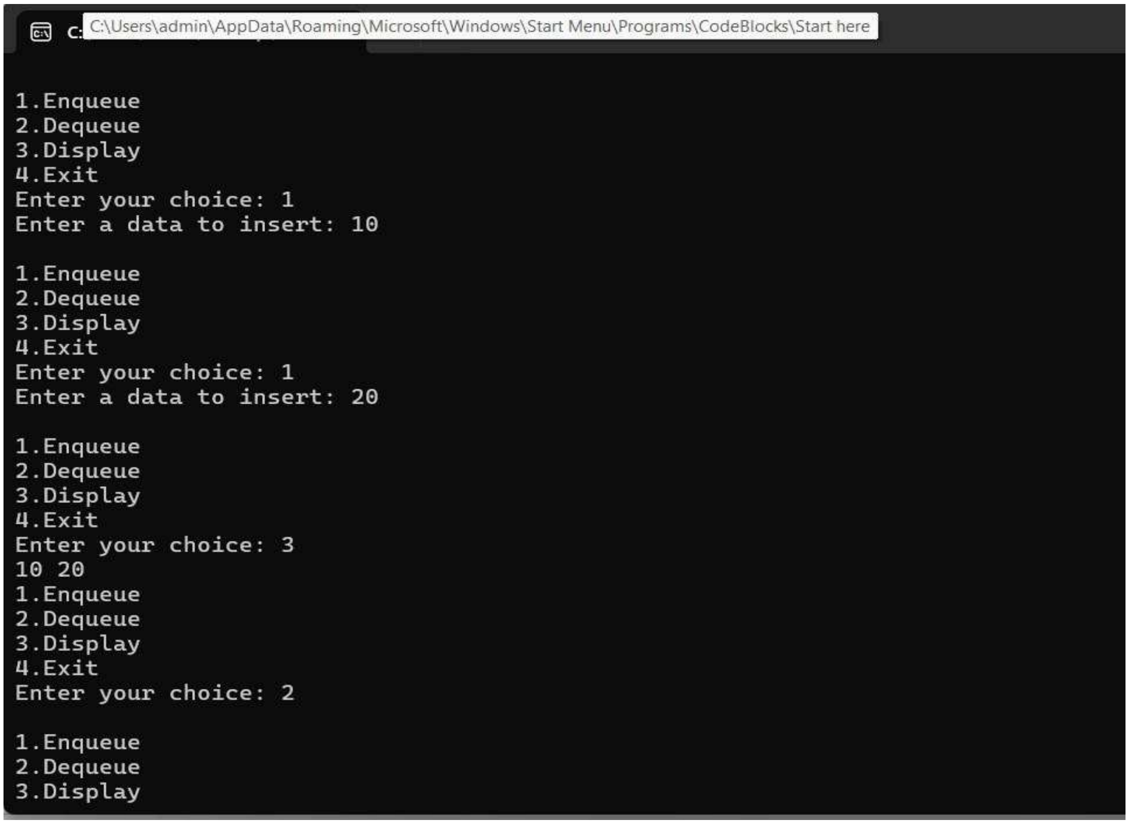
Lab program 3b:

WAP to simulate the working of a circular 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

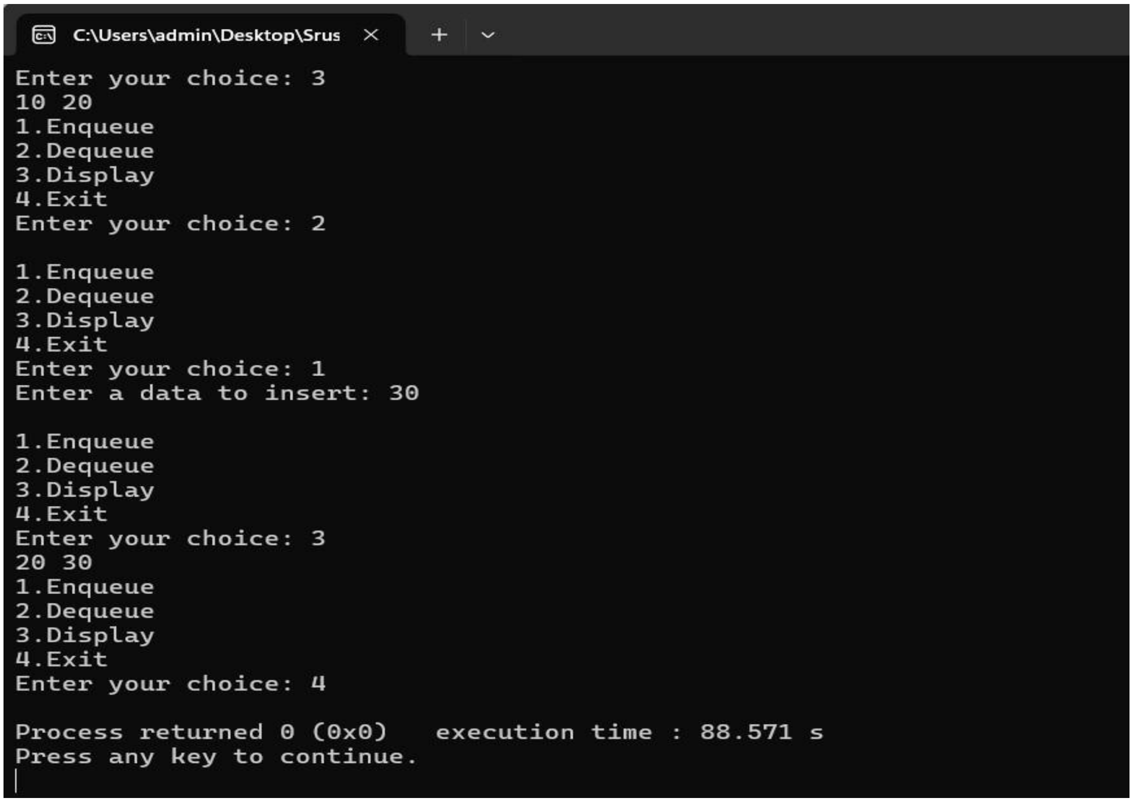
```
#include<stdio.h>
#include<stdlib.h>
#  define  n  4  int
q[n];
int rear=-1, front=-1;

void        enqueue(int        x){
   if(front==(rear+1)%n){
   printf("Queue is full\n");
   }
   else if(front==-1 && rear==-1){
      front=rear=0;
      q[rear]=x;
   }
   else{
      rear=(rear+1)%n;
      q[rear]=x;
   }
}

void dequeue(){ if(front==-1 &&
   rear==-1){    printf("Queue    is
   Empty\n");
   }
   else if(rear==front){ front=rear=-1;
   }
   else {
      front=(front+1)%n;
   }
}

void  display(){  if(front==-1  &&
   rear==-1){    printf("Queue    is
   empty\n");
   }
   else{                           for(int
      i=front;i!=rear;i++){
      printf("%d ",q[i]);
      }
      printf("%d",q[rear]);
   }
}

void   main(){   int   choice,data;   while(1){   printf("\n1.Enqueue
   \n2.Dequeue \n3.Display \n4.Exit\n");
```

```
        printf("Enter your choice: ");
        scanf("%d",&choice);
        switch(choice){
          case 1:printf("Enter a data to insert: ");
                scanf("%d",&data);
                enqueue(data);
                break;
          case 2:dequeue();
                break;
          case 3:display();
                break;
          case 4:exit(0);
               break;
          default:printf("Invalid choice.");
        }
    }
}
```

Output:


C:\Users\admin\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\CodeBlocks\Start here

```
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice: 1
Enter a data to insert: 10

1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice: 1
Enter a data to insert: 20

1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice: 3
10 20
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice: 2

1.Enqueue
2.Dequeue
3.Display
```



```
Enter your choice: 3
10 20
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice: 2

1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice: 1
Enter a data to insert: 30

1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice: 3
20 30
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice: 4

Process returned 0 (0x0)   execution time : 88.571 s
Press any key to continue.
```

Lab program 4a:

WAP to Implement Singly Linked List with following operations a) Createalinkedlist.

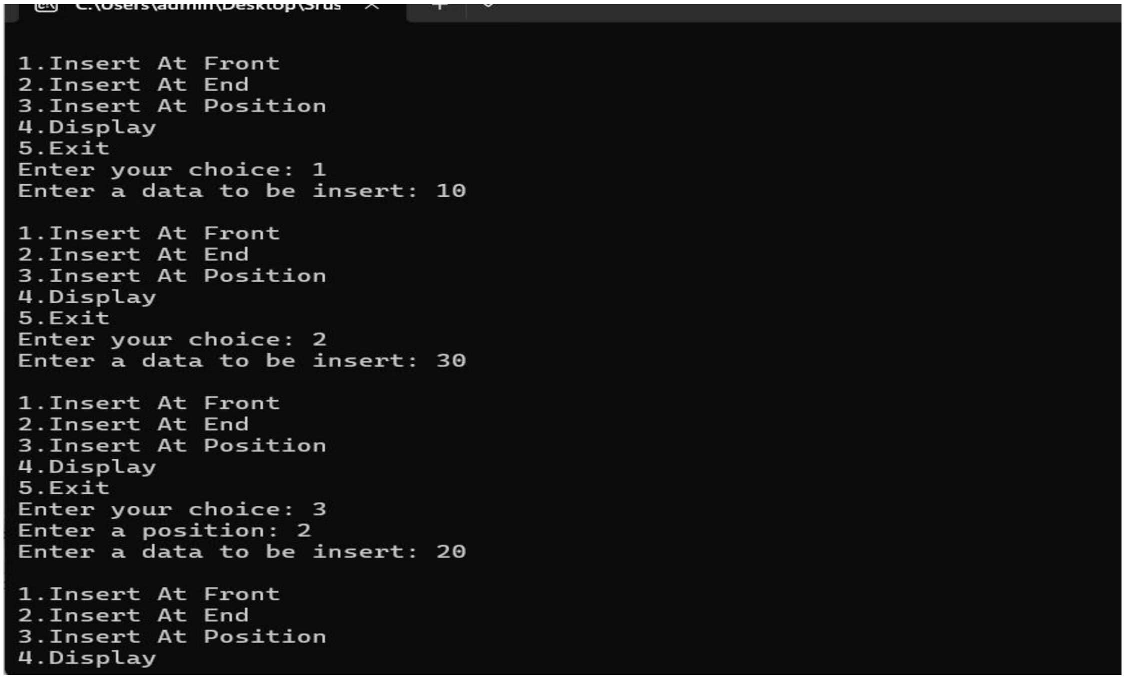
b) Insertion of a node at first position, at any position and at end of list.

Display the contents of the linked list.

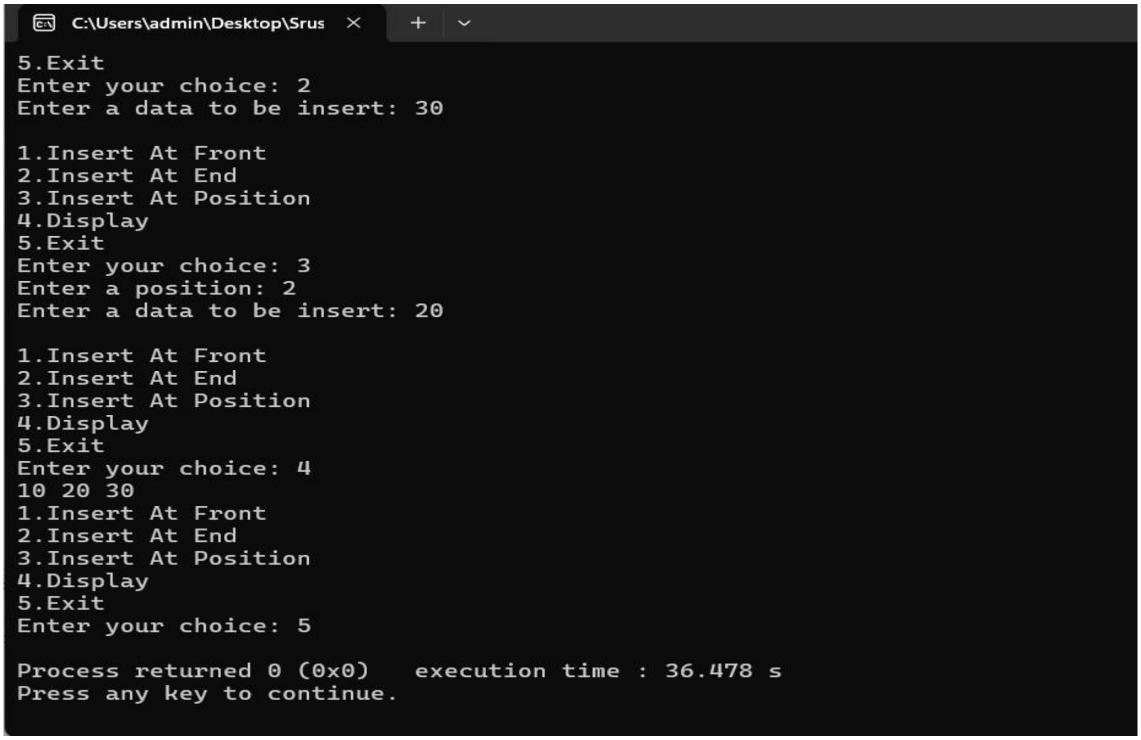
```
#include<stdio.h>
#include<stdlib.h>

struct    Node{    int
   data;  struct  Node
   *next;
};

struct Node *CreateNode(int data){ struct Node *newnode=(struct
   Node*)malloc(sizeof(struct Node)); newnode->data=data;
   newnode->next=NULL;
   return newnode;
}

void insertAtFirst(struct Node* *head,int data){
   struct Node* newnode=CreateNode(data);
```

```
        newnode->next=*head;
        *head=newnode;
}

void insertAtEnd(struct Node* *head,int data){
    struct   Node*   newnode=CreateNode(data);
    if(*head==NULL){ *head=newnode;
        return;
    }
    struct Node *temp=*head; while(temp-
    >next!=NULL){ temp=temp->next;
    }
    temp->next=newnode;
}

void insertAtPos(struct Node* *head,int data,int pos){
    struct        Node*        newnode=CreateNode(data);
    if(pos==1){                            newnode->next=*head;
    *head=newnode;
        return;
    }
    struct Node *temp=*head;
    for(int i=1;i<pos-1 && temp!=NULL;i++){ temp=temp-
        >next;
    }
    if(temp==NULL){
        printf("Position out of range.\n");
        free(newnode);
        return;
    }
    newnode->next=temp->next;
    temp->next=newnode;
}

void display(struct Node *head){
    struct    Node    *temp=head;
    while(temp!=NULL){
    printf("%d      ",temp->data);
    temp=temp->next;
    }
}

void   main(){   struct   Node
    *head=NULL;   int   data,
    choice,pos; while(1){
        printf("\n1.Insert At Front \n2.Insert At End \n3.Insert At Position \n4.Display
        \n5.Exit"); printf("\nEnter your choice: "); scanf("%d",&choice); switch(choice){ case
        1:printf("Enter a data to be insert: "); scanf("%d",&data); insertAtFirst(&head,data);
        break;
          case 2:printf("Enter a data to be insert:
                "); scanf("%d",&data);
                insertAtEnd(&head,data); break;
          case 3:printf("Enter a position: "); scanf("%d",&pos);
                printf("Enter a data to be insert: ");
                scanf("%d",&data);
                insertAtPos(&head,data,pos);
                break;
          case 4:display(head); break;
          case 5:exit(0); break;
          default:printf("Invalid choice.");

        }
    }
}
```

Output:



```
1.Insert At Front
2.Insert At End
3.Insert At Position
4.Display
5.Exit
Enter your choice: 1
Enter a data to be insert: 10

1.Insert At Front
2.Insert At End
3.Insert At Position
4.Display
5.Exit
Enter your choice: 2
Enter a data to be insert: 30

1.Insert At Front
2.Insert At End
3.Insert At Position
4.Display
5.Exit
Enter your choice: 3
Enter a position: 2
Enter a data to be insert: 20

1.Insert At Front
2.Insert At End
3.Insert At Position
4.Display
```



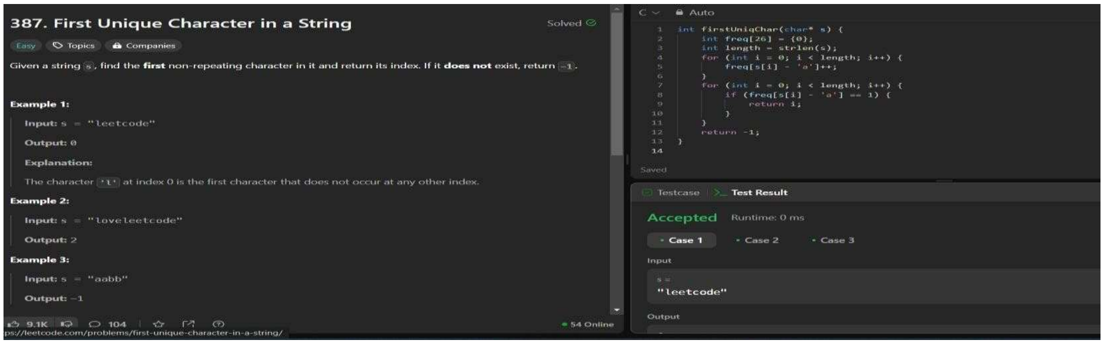
```
5.Exit
Enter your choice: 2
Enter a data to be insert: 30

1.Insert At Front
2.Insert At End
3.Insert At Position
4.Display
5.Exit
Enter your choice: 3
Enter a position: 2
Enter a data to be insert: 20

1.Insert At Front
2.Insert At End
3.Insert At Position
4.Display
5.Exit
Enter your choice: 4
10 20 30
1.Insert At Front
2.Insert At End
3.Insert At Position
4.Display
5.Exit
Enter your choice: 5

Process returned 0 (0x0)   execution time : 36.478 s
Press any key to continue.
```

Lab program 4b:

Program - Leetcode platform

```
int firstUniqChar(char* s) { int
      freq[26] = {0}; int length =
      strlen(s); for (int i = 0; i <
      length; i++) { freq[s[i] -
      'a']++;
      }
      for (int i = 0; i < length; i++) {
            if (freq[s[i] - 'a'] == 1) {
            return i;
            }
      }
      return -1;
}
```

Output:



387. First Unique Character in a String

Given a string s, find the **first** non-repeating character in it and return its index. If it **does not** exist, return -1.

Example 1:

Input: s = "leetcode"

Output: 0

Explanation:

The character 'l' at index 0 is the first character that does not occur at any other index.

Example 2:

Input: s = "loveleetcode"

Output: 2

Example 3:

Input: s = "aabb"

Output: -1

```
int firstUniqChar(char* s) {
    int freq[26] = {0};
    int length = strlen(s);
    for (int i = 0; i < length; i++) {
        freq[s[i] - 'a']++;
    }
    for (int i = 0; i < length; i++) {
        if (freq[s[i] - 'a'] == 1) {
            return i;
        }
    }
    return -1;
}
```

Accepted Runtime: 0 ms

Input

s =

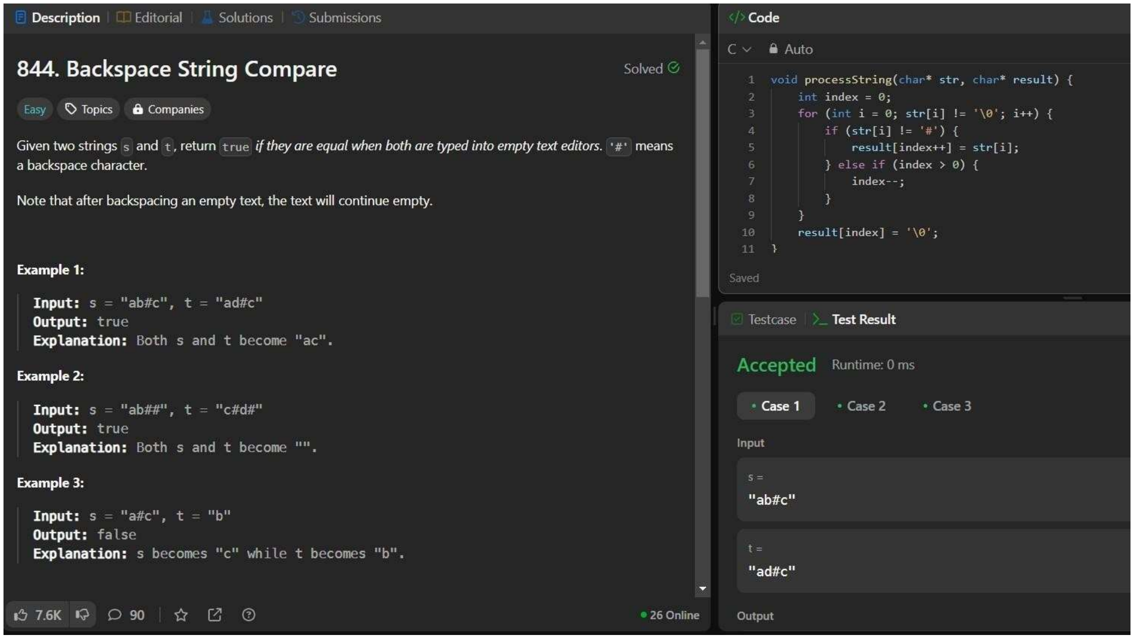
"leetcode"

Output

Program - Leetcode platform

```
void processString(char* str, char* result) { int
            index = 0;
             for (int i = 0; str[i] != '\0'; i++) { if
                        (str[i] != '#') { result[index++]
                        = str[i];
                        } else if (index >
                        0) { index--;
                        }
            }
            result[index] = '\0';
}
bool  backspaceCompare(char*  s,  char*  t)  {  char
            processedS[201];     char     processedT[201];
            processString(s,                     processedS);
            processString(t,       processedT);       return
            strcmp(processedS, processedT) == 0;
}
```

Output:



844. Backspace String Compare

Given two strings s and t, return true *if they are equal when both are typed into empty text editors.* '#' means a backspace character.

Note that after backspacing an empty text, the text will continue empty.

Example 1:

```
Input: s = "ab#c", t = "ad#c"
Output: true
Explanation: Both s and t become "ac".
```

Example 2:

```
Input: s = "ab##", t = "c#d#"
Output: true
Explanation: Both s and t become "".
```

Example 3:

```
Input: s = "a#c", t = "b"
Output: false
Explanation: s becomes "c" while t becomes "b".
```

```
void processString(char* str, char* result) {
    int index = 0;
    for (int i = 0; str[i] != '\0'; i++) {
        if (str[i] != '#') {
            result[index++] = str[i];
        } else if (index > 0) {
            index--;
        }
    }
    result[index] = '\0';
}
```

Accepted Runtime: 0 ms

Input

s =

"ab#c"

t =

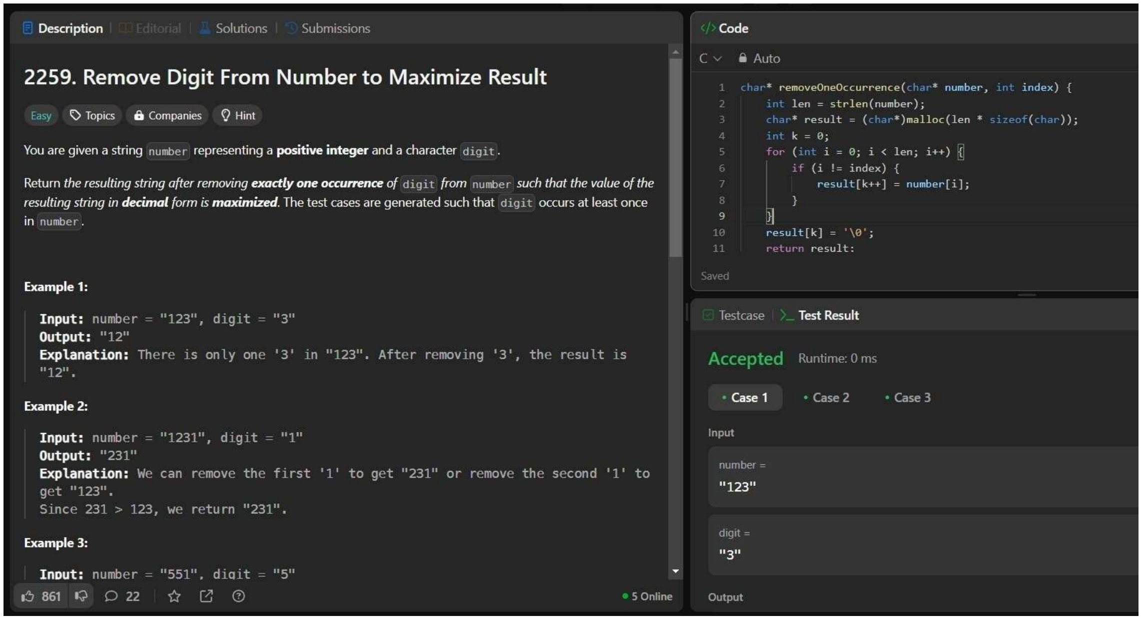
"ad#c"

Output

Program - Leetcode platform

```
char* removeOneOccurrence(char* number, int index) { int
    len = strlen(number);
    char* result = (char*)malloc(len * sizeof(char));
    int k = 0; for (int i = 0; i <
    len; i++) { if (i != index) {
        result[k++] = number[i];
        }
}
    result[k] = '\0'; return
    result;
}
char* removeDigit(char* number, char digit) { char*
    maxString = NULL;
    for (int i = 0; number[i] != '\0'; i++) { if
        (number[i] == digit) {
        char* newString = removeOneOccurrence(number, i); if (maxString
            == NULL || strcmp(newString, maxString) > 0) { if (maxString
            != NULL) { free(maxString);
                }
            maxString = newString;
            }
            else { free(newString);
            }
        }
    }
    return maxString;
}
```

Output:



2259. Remove Digit From Number to Maximize Result

You are given a string `number` representing a **positive integer** and a character `digit`.

Return *the resulting string after removing **exactly one occurrence** of* `digit` *from* `number` *such that the value of the resulting string in **decimal** form is **maximized***. The test cases are generated such that `digit` occurs at least once in `number`.

Example 1:

```
Input: number = "123", digit = "3"
Output: "12"
Explanation: There is only one '3' in "123". After removing '3', the result is
"12".
```

Example 2:

```
Input: number = "1231", digit = "1"
Output: "231"
Explanation: We can remove the first '1' to get "231" or remove the second '1' to
get "123".
Since 231 > 123, we return "231".
```

Example 3:

```
Input: number = "551", digit = "5"
```

```
char* removeOneOccurrence(char* number, int index) {
    int len = strlen(number);
    char* result = (char*)malloc(len * sizeof(char));
    int k = 0;
    for (int i = 0; i < len; i++) {
        if (i != index) {
            result[k++] = number[i];
        }
    }
    result[k] = '\0';
    return result:
```

Accepted Runtime: 0 ms

Input

number =
"123"

digit =
"3"

Output

Lab program 5a:

WAP to Implement Singly Linked List with following operations a) Create a linked list.

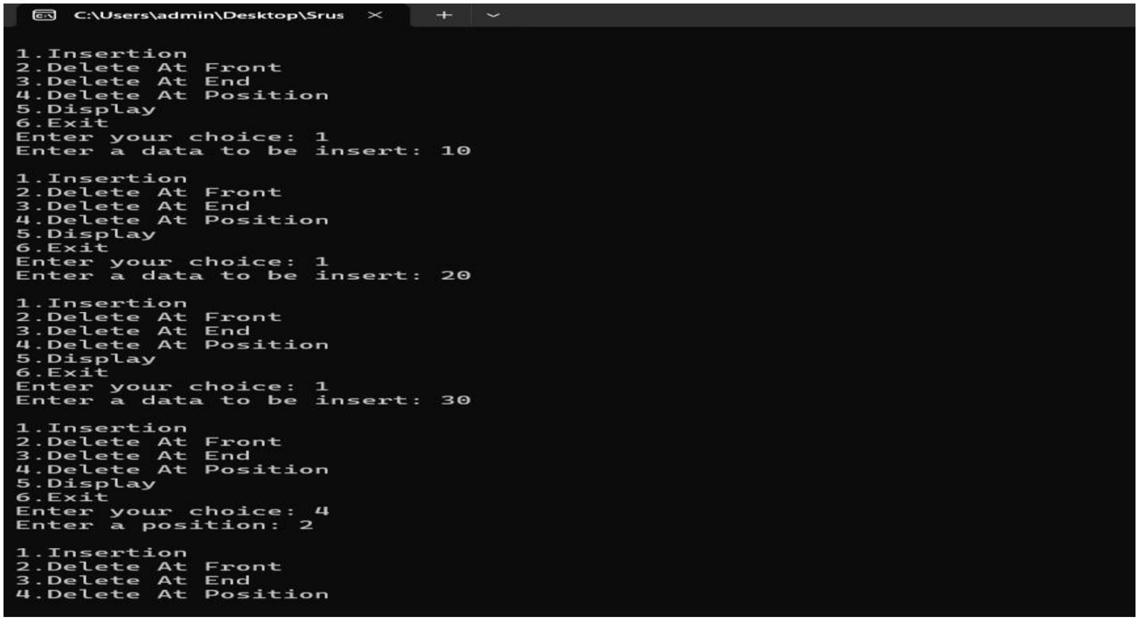
b) Deletion of first element, specified element and last element in the list.

c) Display the contents of the linked list.

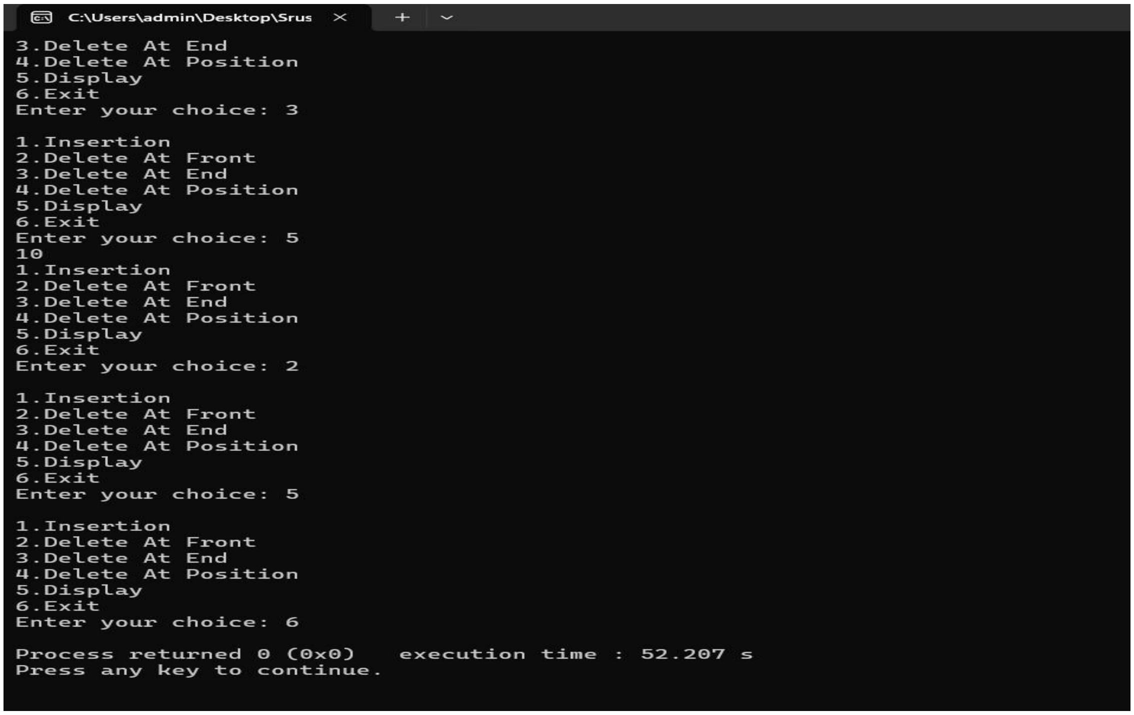
```
#include<stdio.h>
#include<stdlib.h>

struct Node{ int
   data;
   struct Node *next;
};

struct Node *CreateNode(int data){ struct Node *newnode=(struct
   Node*)malloc(sizeof(struct Node)); newnode->data=data;
   newnode->next=NULL;
   return newnode;
}

void insertAtEnd(struct Node* *head,int data){
   struct   Node*   newnode=CreateNode(data);
   if(*head==NULL){ *head=newnode;
      return;
   }
   struct Node *temp=*head; while(temp-
   >next!=NULL){ temp=temp->next;
   }
   temp->next=newnode;
}

void deleteAtFront(struct Node* *head){
   if(*head==NULL){       printf("List       is
   Empty\n");
   }
   struct Node *temp1=*head; struct
   Node *temp2=temp1->next;
   *head=temp2;
   free(temp1);
}

void deleteAtEnd(struct Node* *head){
   if(*head==NULL){      printf("List      is
   empty\n");
   }

   struct Node *temp1=*head; struct
   Node *temp2=NULL;

   if(temp1->next==NULL){
      free(temp1);
```

```
        *head=NULL;
    }

    while(temp1->next!=NULL){
        temp2=temp1;
        temp1=temp1->next;
    }
    temp2->next=NULL;
    free(temp1);
}

void deleteAtPos(struct Node* *head,int pos){
    if(*head==NULL){ printf("List
        is empty\n");
    }
    struct Node *temp1=*head; struct
    Node *temp2=NULL;
    if(pos==1){
        *head=temp1->next;
        free(temp1);
    }
    for(int i=1;i<pos-1 && temp1!=NULL;i++){
        temp1=temp1->next;
    }
    if(temp1==NULL       ||       temp1->next==NULL){
        printf("Position out of range\n");
    }
    temp2=temp1->next;          temp1->next=temp2-
    >next;
    free(temp2);
}

void display(struct Node *head){
    struct   Node   *temp=head;
    while(temp!=NULL){
    printf("%d ",temp->data);
        temp=temp->next;
    }
}

void main(){
    struct Node *head=NULL; int data, choice,pos; while(1){ printf("\n1.Insertion \n2.Delete At Front
    \n3.Delete At End \n4.Delete At Position \n5.Display
\n6.Exit"); printf("\nEnter your
        choice: ");
        scanf("%d",&choice);
        switch(choice){
            case 1:printf("Enter a data to be insert: ");
                   scanf("%d",&data);
                   insertAtEnd(&head,data);
            break;
        case          2:deleteAtFront(&head);
            break;
        case 3:deleteAtEnd(&head);
            break;
        case 4:printf("Enter a position: ");
            scanf("%d",&pos);
            deleteAtPos(&head,pos);
            break;
        case 5:display(head);
            break;
        case 6:exit(0);
            break;
        default:printf("Invalid choice.");

      }
   }
}
```

Output:



```
1.Insertion
2.Delete At Front
3.Delete At End
4.Delete At Position
5.Display
6.Exit
Enter your choice: 1
Enter a data to be insert: 10

1.Insertion
2.Delete At Front
3.Delete At End
4.Delete At Position
5.Display
6.Exit
Enter your choice: 1
Enter a data to be insert: 20

1.Insertion
2.Delete At Front
3.Delete At End
4.Delete At Position
5.Display
6.Exit
Enter your choice: 1
Enter a data to be insert: 30

1.Insertion
2.Delete At Front
3.Delete At End
4.Delete At Position
5.Display
6.Exit
Enter your choice: 4
Enter a position: 2

1.Insertion
2.Delete At Front
3.Delete At End
4.Delete At Position
```



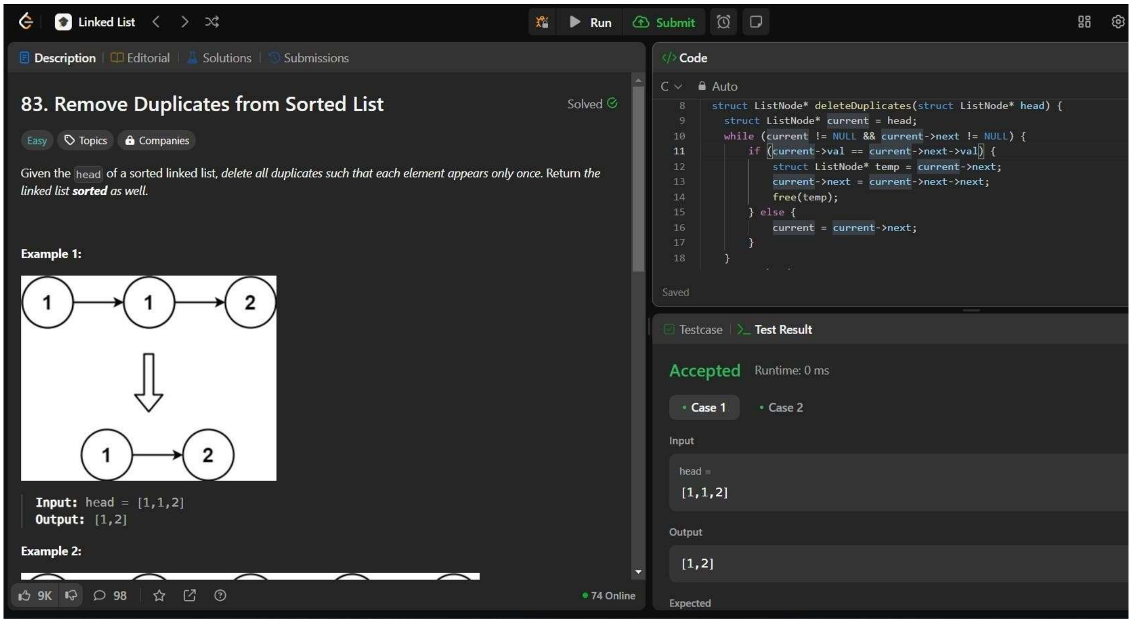
```
3.Delete At End
4.Delete At Position
5.Display
6.Exit
Enter your choice: 3

1.Insertion
2.Delete At Front
3.Delete At End
4.Delete At Position
5.Display
6.Exit
Enter your choice: 5
10
1.Insertion
2.Delete At Front
3.Delete At End
4.Delete At Position
5.Display
6.Exit
Enter your choice: 2

1.Insertion
2.Delete At Front
3.Delete At End
4.Delete At Position
5.Display
6.Exit
Enter your choice: 5

1.Insertion
2.Delete At Front
3.Delete At End
4.Delete At Position
5.Display
6.Exit
Enter your choice: 6

Process returned 0 (0x0)   execution time : 52.207 s
Press any key to continue.
```

Lab program 5b:

Program - Leetcode platform

```
struct ListNode*
deleteDuplicates(struct ListNode* head) { struct ListNode*
current = head;
        while (current != NULL && current->next != NULL) { if
                (current->val == current->next->val) { struct
                ListNode* temp = current->next; current->next
                = current->next->next; free(temp);
                }       else
                {
                        current = current->next;
                }
        }
        return head;
}
struct ListNode* createNode(int val) { struct ListNode* newNode = (struct
        ListNode*)malloc(sizeof(struct ListNode)); newNode->val = val; newNode-
        >next = NULL;
        return newNode;
}
void printList(struct ListNode* head) { struct
        ListNode* current = head; while
        (current != NULL) { printf("%d -> ",
        current->val); current = current-
        >next;
        }
        printf("NULL\n");
}
```

Output:



83. Remove Duplicates from Sorted List

Solved

Easy · Topics · Companies

Given the `head` of a sorted linked list, *delete all duplicates such that each element appears only once. Return the linked list* ***sorted*** *as well.*

Example 1:

```
Input: head = [1,1,2]
Output: [1,2]
```

Example 2:

```
struct ListNode* deleteDuplicates(struct ListNode* head) {
  struct ListNode* current = head;
  while (current != NULL && current->next != NULL) {
      if (current->val == current->next->val) {
          struct ListNode* temp = current->next;
          current->next = current->next->next;
          free(temp);
      } else {
          current = current->next;
      }
  }
```

Accepted   Runtime: 0 ms

Case 1   Case 2

Input

```
head =
[1,1,2]
```

Output

```
[1,2]
```

Expected

Lab program 6a:

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

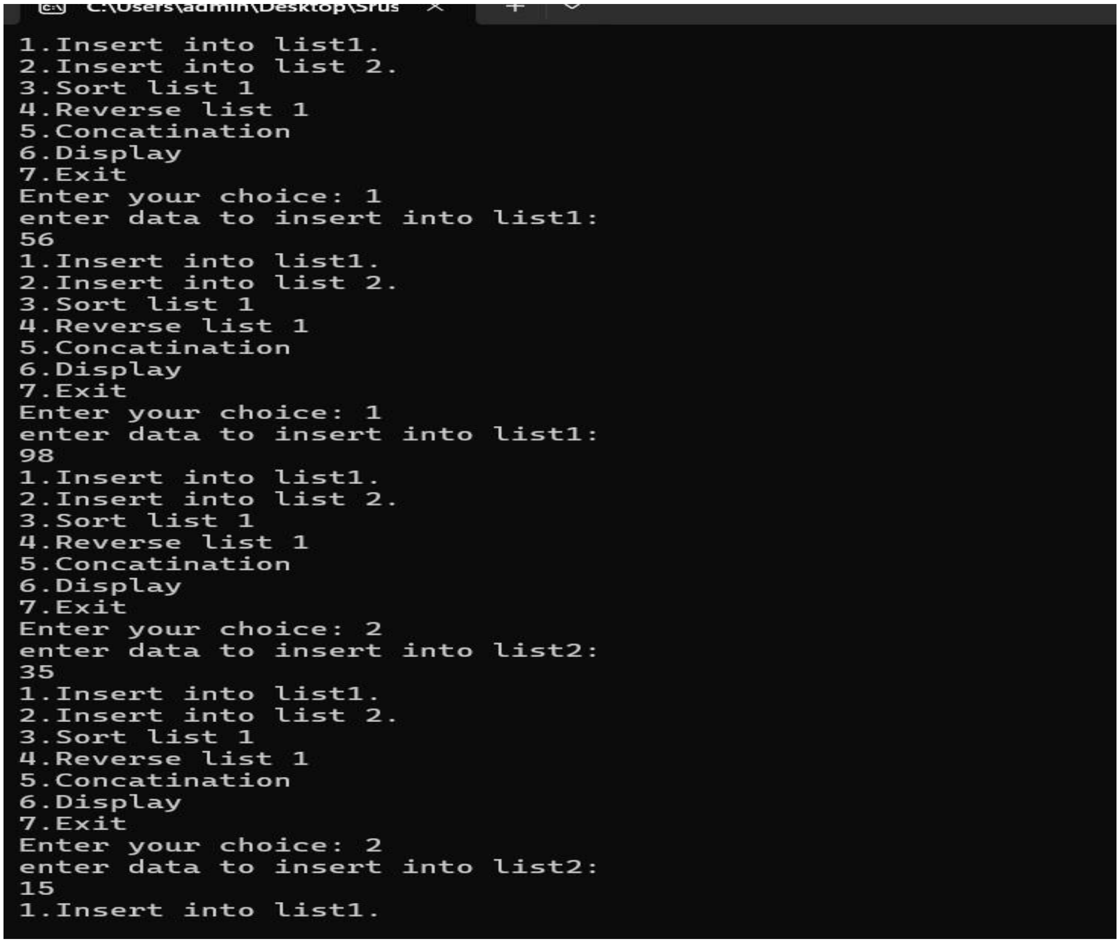
```
#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>

struct    Node{    int
   data;  struct  Node
   *next;
};

struct Node *createNode(int data){
   struct Node *newnode=(struct Node *)malloc(sizeof(struct Node));
   newnode->data=data; newnode->next=NULL; return newnode;
}

void insertEnd(struct Node* *head,int data){
   struct Node *newnode=createNode(data);
   if(*head==NULL){
       *head=newnode;
```

```
   }else{
   struct Node *temp=*head; while(temp-
   >next!=NULL){ temp=temp->next;
   }
   temp->next=newnode;
   }
}

void       sort(struct       Node*       *head){
   if(*head==NULL || (*head)->next==NULL){
   return;
   }
   struct   Node   *i=*head;
   while(i!=NULL){      struct
   Node             *j=i->next;
   while(j!=NULL){        if(i-
   >data   >   j->data){   int
   temp=i->data;  i->data=j-
   >data;
         j->data=temp;
       }
       j=j->next;
     }
     i=i->next;
   }
}

void reverse(struct Node* *head){
   struct Node *prev=NULL; struct
   Node *curr=*head; struct Node
   *nextn=NULL;

   while(curr!=NULL){
     nextn=curr->next;
     curr->next=prev;
     prev=curr;
     curr=nextn;
   }
   *head=prev;
}

void concate(struct Node* *head1,struct Node* *head2){
   if(*head1==NULL){
     *head1=*head2;
   }
```

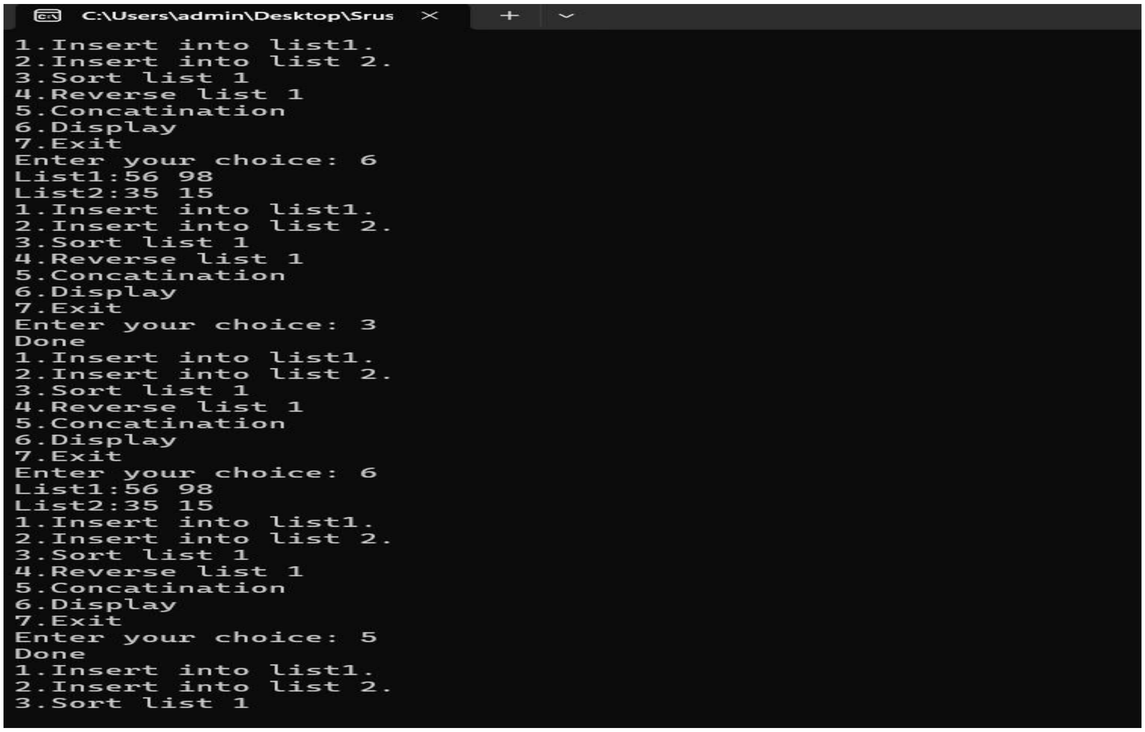
```
    struct Node *temp=*head1; while(temp-
    >next!=NULL){ temp=temp->next;
    }
    temp->next=*head2;
    *head2=NULL;
}
void display(struct Node* head){
    struct    Node    *temp=head;
    while(temp!=NULL){
    printf("%d        ",temp->data);
    temp=temp->next;
    }
    printf("\n");
}

void main()
{
    struct Node *list1=NULL;
    struct Node *list2=NULL; int
    data,choice;

    while(true){ printf("1.Insert into
        list1.\n"); printf("2.Insert into
        list  2.\n");  printf("3.Sort  list
        1\n");   printf("4.Reverse   list
        1\n");
        printf("5.Concatination\n");
        printf("6.Display\n");
        printf("7.Exit\n");

        printf("Enter your choice: "); scanf("%d",&choice);
        switch(choice){ case 1:printf("enter data to insert
        into list1: \n"); scanf("%d",&data);
        insertEnd(&list1,data); break;
            case 2:printf("enter data to insert into list2:
                  \n"); scanf("%d",&data);
                  insertEnd(&list2,data); break;
            case 3:sort(&list1);
                  printf("Done\n");
                  break;
            case 4:reverse(&list1);
```

```
                printf("Done\n");
                break;
            case 5:concate(&list1,&list2);
                printf("Done\n");
                break;
            case 6:printf("List1:");
                display(list1);
                printf("List2:");
                display(list2);
                break;
            case 7:exit(0);
                break;
            default:printf("Invalid Choice.... \n");
                break;

        }
    }
}
```

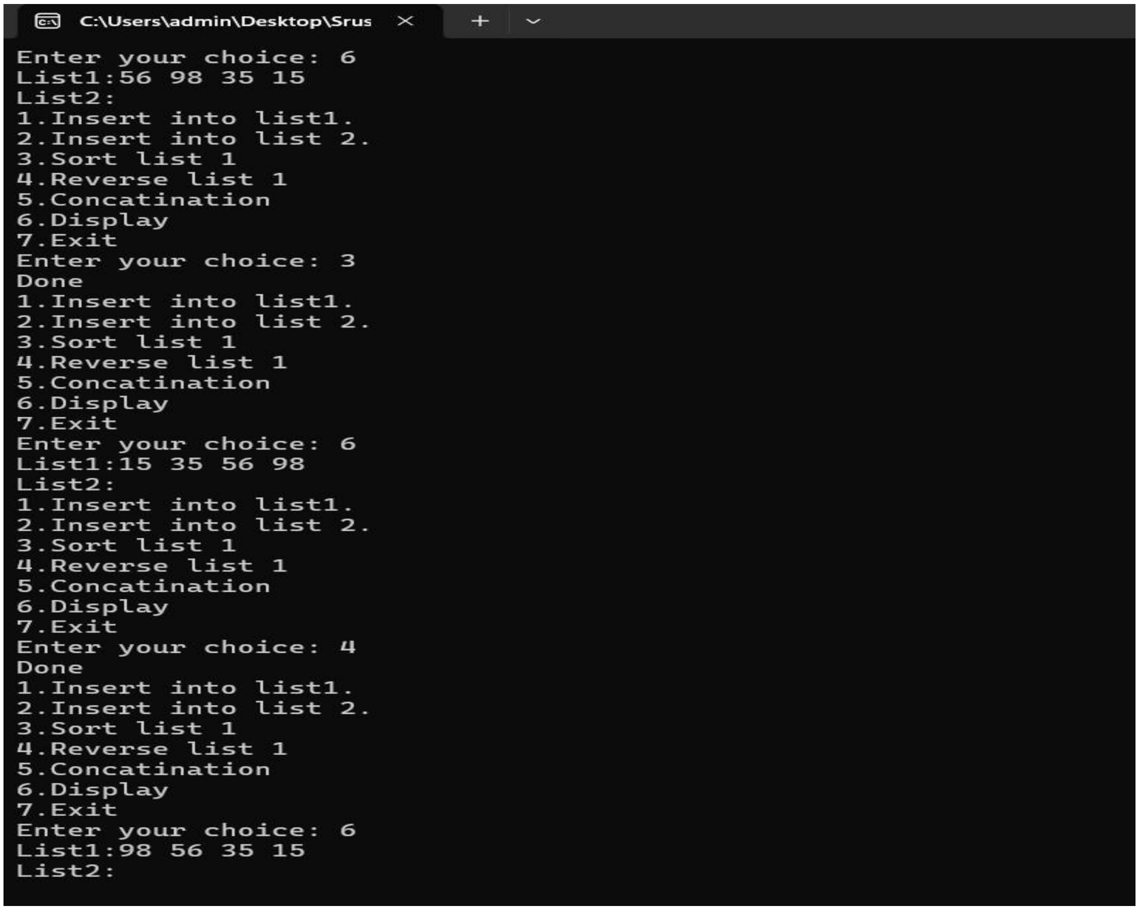
Output:



```
1.Insert into list1.
2.Insert into list 2.
3.Sort list 1
4.Reverse list 1
5.Concatination
6.Display
7.Exit
Enter your choice: 1
enter data to insert into list1:
56
1.Insert into list1.
2.Insert into list 2.
3.Sort list 1
4.Reverse list 1
5.Concatination
6.Display
7.Exit
Enter your choice: 1
enter data to insert into list1:
98
1.Insert into list1.
2.Insert into list 2.
3.Sort list 1
4.Reverse list 1
5.Concatination
6.Display
7.Exit
Enter your choice: 2
enter data to insert into list2:
35
1.Insert into list1.
2.Insert into list 2.
3.Sort list 1
4.Reverse list 1
5.Concatination
6.Display
7.Exit
Enter your choice: 2
enter data to insert into list2:
15
1.Insert into list1.
```



```
1.Insert into list1.
2.Insert into list 2.
3.Sort list 1
4.Reverse list 1
5.Concatination
6.Display
7.Exit
Enter your choice: 6
List1:56 98
List2:35 15
1.Insert into list1.
2.Insert into list 2.
3.Sort list 1
4.Reverse list 1
5.Concatination
6.Display
7.Exit
Enter your choice: 3
Done
1.Insert into list1.
2.Insert into list 2.
3.Sort list 1
4.Reverse list 1
5.Concatination
6.Display
7.Exit
Enter your choice: 6
List1:56 98
List2:35 15
1.Insert into list1.
2.Insert into list 2.
3.Sort list 1
4.Reverse list 1
5.Concatination
6.Display
7.Exit
Enter your choice: 5
Done
1.Insert into list1.
2.Insert into list 2.
3.Sort list 1
```



```
Enter your choice: 6
List1:56 98 35 15
List2:
1.Insert into list1.
2.Insert into list 2.
3.Sort list 1
4.Reverse list 1
5.Concatination
6.Display
7.Exit
Enter your choice: 3
Done
1.Insert into list1.
2.Insert into list 2.
3.Sort list 1
4.Reverse list 1
5.Concatination
6.Display
7.Exit
Enter your choice: 6
List1:15 35 56 98
List2:
1.Insert into list1.
2.Insert into list 2.
3.Sort list 1
4.Reverse list 1
5.Concatination
6.Display
7.Exit
Enter your choice: 4
Done
1.Insert into list1.
2.Insert into list 2.
3.Sort list 1
4.Reverse list 1
5.Concatination
6.Display
7.Exit
Enter your choice: 6
List1:98 56 35 15
List2:
```

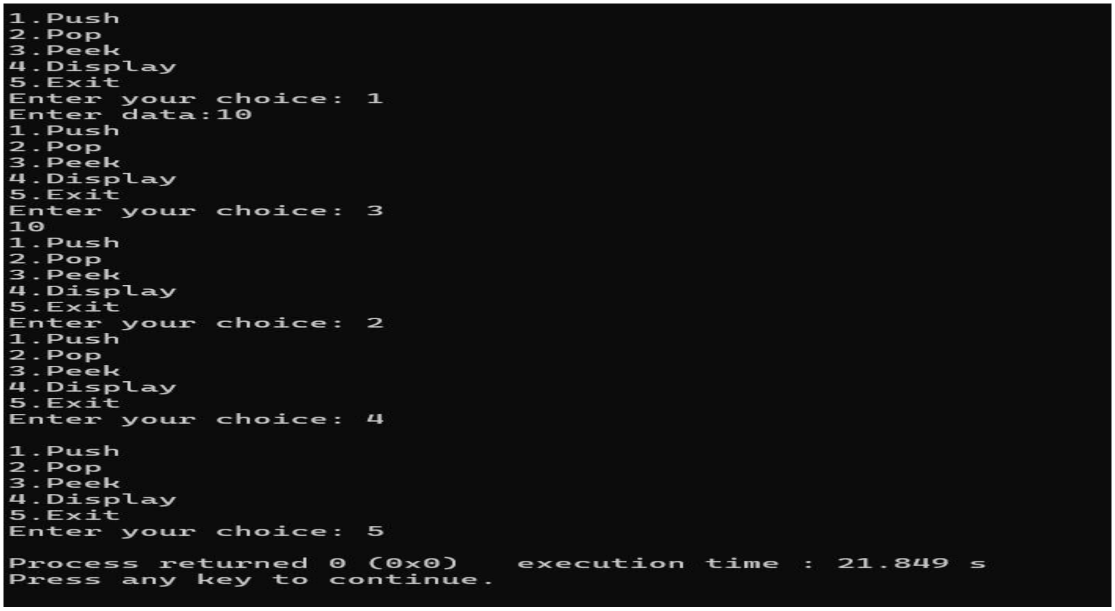
Lab program 6b:

WAP to Implement Single Link List to simulate Stack Operation.

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

struct Node{ int
   data;
   struct Node *next;
};

void push(int x,struct Node* *top){ struct Node newnode=(struct
   Node)malloc(sizeof(struct      Node));      newnode->data=x;
   newnode->next=*top;
   *top=newnode;
}

void pop(struct Node* *top){
   if(*top==NULL){
   printf("Empty\n");
   }
   struct Node *temp=*top;
   *top=(*top)->next;
   free(temp);
}
void peek(struct Node* top) {
   if   (top   ==   NULL)   {
   printf("Empty\n");
   } else { printf("%d \n", top-
      >data);
   }
}

void display(struct Node* top) {
   struct  Node*  temp  =  top;
   while  (temp  !=  NULL)  {
   printf("%d  ",  temp->data);
   temp = temp->next;
   }
   printf("\n");
}

void      main(){      int
   data,choice;      struct
   Node *top=NULL;
```

```
    while(true){
      printf("1.Push\n");
      printf("2.Pop\n");
      printf("3.Peek\n");
      printf("4.Display\n");
      printf("5.Exit\n");

      printf("Enter your choice: ");
      scanf("%d",&choice);

      switch(choice){
        case 1:printf("Enter data:");
            scanf("%d",&data
            );
            push(data,&top);
            break;
        case 2:pop(&top); break;
        case 3:peek(top); break;
        case 4:display(top);
            break;
        case 5:exit(0);
            break;
        default:printf("Invalid Choice\n");
             break;
      }
    }
  }
```

Output:



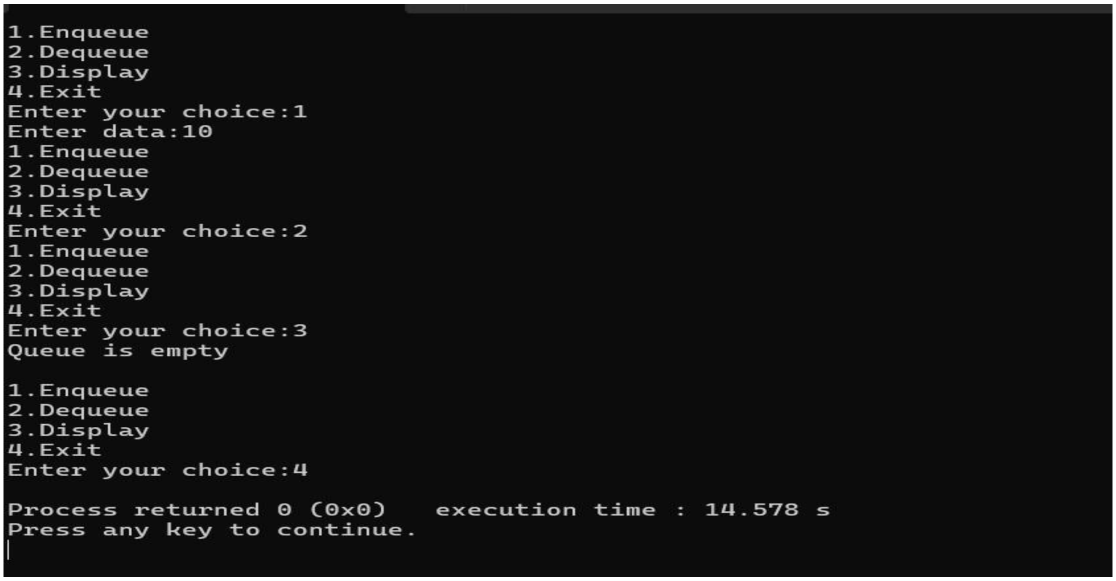
```
1.Push
2.Pop
3.Peek
4.Display
5.Exit
Enter your choice: 1
Enter data:10
1.Push
2.Pop
3.Peek
4.Display
5.Exit
Enter your choice: 3
10
1.Push
2.Pop
3.Peek
4.Display
5.Exit
Enter your choice: 2
1.Push
2.Pop
3.Peek
4.Display
5.Exit
Enter your choice: 4

1.Push
2.Pop
3.Peek
4.Display
5.Exit
Enter your choice: 5

Process returned 0 (0x0)   execution time : 21.849 s
Press any key to continue.
```

WAP to Implement Single Link List to simulate Queue Operation.

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

struct Node{ int
  data;
  struct Node *next;
};

void enqueue(struct Node* *front,struct Node* *rear,int data){
  struct Node *newnode=(struct Node*)malloc(sizeof(struct Node));
  newnode->data=data; newnode->next=NULL;

  if(*front==NULL && *rear==NULL){
    *front=*rear=newnode;
  }
  else{
    (*rear)->next=newnode;
    *rear=newnode;
  }
}

void dequeue(struct Node* *front,struct Node* *rear){
  if(*front==NULL && *rear==NULL){ printf("Queue is
  empty\n");
  }
  else if(*front==*rear){
    *front=*rear=NULL;
  }
  else{
    *front=(*front)->next;
  }
}

void display(struct Node* front){
  struct Node *temp=front;
  if(front==NULL){
  printf("Queue is empty\n");
  }
  while(temp!=NULL){
    printf("%d ",temp->data);
    temp=temp->next;
  }
  printf("\n");
}
```

```
void main(){
   int    data,choice;    struct
   Node           *front=NULL;
   struct Node *rear=NULL;

   while(1){
      printf("1.Enqueue\n");
      printf("2.Dequeue\n");
      printf("3.Display\n");
      printf("4.Exit\n");

      printf("Enter                  your                  choice:");
      scanf("%d",&choice);

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

Output:



```
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice:1
Enter data:10
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice:2
1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice:3
Queue is empty

1.Enqueue
2.Dequeue
3.Display
4.Exit
Enter your choice:4

Process returned 0 (0x0)   execution time : 14.578 s
Press any key to continue.
```

Lab program 7a:

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.

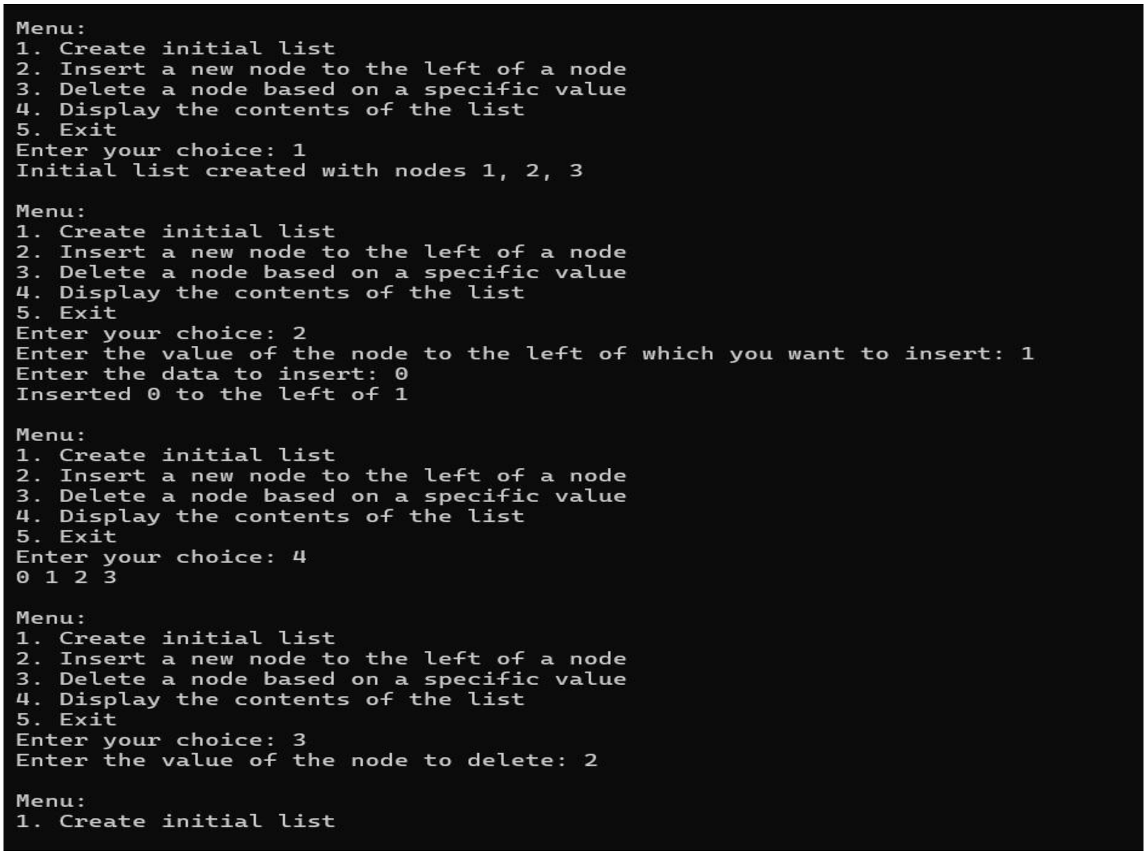
d) Display the contents of the list.

```
#include    <stdio.h>
#include    <stdlib.h>
struct   Node   {   int
data;   struct   Node
*prev;
   struct Node *next;
};
struct Node* createNode(int data) { struct Node *newNode = (struct
   Node *)malloc(sizeof(struct Node)); if (!newNode) { printf("Memory
   allocation failed\n"); exit(1);
   }
   newNode->data = data;
   newNode->prev = NULL;
   newNode->next = NULL;
   return newNode;
}
void insertLeft(struct Node **head, struct Node *node, int data) {
   struct Node *newNode = createNode(data); newNode->next =
   node; newNode->prev = node->prev; if (node->prev != NULL) {
   node->prev->next = newNode;
   } else {
      *head = newNode;
   }
   node->prev = newNode;
}
void deleteNode(struct Node **head, int value) { struct
   Node *temp = *head;
   while (temp != NULL && temp->data != value) { temp
      = temp->next;
   }
   if (temp == NULL) { printf("Node with value %d not
      found\n", value); return;
   }
```

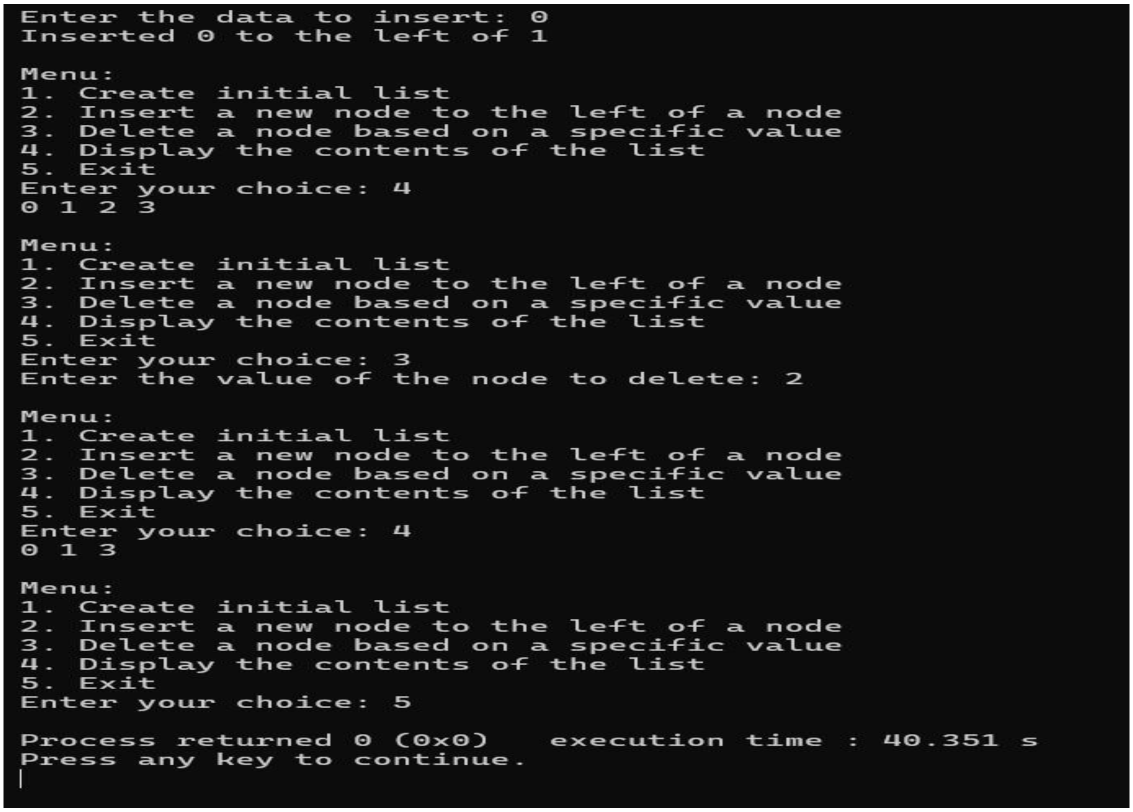
```
  if (temp->prev != NULL) { temp->prev->next
    = temp->next;
  } else {
    *head = temp->next;
  }
  if (temp->next != NULL) { temp->next->prev
    = temp->prev;
  }
  free(temp);
}
void displayList(struct Node *head) {
  struct Node *temp = head; while
  (temp != NULL) { printf("%d ",
  temp->data);
    temp = temp->next;
  }
  printf("\n");
}
int main() { struct Node *head = NULL;
  struct Node *node1, *node2, *node3;
  int choice, data, value; while (1) {
  printf("\nMenu:\n");    printf("1.
  Create initial list\n");
    printf("2. Insert a new node to the left of a node\n");
    printf("3. Delete a node based on a specific value\n");
    printf("4. Display the contents of the list\n");
    printf("5. Exit\n"); printf("Enter your choice: ");
    scanf("%d", &choice); switch (choice) { case 1:node1 =
    createNode(1); node2 = createNode(2); node3 =
    createNode(3); head = node1; node1->next = node2;
    node2->prev = node1; node2->next = node3; node3-
    >prev = node2;
          printf("Initial list created with nodes 1, 2, 3\n"); break;

      case 2:printf("Enter the value of the node to the left of which you want to insert: ");
          scanf("%d", &value);
          printf("Enter the data to insert: ");
```

```
                scanf("%d", &data); struct Node *temp = head;
                while (temp != NULL && temp->data != value) {
                 temp = temp->next;
                }

              if (temp != NULL) { insertLeft(&head,
                 temp, data);
                 printf("Inserted %d to the left of %d\n", data, value);
              } else {
                 printf("Node with value %d not found\n", value);
              }
              break;

           case 3:
              printf("Enter the value of the node to delete: ");
              scanf("%d",      &value);      deleteNode(&head,
              value); break;

           case 4:
              displayList(head);
              break;

           case 5:
              exit(0);
              break;

           default: printf("Invalid
              choice\n"); break;
        }
     }

     return 0;
  }
```

Output:



```
Menu:
1. Create initial list
2. Insert a new node to the left of a node
3. Delete a node based on a specific value
4. Display the contents of the list
5. Exit
Enter your choice: 1
Initial list created with nodes 1, 2, 3

Menu:
1. Create initial list
2. Insert a new node to the left of a node
3. Delete a node based on a specific value
4. Display the contents of the list
5. Exit
Enter your choice: 2
Enter the value of the node to the left of which you want to insert: 1
Enter the data to insert: 0
Inserted 0 to the left of 1

Menu:
1. Create initial list
2. Insert a new node to the left of a node
3. Delete a node based on a specific value
4. Display the contents of the list
5. Exit
Enter your choice: 4
0 1 2 3

Menu:
1. Create initial list
2. Insert a new node to the left of a node
3. Delete a node based on a specific value
4. Display the contents of the list
5. Exit
Enter your choice: 3
Enter the value of the node to delete: 2

Menu:
1. Create initial list
```



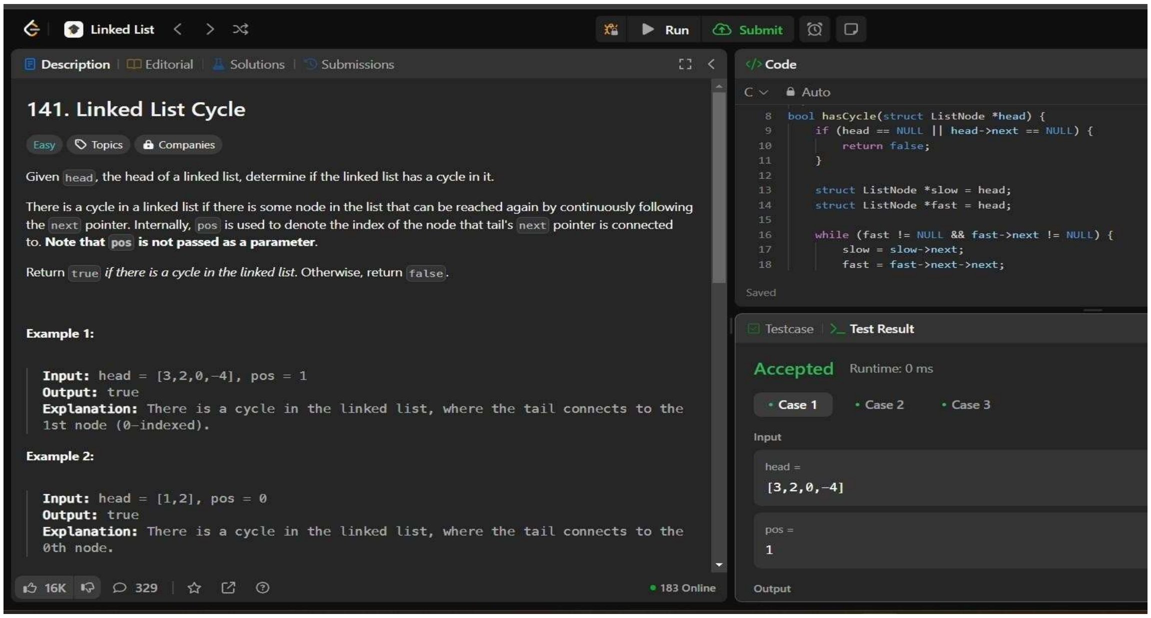
```
Enter the data to insert: 0
Inserted 0 to the left of 1

Menu:
1. Create initial list
2. Insert a new node to the left of a node
3. Delete a node based on a specific value
4. Display the contents of the list
5. Exit
Enter your choice: 4
0 1 2 3

Menu:
1. Create initial list
2. Insert a new node to the left of a node
3. Delete a node based on a specific value
4. Display the contents of the list
5. Exit
Enter your choice: 3
Enter the value of the node to delete: 2

Menu:
1. Create initial list
2. Insert a new node to the left of a node
3. Delete a node based on a specific value
4. Display the contents of the list
5. Exit
Enter your choice: 4
0 1 3

Menu:
1. Create initial list
2. Insert a new node to the left of a node
3. Delete a node based on a specific value
4. Display the contents of the list
5. Exit
Enter your choice: 5

Process returned 0 (0x0)   execution time : 40.351 s
Press any key to continue.
```

Lab program 7b:

Program - Leetcode platform

```
bool hasCycle(struct ListNode *head) { if (head ==
    NULL || head->next == NULL) { return
    false;
    }
    struct ListNode *slow = head; struct
    ListNode *fast = head;
    while (fast != NULL && fast->next != NULL) {
        slow = slow->next; fast
        = fast->next->next; if
        (slow == fast) {
            return true;
        }
    }
    return false;
}
struct ListNode* createNode(int val) { struct ListNode* newNode = (struct
    ListNode*)malloc(sizeof(struct ListNode)); newNode->val = val; newNode-
    >next = NULL; return newNode;
}
struct ListNode* createLinkedListWithCycle(int arr[], int size, int pos) {
    if (size == 0) return NULL;
    struct ListNode* head = createNode(arr[0]); struct
    ListNode* current = head;
    struct ListNode* cycleNode = NULL;
    for (int i = 1; i < size; i++) { current->next =
        createNode(arr[i]);     current     =
        current->next; if (i == pos) {
            cycleNode = current;
        }
    }
    if (cycleNode != NULL) { current->next
    = cycleNode;
} return head;
}
```

Output:



141. Linked List Cycle

Given `head`, the head of a linked list, determine if the linked list has a cycle in it.

There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the `next` pointer. Internally, `pos` is used to denote the index of the node that tail's `next` pointer is connected to. **Note that `pos` is not passed as a parameter.**

Return `true` *if there is a cycle in the linked list*. Otherwise, return `false`.

Example 1:

```
Input: head = [3,2,0,-4], pos = 1
Output: true
Explanation: There is a cycle in the linked list, where the tail connects to the
1st node (0-indexed).
```

Example 2:

```
Input: head = [1,2], pos = 0
Output: true
Explanation: There is a cycle in the linked list, where the tail connects to the
0th node.
```

```c
bool hasCycle(struct ListNode *head) {
    if (head == NULL || head->next == NULL) {
        return false;
    }

    struct ListNode *slow = head;
    struct ListNode *fast = head;

    while (fast != NULL && fast->next != NULL) {
        slow = slow->next;
        fast = fast->next->next;
```

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

head =
[3,2,0,-4]

pos =
1

Lab program 8a:

Write a program

a) ToconstructabinarySearchtree.

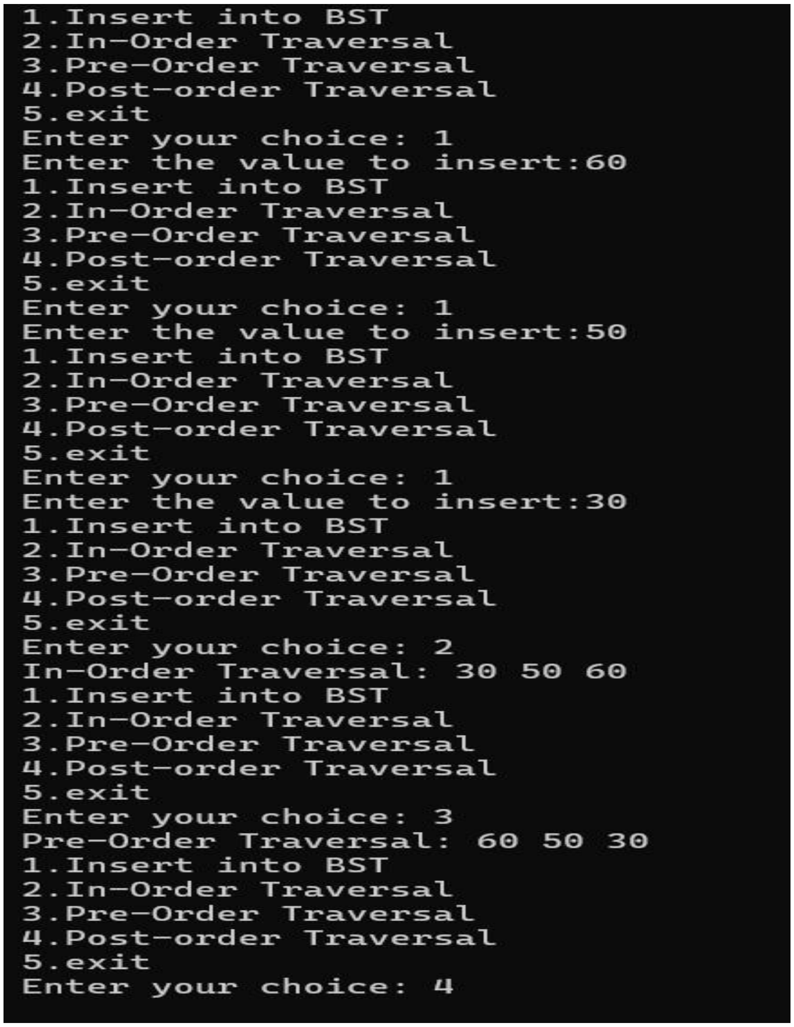
b) To traverse the tree using all the methods i.e., in-order, preorder and post order

c) To display the elements in the tree.

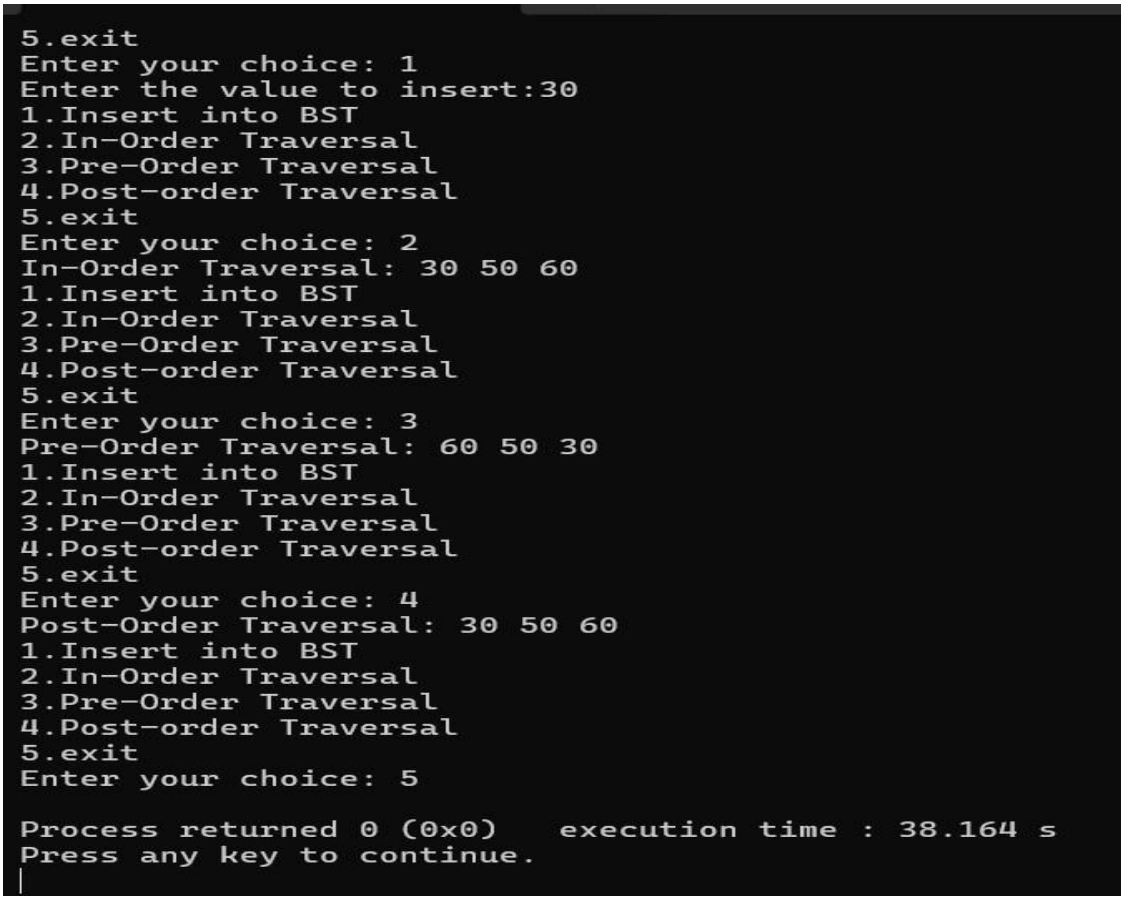
```c
#include<stdio.h>
#include<stdlib.h>

struct   Node{   int   data;
   struct Node *left,*right;
};

struct Node *createNode(int data){
   struct Node *newnode=(struct Node*)malloc(sizeof(struct Node)); newnode->data=data;
   newnode->left=newnode->right=NULL;
   return newnode;
};

struct Node*insert(struct Node *root,int data){
   if(root==NULL){
      return createNode(data);
   }
   if(data<root-data){
      root->left=insert(root->left,data);
    }
    else if(data>root->data){ root->right=insert(root-
      >right,data);
    }
    return root;
  };
  void inOrder(struct Node*root){
    if(root!=NULL){ inOrder(root-
    >left); printf("%d ",root->data);
      inOrder(root->right);
    }
  }
  void preOrder(struct Node
    *root){ if(root!=NULL){
    printf("%d ",root->data);
    preOrder(root->left);
      preOrder(root->right);
    }
  }
  void postOrder(struct Node *root){
    if(root!=NULL){ postOrder(root-
    >left); postOrder(root->right);
      printf("%d ",root->data);
    }
  }

  void  main(){  struct  Node
    *root=NULL;
    int choice,data; while(1){ printf("1.Insert into BST \n2.In-Order Traversal \n3.Pre-Order
    Traversal \n4.Post-order
  Traversal                  \n5.exit\n");
      printf("Enter your choice: ");
      scanf("%d",&choice);
switch(choice){

        case 1:printf("Enter the value to insert:");
            scanf("%d",&data);
            root=insert(root,data); break;
        case 2:printf("In-Order Traversal: "); inOrder(root);
```

```
                printf("\n");
                break;
            case 3:printf("Pre-Order Traversal: ");
                preOrder(root);
                printf("\n");
                break;
            case 4:printf("Post-Order Traversal: ");
                postOrder(root);
                printf("\n");
                break;
            case 5:exit(0);
                break;
            default:printf("Invalid choice...\n");
        }
    }
}
```

Output:



```
1.Insert into BST
2.In-Order Traversal
3.Pre-Order Traversal
4.Post-order Traversal
5.exit
Enter your choice: 1
Enter the value to insert:60
1.Insert into BST
2.In-Order Traversal
3.Pre-Order Traversal
4.Post-order Traversal
5.exit
Enter your choice: 1
Enter the value to insert:50
1.Insert into BST
2.In-Order Traversal
3.Pre-Order Traversal
4.Post-order Traversal
5.exit
Enter your choice: 1
Enter the value to insert:30
1.Insert into BST
2.In-Order Traversal
3.Pre-Order Traversal
4.Post-order Traversal
5.exit
Enter your choice: 2
In-Order Traversal: 30 50 60
1.Insert into BST
2.In-Order Traversal
3.Pre-Order Traversal
4.Post-order Traversal
5.exit
Enter your choice: 3
Pre-Order Traversal: 60 50 30
1.Insert into BST
2.In-Order Traversal
3.Pre-Order Traversal
4.Post-order Traversal
5.exit
Enter your choice: 4
```



```
5.exit
Enter your choice: 1
Enter the value to insert:30
1.Insert into BST
2.In-Order Traversal
3.Pre-Order Traversal
4.Post-order Traversal
5.exit
Enter your choice: 2
In-Order Traversal: 30 50 60
1.Insert into BST
2.In-Order Traversal
3.Pre-Order Traversal
4.Post-order Traversal
5.exit
Enter your choice: 3
Pre-Order Traversal: 60 50 30
1.Insert into BST
2.In-Order Traversal
3.Pre-Order Traversal
4.Post-order Traversal
5.exit
Enter your choice: 4
Post-Order Traversal: 30 50 60
1.Insert into BST
2.In-Order Traversal
3.Pre-Order Traversal
4.Post-order Traversal
5.exit
Enter your choice: 5

Process returned 0 (0x0)   execution time : 38.164 s
Press any key to continue.
```

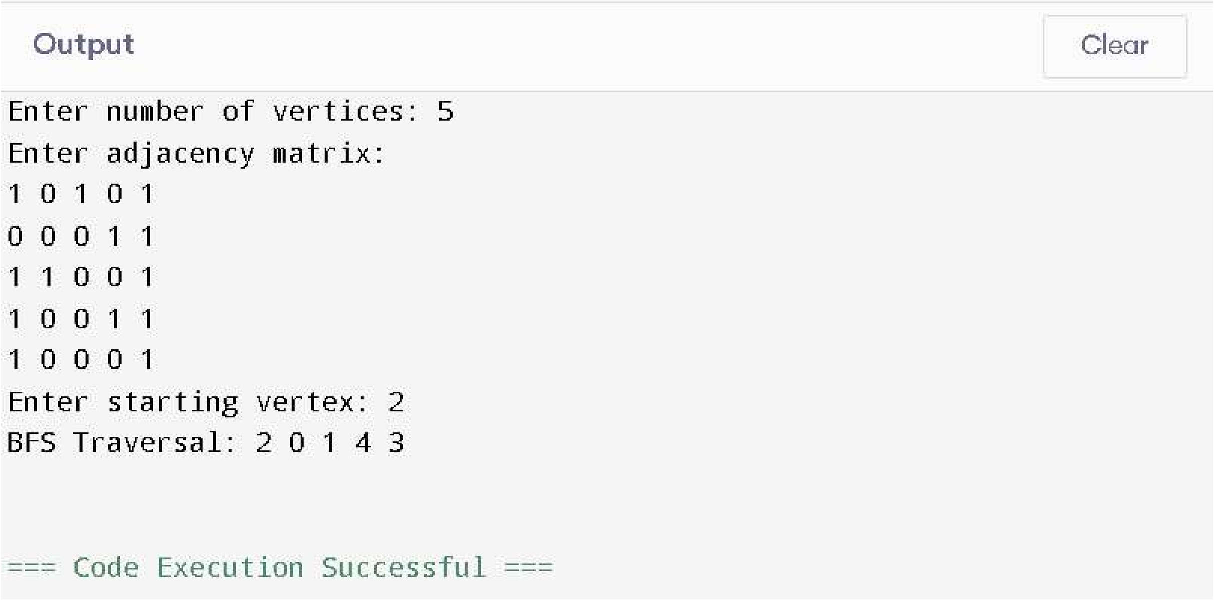
Lab program 9:

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

```
#include <stdio.h>

#define MAX 100

int graph[MAX][MAX], visited[MAX], queue[MAX]; int
front = 0, rear = 0;

// BFS Function void
BFS(int start, int n) {
printf("BFS Traversal: ");
visited[start] = 1;
queue[rear++] = start;

   while (front < rear) {        int
current = queue[front++];
printf("%d ", current);

      for (int i = 0; i < n; i++) {           if
(graph[current][i] == 1 && !visited[i]) {
visited[i] = 1;              queue[rear++] = i;
         }
      }
   }
   printf("\n");
}

int main() {
int n, start;

   printf("Enter number 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",
&graph[i][j]);
      }
   }

   printf("Enter starting vertex: ");
scanf("%d", &start);
```

```
    BFS(start, n);

    return 0;
}
```

Output:



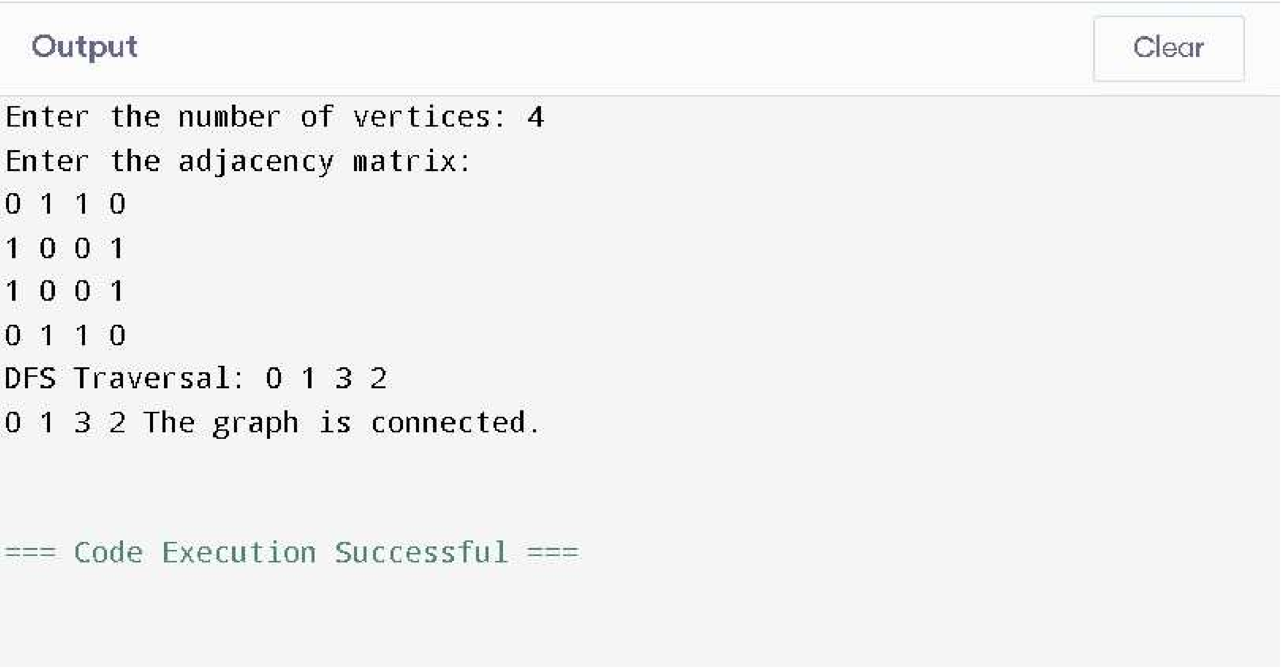
```
Enter number of vertices: 5
Enter adjacency matrix:
1 0 1 0 1
0 0 0 1 1
1 1 0 0 1
1 0 0 1 1
1 0 0 0 1
Enter starting vertex: 2
BFS Traversal: 2 0 1 4 3


=== Code Execution Successful ===
```

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

```
#include <stdio.h>

#define MAX 100

int graph[MAX][MAX], visited[MAX];

// DFS Function void
DFS(int start, int n) {
   printf("%d ", start); // Print the current vertex
   visited[start] = 1;   // Mark the current vertex as visited

   for (int i = 0; i < n; i++) {
      if (graph[start][i] == 1 && !visited[i]) {
         DFS(i, n); // Recursively visit connected vertices
      }
   }
}

// Check Connectivity int
isConnected(int n) {     //
Reset visited array to 0
for (int i = 0; i < n; i++) {
      visited[i] = 0;
```

```
    }

    // Perform DFS starting from vertex 0
    DFS(0, n);

    // Check if all vertices were visited
    for (int i = 0; i < n; i++) {
if (!visited[i]) {
            return 0; // If any vertex is not visited, the graph is not connected
        }
    }
    return 1; // All vertices are visited, graph is connected
}

int main() {
int n;

    printf("Enter the number of vertices: ");
scanf("%d", &n);

    printf("Enter the adjacency matrix:\n");
for (int i = 0; i < n; i++) {         for (int j = 0; j
< n; j++) {
            scanf("%d", &graph[i][j]);
        }
    }

    // Reset visited array before traversal
    for (int i = 0; i < n; i++) {
        visited[i] = 0;
    }

    printf("DFS Traversal: ");
    DFS(0, n); // Perform DFS traversal from vertex 0
printf("\n");

    if (isConnected(n)) {
        printf("The graph is connected.\n");
    } else {
        printf("The graph is not connected.\n");
    }
    return 0;
}
```

Output:



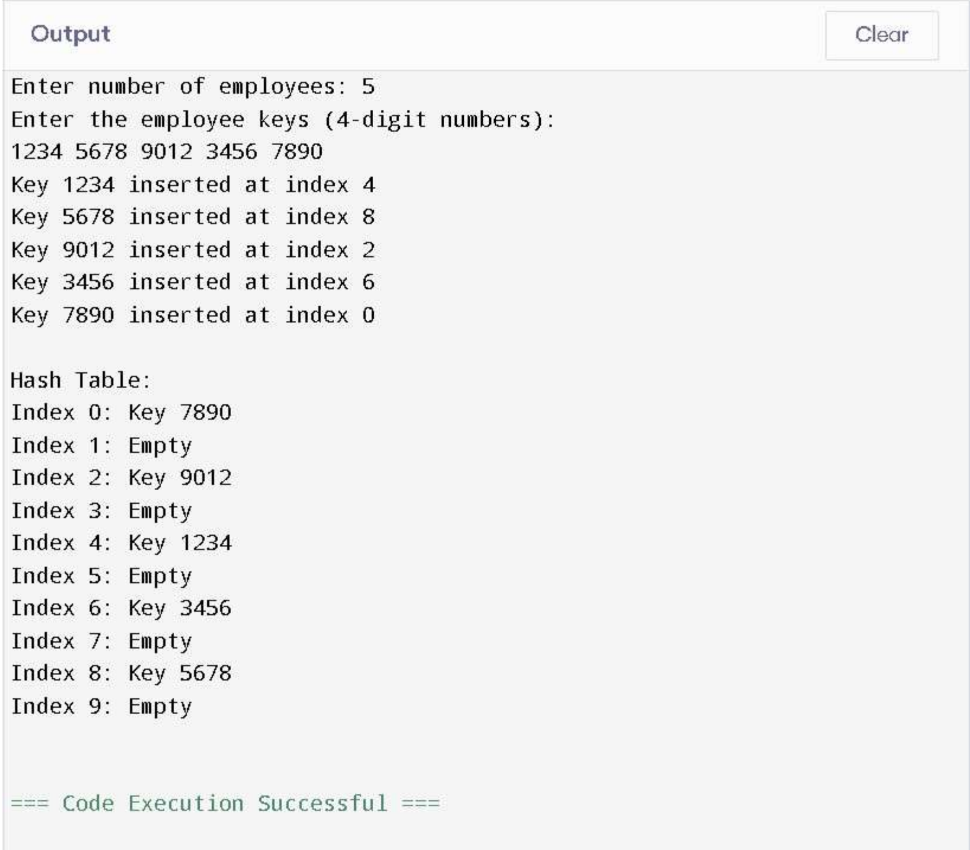
```
Enter the number of vertices: 4
Enter the adjacency matrix:
0 1 1 0
1 0 0 1
1 0 0 1
0 1 1 0
DFS Traversal: 0 1 3 2
0 1 3 2 The graph is connected.


=== Code Execution Successful ===
```

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 -> 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 MAX_EMPLOYEES 100 // Maximum number of employees
#define MAX_KEYS 100      // Maximum number of keys
#define TABLE_SIZE 10     // Size of hash table (m)

typedef struct {
    int key;
    // You can add other employee details here
    // For simplicity, we use just the key
} Employee;

int hashTable[TABLE_SIZE]; // Hash table to store keys
```

```
Employee employees[MAX_EMPLOYEES]; // Array to store employee records int
N; // Number of employees
int m = TABLE_SIZE; // Size of the hash table

// Hash function: H(K) = K mod m int
hashFunction(int key) {
  return key % m;
}

// Linear Probing to resolve collisions
int linearProbing(int key) {    int
index = hashFunction(key);
  int originalIndex = index; // Store original index to detect full table

  while (hashTable[index] != -1) {
if (hashTable[index] == key) {
      return index; // Key already exists (no insertion needed)
    }
    // Linear probing: move to the next index
    index = (index + 1) % m;

    // If we have checked all positions, return -1 indicating table is full
if (index == originalIndex) {
      return -1;
    }
  }
  return index;
}

// Function to insert a key into the hash table void
insert(int key) {
  int index = linearProbing(key);

  if (index != -1) {
    hashTable[index] = key; // Store the key at the found index
    printf("Key %d inserted at index %d\n", key, index);
  } else {
    printf("Error: Hash table is full. Key %d cannot be inserted.\n", key);
  }
}

// Function to display the hash table
void displayHashTable() {
printf("\nHash Table:\n");    for (int
i = 0; i < m; i++) {      if (hashTable[i]
!= -1) {
      printf("Index %d: Key %d\n", i, hashTable[i]);
    } else {
      printf("Index %d: Empty\n", i);
    }
  }
}

int main() {
    // Initialize hash table to -1 (empty)
    for (int i = 0; i < m; i++) {
        hashTable[i] = -1;
    }

    printf("Enter number of employees: ");
scanf("%d", &N);

    printf("Enter the employee keys (4-digit numbers):\n");
for (int i = 0; i < N; i++) {
        scanf("%d", &employees[i].key); // Read key for each employee
    }

    // Insert the employee keys into the hash table
for (int i = 0; i < N; i++) {
        insert(employees[i].key);
    }

    // Display the final hash table
    displayHashTable();

    return 0;
}
```

Output:



```
Enter number of employees: 5
Enter the employee keys (4-digit numbers):
1234 5678 9012 3456 7890
Key 1234 inserted at index 4
Key 5678 inserted at index 8
Key 9012 inserted at index 2
Key 3456 inserted at index 6
Key 7890 inserted at index 0

Hash Table:
Index 0: Key 7890
Index 1: Empty
Index 2: Key 9012
Index 3: Empty
Index 4: Key 1234
Index 5: Empty
Index 6: Key 3456
Index 7: Empty
Index 8: Key 5678
Index 9: Empty


=== Code Execution Successful ===
```