VISVESVARAYA TECHNOLOGICAL UNIVERSITY
"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Data Structures using C
Submitted by

Sirigireddy Pranav Reddy(1BM22CS281)

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
June-2023 to September-2023

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

CERTIFICATE

This is to certify that the Lab work entitled "**Data Structures using C**" carried out by**Sirigireddy Pranav Reddy(1BM22CS281),** 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 academic semester December-2023 to March-2024. The Lab report has been approved as it satisfies the academic requirements in respect of a **Data Structures using C (23CS3PCDST)** work prescribed for the said degree.

Dr.Madhavi R P	Dr. Jyothi S Nayak
Assistant professor	Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

Index Sheet

7	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 Display the contents of the list 7b) Program - Leetcode platform	56-66
8	8a) 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 To display the elements in the tree. 8b) Program - Leetcode platform	67-71
9	a) Write a program to traverse a graph using BFS method. b) Write a program to check whether given graph is connected or not using DFS method.	72-77
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.	

Course Outcome

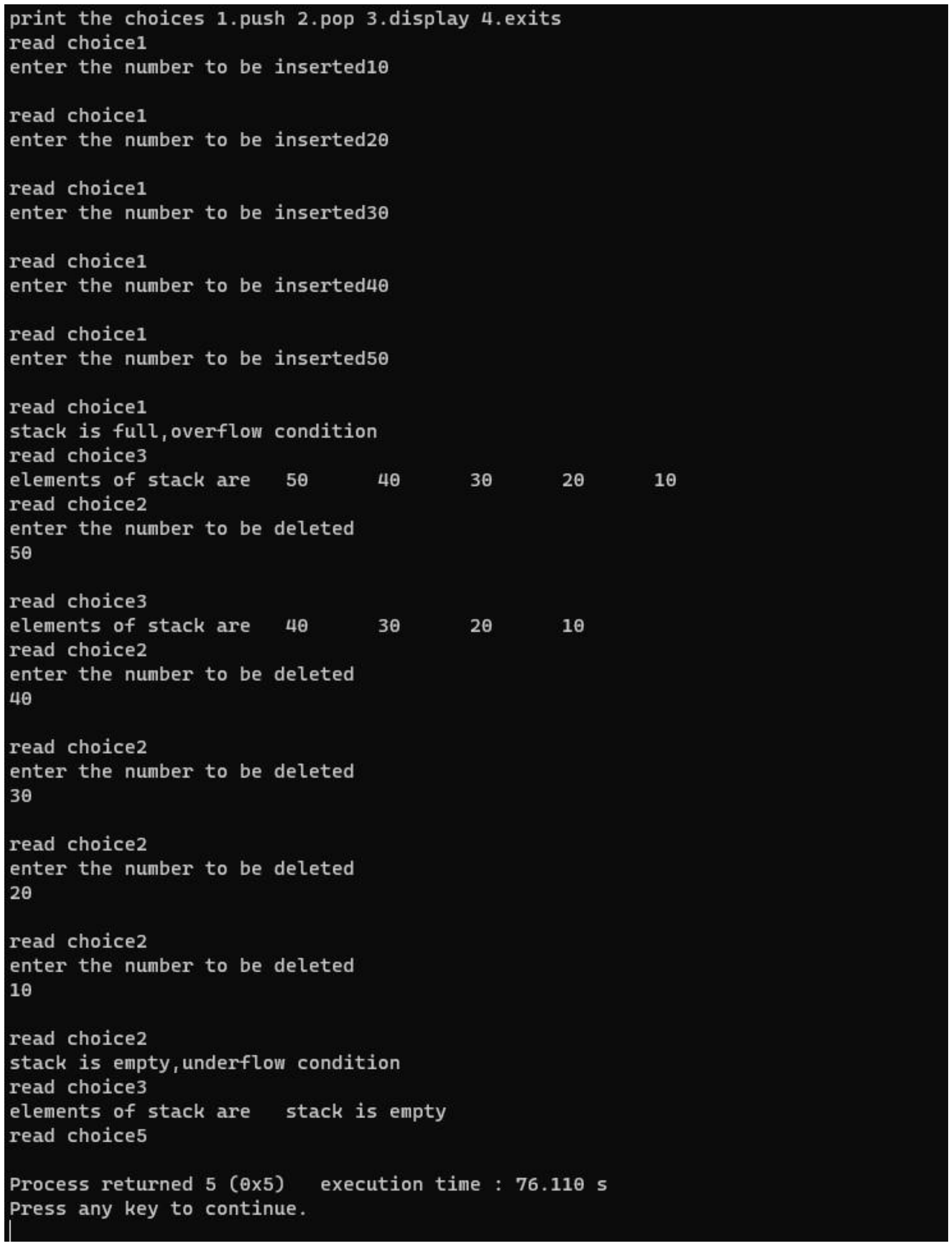
CO1	Apply the concept of linear and nonlinear data structures.

CO2	Analyse data structure operations for a given problem.
CO3	CO3 Design and implement operations of linear and nonlinear data structure.
CO4	Conduct practical experiments for demonstrating the operations of different data structures and sorting techniques.

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 n 5  int
stack[n];  int top=-
1;  void push();
void pop();  void
display();  void
push()
{
        int item;  if(top==n-1)
        {
                printf("stack is full,overflow condition");  return;
        }
        else
        {
                printf("enter the number to be inserted");
                scanf("%d",&item);
                top++;
                stack[top]=item;
        }
}
void pop()
{
        int item;  if(top==-1)
        {
                printf("stack is empty,underflow condition");  return;
        }
        else
        {
                printf("enter the number to be deleted\n");
                scanf("%d",&item);  item=stack[top];
                top=top-1;
        }
}
void display()
{  int i;
        printf("elements of stack are\t");  for(i=top;i>=0;i--)
        {
                printf("%d\t",stack[i]);
         }
        if(top==-1)
        {
                printf("stack is empty");
        }
}
void main()
{
        int choice;
        printf("print the choices 1.push 2.pop 3.display 4.exits\n");
printf("read choice");      scanf("%d",&choice);
        do
        {
                switch(choice)
                {
                        case 1:push();
                        break;  case
                        2:pop();  break;
                        case
                        3:display();
                        break;

                        case                    4:exit(0);
                        default:printf("invalid choice");
                        break;
                }
                printf("\nread choice");  scanf("%d",&choice);
                }while(choice!=5);
        }
```

OUTPUT:



```
print the choices 1.push 2.pop 3.display 4.exits
read choice1
enter the number to be inserted10

read choice1
enter the number to be inserted20

read choice1
enter the number to be inserted30

read choice1
enter the number to be inserted40

read choice1
enter the number to be inserted50

read choice1
stack is full,overflow condition
read choice3
elements of stack are   50      40      30      20      10
read choice2
enter the number to be deleted
50

read choice3
elements of stack are   40      30      20      10
read choice2
enter the number to be deleted
40

read choice2
enter the number to be deleted
30

read choice2
enter the number to be deleted
20

read choice2
enter the number to be deleted
10

read choice2
stack is empty,underflow condition
read choice3
elements of stack are   stack is empty
read choice5

Process returned 5 (0x5)   execution time : 76.110 s
Press any key to continue.
```

2.a) 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).

b) Demonstration of account creation on LeetCode platform Program - Leetcode platform

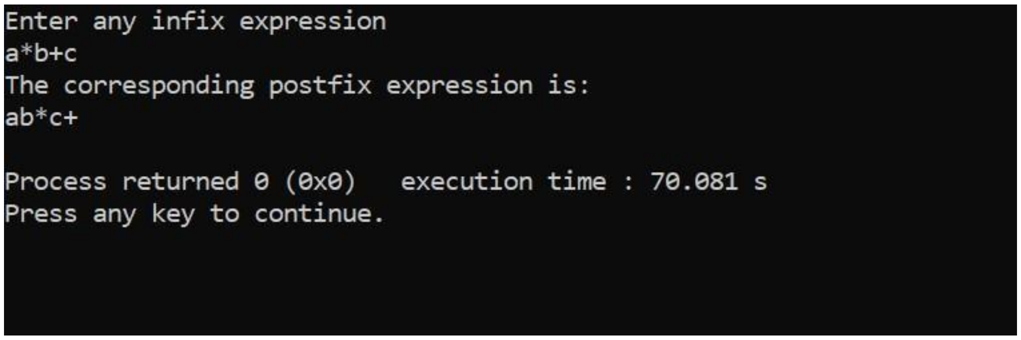
```
 // InfixToPostfix
#include <stdio.h>
#include <conio.h> #include
<ctype.h>
#include <string.h>
#define MAX 100
char st[MAX]; int
top = -1;
void infixtopostfix(char source[], char target[]);
int getpriority(char); void push(char st[], char);
char pop(char st[]); int main()
{
   char infix[100], postfix[100];
printf("Enter any infix expression\n");
gets(infix);     strcpy(postfix, "");
   infixtopostfix(infix, postfix);
   printf("The corresponding postfix expression is:\n");
puts(postfix);     return 0;
}
```

```
int getpriority(char op)
{
   if (op == '/' || op == '*' || op == '%')
return 1;    else if (op == '+' || op == '-
')        return 0;
}
void push(char st[], char val)
{
   if (top == MAX - 1)
printf("Stack overflow\n");    else
   {
      top++;
st[top] = val;
   }
}
char pop(char st[])
{
   char val = ' ';
if (top == -1)
   {
      printf("Stack Underflow\n");
   }
   else
   {
        val = st[top];
top--;
    }
    return val;
}
void infixtopostfix(char source[], char target[])
{     int i = 0, j = 0;     char
temp;     strcpy(target,
"");     while (source[i] !=
'\0')
    {
        if (source[i] == '(')
        {
            push(st, source[i]);
i++;
        }
        else if (source[i] == ')')
        {
            while ((top != -1) && (st[top] != '('))
            {
                target[j] = pop(st);
j++;
            }
            if (top == -1)
            {
```

```
            printf("\n Incorrect Expression");
exit(1);
          }
          temp = pop(st);
i++;
      }
      else if (isdigit(source[i]) || isalpha(source[i]))
      {
          target[j] = source[i];
j++;          i++;
      }
      else if (source[i] == '+' || source[i] == '-' || source[i] == '*' || source[i] == '/' ||
source[i] == '^')
      {
          while ((top != -1) && (st[top] != '(') && (getpriority(st[top]) >
getpriority(source[i])))
          {
            target[j] = pop(st);
j++;
          }
          push(st, source[i]);
i++;      }       else
      {
          printf("\n Incorrect Element in Expression");
exit(1);
```

```
        }
    }
    while ((top != -1) && (st[top] != '('))
    {
        target[j] = pop(st);
j++;
    }
    target[j] = '\0';
}
```

OUTPUT:



```
Enter any infix expression
a*b+c
The corresponding postfix expression is:
ab*c+

Process returned 0 (0x0)   execution time : 70.081 s
Press any key to continue.
```

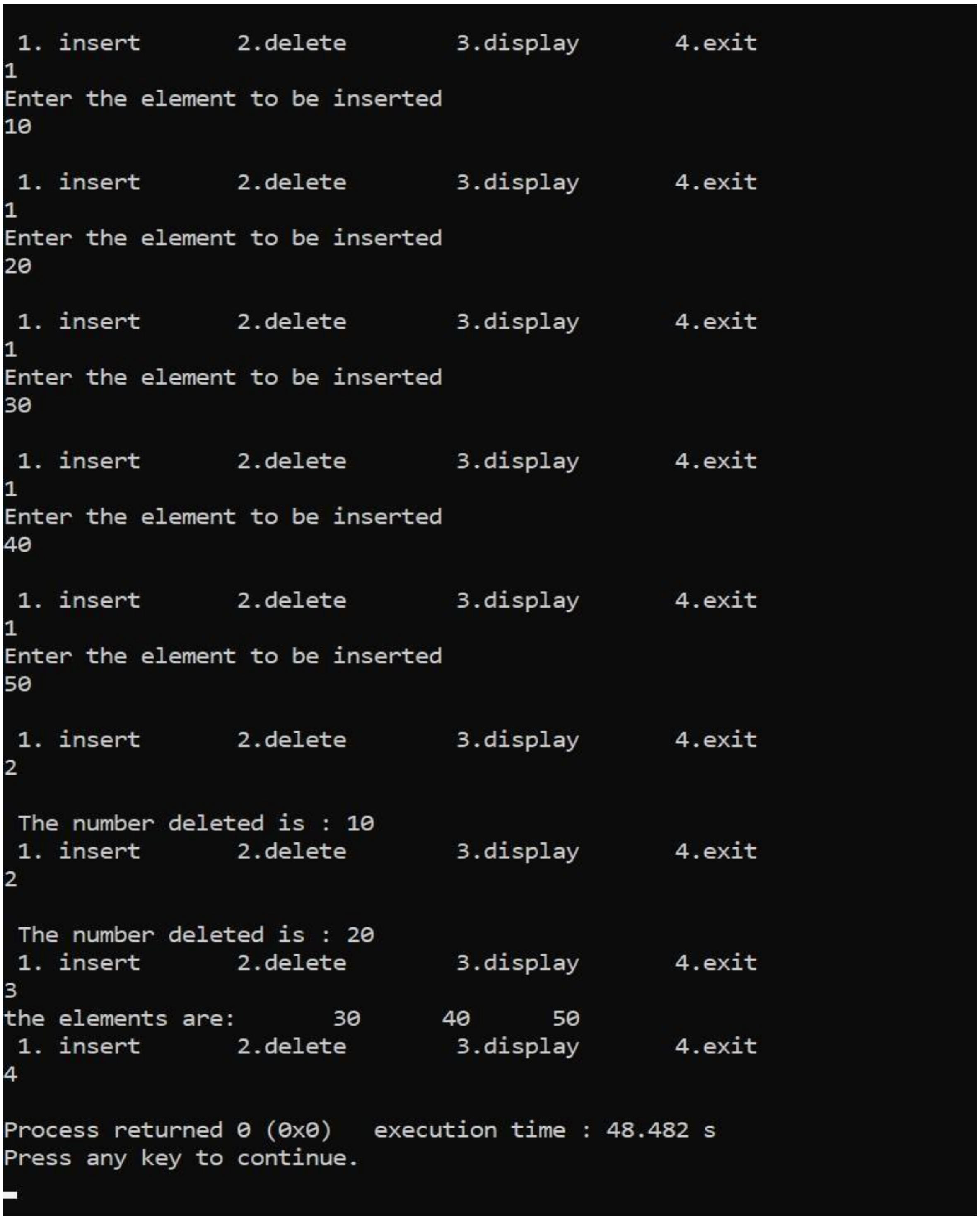
3a) WAP to simulate the working of a queue of integers using an array. Provide the following operations: Insert, Delete, Display the program should print appropriate messages for queue empty and queue overflow conditions.

```
#include <stdio.h>

#include <stdlib.h>

#define MAX 5 int

q[MAX]; int front =

-1; int rear = -1;

void insert(); int

delete_q(); void

display(); int

main()

{    while

(1)

   {

      int choice, d;
```

```
        printf("\n 1. insert \t 2.delete \t 3.display \t 4.exit\n");
scanf("%d", &choice);         switch (choice)
        {
        case 1:
insert();
break;         case 2:

            d = delete_q();
if (d != -1)
                printf("\n The number deleted is:%d",d);
break;         case 3:             display();             break;
case 4:             exit(0);
        }
    }
}
void insert()
{
    if (rear == MAX - 1)
    {
        printf("Queue is Full\n");
return;
    }
    printf("Enter the element to be inserted\n");
int a;
```

```
    scanf("%d", &a);     if ((front ==
-1) && (rear == -1))     {
        front = rear = 0;
    }
    else
    {
        rear++;
    }
    q[rear] = a;
}
int delete_q()
{    int val;     if (front == -1 ||
rear < front)
    {
        printf("Underflow\n");
return -1;
    }
    else
    {
        val = q[front];
front++;        if
(front > rear)
        {
            front = rear = -1;
        }
        return val;
    }
}
void display()
{
    printf("the elements are:\t");
for (int i = front; i <= rear; i++)
    {
        printf("%d \t", q[i]);
    }
}
```

OUTPUT:



```
 1. insert       2.delete        3.display       4.exit
1
Enter the element to be inserted
10

 1. insert       2.delete        3.display       4.exit
1
Enter the element to be inserted
20

 1. insert       2.delete        3.display       4.exit
1
Enter the element to be inserted
30

 1. insert       2.delete        3.display       4.exit
1
Enter the element to be inserted
40

 1. insert       2.delete        3.display       4.exit
1
Enter the element to be inserted
50

 1. insert       2.delete        3.display       4.exit
2

 The number deleted is : 10
 1. insert       2.delete        3.display       4.exit
2

 The number deleted is : 20
 1. insert       2.delete        3.display       4.exit
3
the elements are:       30      40      50
 1. insert       2.delete        3.display       4.exit
4

Process returned 0 (0x0)   execution time : 48.482 s
Press any key to continue.
```

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>

#define MAX 5 int

queue[MAX];

int front = -1, rear = -1;

void insert(); int

delete_element(); int

peek(); void display();

int main()

{

   int option, val;

do

   {

      printf("Enter : 1-Insert, 2-Delete, 3-Peek, 4-Display & 5-Exit : \n");

printf("Enter your option : \n");        scanf("%d", &option);        switch

(option)

      {

      case 1:

insert();

break;        case 2:

         val = delete_element();

if (val != -1)
```

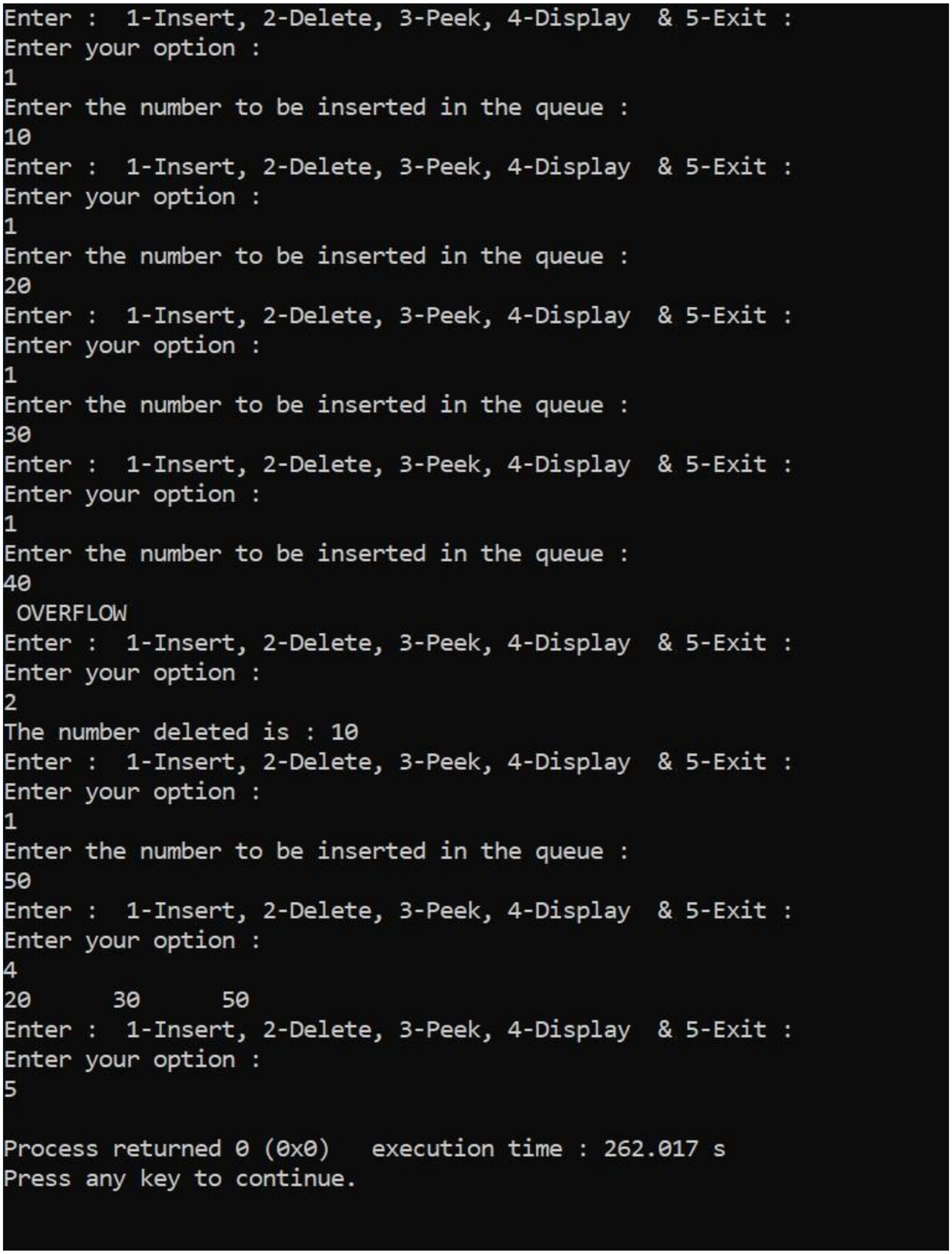
```
                printf("The number deleted is : %d \n", val);
break;          case 3:
            val = peek();
if (val != -1)
                printf("\n The first value in queue is : %d \n", val);
break;          case 4:             display();              break;
        }
    } while (option != 5);
return 0;
}
void insert()
{
    int num;
    printf("Enter the number to be inserted in the queue : \n");
scanf("%d", &num);     if (front == 0 && rear == MAX - 1)
printf(" OVERFLOW \n");
    else if (front == -1 && rear == -1)
    {
        front = rear = 0;
queue[rear] = num;
    }
    else if (rear == MAX - 1 && front != 0)
    {       rear = 0;
queue[rear] = num;
```

```
   }
   else
   {
       rear++;
queue[rear] = num;
   }
}
int delete_element()
{    int val;     if (front == -1 &&
rear == -1)
   {
       printf("UNDERFLOW \n");
return -1;
   }
   val = queue[front];
if (front == rear)
front = rear = -1;
else
   {
       if (front == MAX - 1)
front = 0;         else
front++;
   }
   return val;
}
```

```
int peek()
{
   if (front == -1 && rear == -1)
   {
      printf("QUEUE IS EMPTY \n");
return -1;
   }
   else
   {
      return queue[front];
   }
}
void display()
{    int
i;
   // printf("\n");     if (front == -1
&& rear == -1)
printf("QUEUE IS EMPTY\n");
else
   {
      if (front < rear)
      {
         for (i = front; i <= rear; i++)
printf("%d\t", queue[i]);
```

```
        }
else
        {
            for (i = front; i < MAX; i++)
printf("%d \t", queue[i]);             for (i
= 0; i <= rear; i++)
printf("%d \t ", queue[i]);
        }
        printf("\n");
    }
}
```

OUTPUT:



```
Enter :  1-Insert, 2-Delete, 3-Peek, 4-Display  & 5-Exit :
Enter your option :
1
Enter the number to be inserted in the queue :
10
Enter :  1-Insert, 2-Delete, 3-Peek, 4-Display  & 5-Exit :
Enter your option :
1
Enter the number to be inserted in the queue :
20
Enter :  1-Insert, 2-Delete, 3-Peek, 4-Display  & 5-Exit :
Enter your option :
1
Enter the number to be inserted in the queue :
30
Enter :  1-Insert, 2-Delete, 3-Peek, 4-Display  & 5-Exit :
Enter your option :
1
Enter the number to be inserted in the queue :
40
 OVERFLOW
Enter :  1-Insert, 2-Delete, 3-Peek, 4-Display  & 5-Exit :
Enter your option :
2
The number deleted is : 10
Enter :  1-Insert, 2-Delete, 3-Peek, 4-Display  & 5-Exit :
Enter your option :
1
Enter the number to be inserted in the queue :
50
Enter :  1-Insert, 2-Delete, 3-Peek, 4-Display  & 5-Exit :
Enter your option :
4
20      30      50
Enter :  1-Insert, 2-Delete, 3-Peek, 4-Display  & 5-Exit :
Enter your option :
5

Process returned 0 (0x0)   execution time : 262.017 s
Press any key to continue.
```

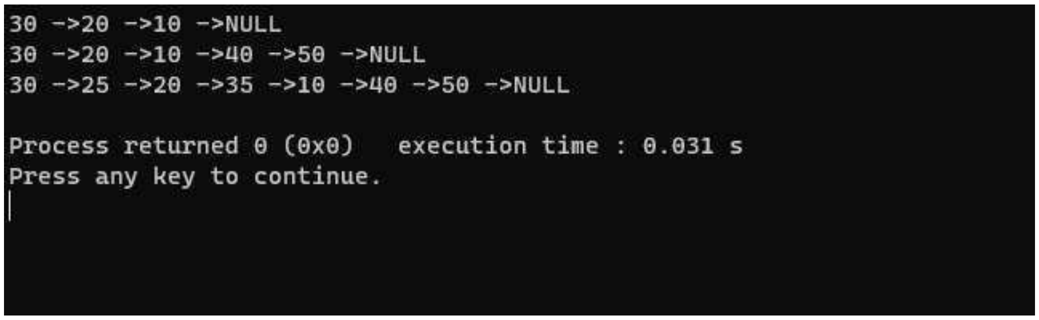
4a) WAP to Implement Singly Linked List with following operations a) Create a linked list.

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> struct
node
{    int data;    struct
node *next;
};
void insertatbegin(struct node **head, int item)
{
   struct node *newnode = (struct node *)malloc(sizeof(struct node));
newnode->next = *head;    newnode->data = item;
   *head = newnode;
}
void insertatend(struct node **head, int item)
{
   struct node *newnode = (struct node *)malloc(sizeof(struct node));
struct node *temp = *head;    newnode->next = NULL;    newnode-
>data = item;
   if (*head == NULL)
   {
        *head = newnode;
return;
    }
    while (temp->next != NULL)
    {
        temp = temp->next;
    }
    temp->next = newnode;
}
void insertatspecific(struct node **head, int item, int loc)
{    if (loc <=
0)
    {
        printf("invalid position\n");
return;
    }
    if (loc == 1 || *head == NULL)
    {
        insertatbegin(head, item);
return;
    }
    struct node *newnode = (struct node *)malloc(sizeof(struct node));
newnode->data = item;
    struct node *temp = *head;
int count = 1;
```

```
   while (count < loc - 1 && temp->next != NULL)
   {
       temp = temp->next;
count++;
   }
   newnode->next = temp->next;     temp-
>next = newnode;
}
void display(struct node *head)
{
   struct node *temp = head;
if (temp == NULL)
   {
       printf("linked list is empty\n");
return;
   }
   while (temp != NULL)
   {
       printf("%d ->", temp->data);
temp = temp->next;
   }
   printf("NULL\n");
}
int main()
{
    struct node *head = NULL;
insertatbegin(&head, 10);
insertatbegin(&head, 20);
insertatbegin(&head, 30);
insertatend(&head, 40);
insertatend(&head, 50);
insertatspecific(&head, 25, 2);
insertatspecific(&head, 35, 4);
display(head);     return 0;
}
```

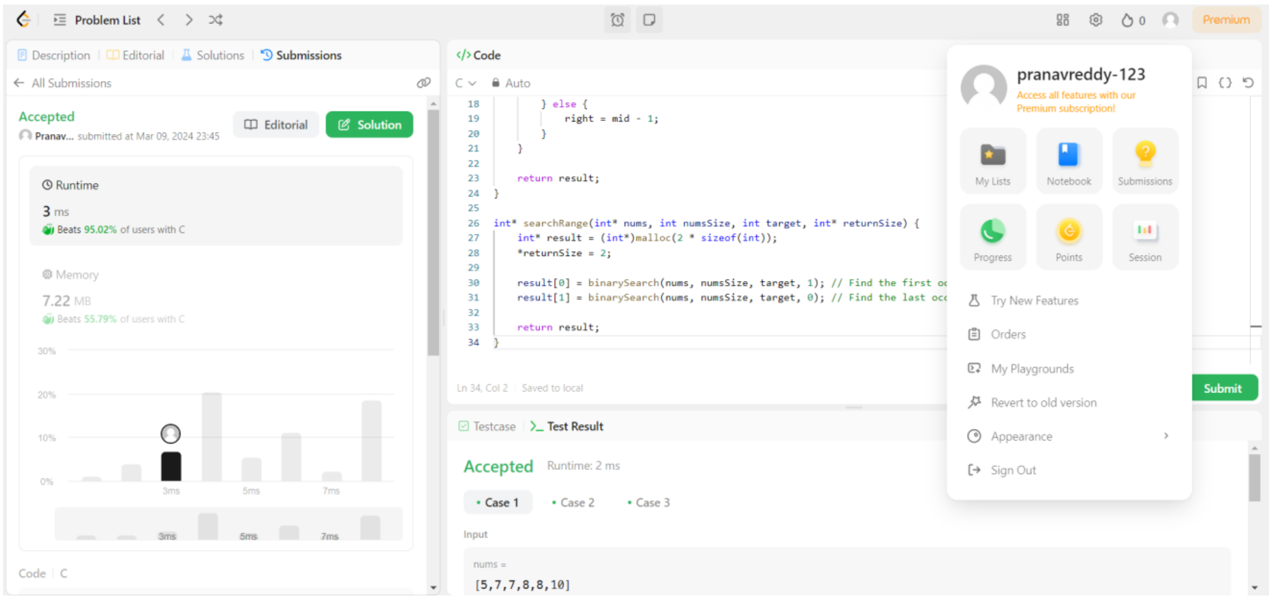
OUTPUT:



```
30 ->20 ->10 ->NULL
30 ->20 ->10 ->40 ->50 ->NULL
30 ->25 ->20 ->35 ->10 ->40 ->50 ->NULL

Process returned 0 (0x0)   execution time : 0.031 s
Press any key to continue.
```

4b) Program - Leetcode platform

OUTPUT:



Accepted
Pranav... submitted at Mar 09, 2024 23:45

Runtime
3 ms
Beats 95.02% of users with C

Memory
7.22 MB
Beats 55.79% of users with C

```
        } else {
            right = mid - 1;
        }
    }

    return result;
}

int* searchRange(int* nums, int numsSize, int target, int* returnSize) {
    int* result = (int*)malloc(2 * sizeof(int));
    *returnSize = 2;

    result[0] = binarySearch(nums, numsSize, target, 1); // Find the first oc
    result[1] = binarySearch(nums, numsSize, target, 0); // Find the last occ

    return result;
}
```

Accepted Runtime: 2 ms

Input

```
nums =
[5,7,7,8,8,10]
```

5a) WAP to Implement Singly Linked List with following operations a) Create a linked list.

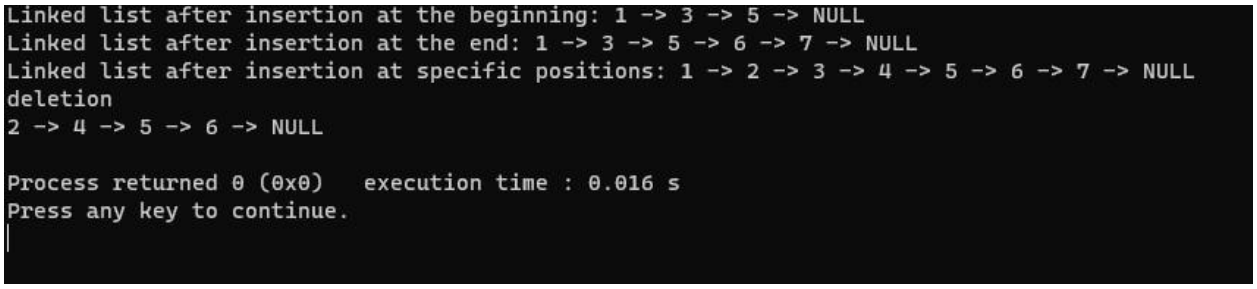
b) Deletion of first element, specified element and last element in the list.Display the contents of the linked list.

```
#include <stdio.h>
#include <stdlib.h> struct
Node
{    int data;     struct
Node *next;
};
void insertAtBeginning(struct Node **head, int value)
{
   struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
newNode->data = value;     newNode->next = *head;
```

```
    *head = newNode;
}
void insertAtEnd(struct Node **head, int value)
{
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
struct Node *temp = *head;     newNode->data = value;     newNode-
>next = NULL;
    if (*head == NULL)
    {
        *head = newNode;
        return;
    }
    while (temp->next != NULL)
    {
        temp = temp->next;
    }
    temp->next = newNode;
}
void insertAtPosition(struct Node **head, int value, int position)
{
    if (position <= 0)
    {
        printf("Invalid position\n");
return;
```

```
   }
   if (position == 1 || *head == NULL)
   {
      insertAtBeginning(head, value);
return;
   }
   struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
                                                            newNode->data = value;
   struct Node *temp = *head;
int count = 1;
   while (count < position - 1 && temp->next != NULL)
   {
      temp = temp->next;
count++;
   }
   if (count < position - 1)
   {
      printf("Invalid position\n");
return;
   }
   newNode->next = temp->next;     temp-
>next = newNode;
}
void deleteAtBegining(struct Node **head)
{
    if (*head == NULL)
    {
        printf("The linkedlist is already empty\n");
return;
    }
    else
    {
        struct Node *first = *head;
*head = (*head)->next;
        free(first);
    }
}
void deleteAtEnd(struct Node **head)
{
    if (*head == NULL)
    {
        printf("The linkedlist is already empty\n");
return;
    }
    else
    {
        struct Node *temp = *head;
while (temp->next->next != NULL)
        {
            temp = temp->next;
        }
        struct Node *lastNode = temp->next;
temp->next = NULL;        free(lastNode);
    }
}
void deleteAtIndex(struct Node **head, int pos)
{
    if (*head == NULL)
    {
        printf("The Linked List is Empty \n");
    }
    else
    {
        struct Node *temp = *head;
        pos--;
        while (pos-- && temp != NULL)
        {
            temp = temp->next;
        }
        if (temp == NULL)
        {
            printf("pos not exist\n");
        }
else
        {
            struct Node *nxt = temp->next->next;
struct Node *del = temp->next;            temp->next
= temp->next->next;            free(del);

        }
    }
}
void displayLinkedList(struct Node *head)
{
    struct Node *temp = head;
if (temp == NULL)
    {
        printf("Linked list is empty.\n");
return;
    }
    while (temp != NULL)
    {
        printf("%d -> ", temp->data);
temp = temp->next;
    }
    printf("NULL\n");
}
int main()
{
    struct Node *head = NULL;
insertAtBeginning(&head, 5);
insertAtBeginning(&head, 3);
insertAtBeginning(&head, 1);
    printf("Linked list after insertion at the beginning: ");
displayLinkedList(head);     insertAtEnd(&head, 6);
insertAtEnd(&head, 7);
    printf("Linked list after insertion at the end: ");
displayLinkedList(head);     insertAtPosition(&head,
2, 2);     insertAtPosition(&head, 4, 4);
    printf("Linked list after insertion at specific positions: ");
displayLinkedList(head);     printf("deletion\n");
deleteAtBegining(&head);     deleteAtIndex(&head, 1);
deleteAtEnd(&head);     displayLinkedList(head);     return
0;
}
```

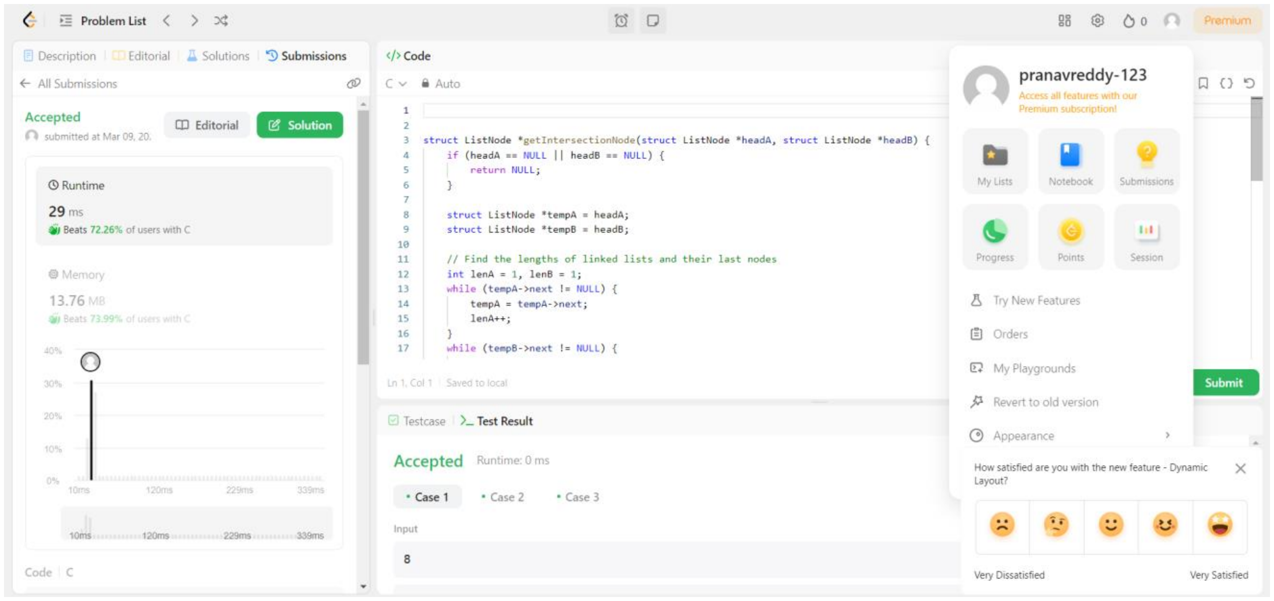
OUTPUT:



```
Linked list after insertion at the beginning: 1 -> 3 -> 5 -> NULL
Linked list after insertion at the end: 1 -> 3 -> 5 -> 6 -> 7 -> NULL
Linked list after insertion at specific positions: 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> NULL
deletion
2 -> 4 -> 5 -> 6 -> NULL

Process returned 0 (0x0)   execution time : 0.016 s
Press any key to continue.
```

b) Program - Leetcode platform

OUTPUT:



Accepted
submitted at Mar 09, 20.

Runtime
29 ms
Beats 72.26% of users with C

Memory
13.76 MB
Beats 73.99% of users with C

```c

struct ListNode *getIntersectionNode(struct ListNode *headA, struct ListNode *headB) {
    if (headA == NULL || headB == NULL) {
        return NULL;
    }

    struct ListNode *tempA = headA;
    struct ListNode *tempB = headB;

    // Find the lengths of linked lists and their last nodes
    int lenA = 1, lenB = 1;
    while (tempA->next != NULL) {
        tempA = tempA->next;
        lenA++;
    }
    while (tempB->next != NULL) {
```

Ln 1, Col 1 | Saved to local

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

8

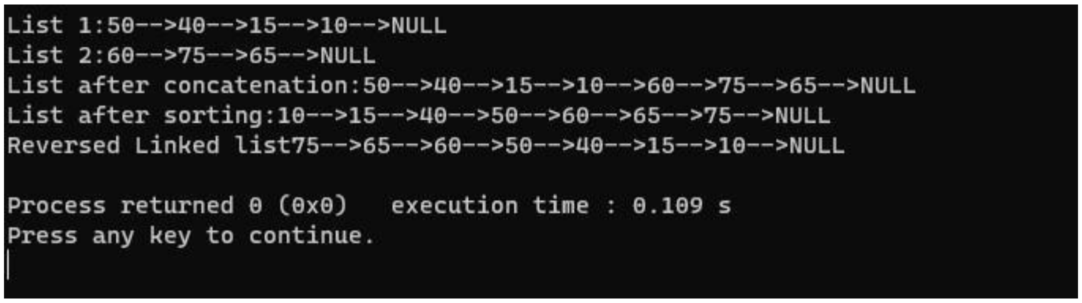
6a) WAP to Implement Singly Linked List with following operations: Sort the linked list, Reverse the linked list, Concatenation of two linked lists.

```c
#include <stdio.h>

#include <stdlib.h> struct

node

{    int data;     struct

node *next;

};

void insertatbegin(struct node **head, int value)

{
   struct node *new_node = (struct node *)malloc(sizeof(struct node));
new_node->data = value;     new_node->next = *head;
   *head = new_node;
}
void concat(struct node *head1, struct node *head2)
{
   if (head1->next == NULL)
head1->next = head2;     else
       concat(head1->next, head2);
}
void sortlist(struct node **head1)
{
   struct node *temp, *i;
   for (temp = *head1; temp != NULL; temp = temp->next)
   {
       for (i = temp->next; i != NULL; i = i->next)
       {
           if (i->data < temp->data)
           {
               int tem = i->data;
i->data = temp->data;
temp->data = tem;
```

```
            }
        }
    }
}
void reverse(struct node **head1)
{
    struct node *prev = NULL;
struct node *current = *head1;
struct node *next = NULL;     while
(current != NULL)
    {
        next = current->next;
current->next = prev;
prev = current;         current
= next;
    }
    *head1 = prev;
}
void printlist(struct node *node)
{
    struct node *temp = node;
while (temp != NULL)
```

```
   {
       printf("%d-->", temp->data);
temp = temp->next;
   }
   printf("NULL\n");
}
int main()
{
   struct node *head1 = NULL;     insertatbegin(&head1, 10);
insertatbegin(&head1, 15);     insertatbegin(&head1, 40);
insertatbegin(&head1, 50);     printf("List 1:");     printlist(head1);
struct node *head2 = NULL;     insertatbegin(&head2, 65);
insertatbegin(&head2, 75);     insertatbegin(&head2, 60);     printf("List
2:");     printlist(head2);     concat(head1, head2);     printf("List after
concatenation:");     printlist(head1);     sortlist(&head1);     printf("List
after sorting:");     printlist(head1);     reverse(&head1);
printf("Reversed Linked list");     printlist(head1);
}
```

OUTPUT:

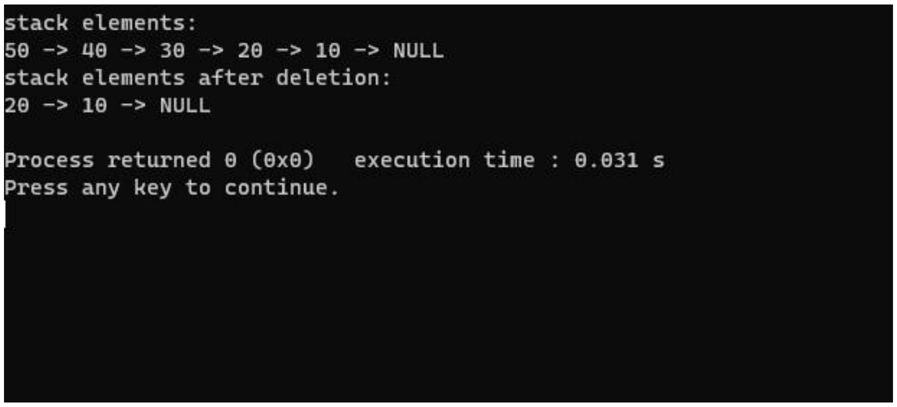

```
List 1:50-->40-->15-->10-->NULL
List 2:60-->75-->65-->NULL
List after concatenation:50-->40-->15-->10-->60-->75-->65-->NULL
List after sorting:10-->15-->40-->50-->60-->65-->75-->NULL
Reversed Linked list75-->65-->60-->50-->40-->15-->10-->NULL

Process returned 0 (0x0)   execution time : 0.109 s
Press any key to continue.
```

6b) WAP to Implement Singly Linked List to simulate Stack & Queue Operations.

```
#include <stdio.h>

#include <stdlib.h> struct

Node

{    int data;     struct

Node *next;

};

void insertAtBeginning(struct Node **head, int value)

{

   struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));

newNode->data = value; newNode->next = *head;

   *head = newNode;

}

void deleteAtBeginning(struct Node **head)

{

   if (*head == NULL)

   {
        printf("Linked list is already empty.\n");
return;
    }
    struct Node *temp = *head;
*head = (*head)->next;
free(temp);
}
void display(struct Node *head)
{
    struct Node *temp = head;
if (temp == NULL)
    {
        printf("Linked list is empty.\n");
return;
    }
    while (temp != NULL)
    {
        printf("%d -> ", temp->data);
temp = temp->next;
    }
    printf("NULL\n");
}
```

```
int main()

{

   struct Node *head = NULL;

insertAtBeginning(&head, 10);

insertAtBeginning(&head, 20);

insertAtBeginning(&head, 30);

insertAtBeginning(&head, 40);

insertAtBeginning(&head, 50);

printf("stack elements:\n");

display(head);

deleteAtBeginning(&head);

deleteAtBeginning(&head);

deleteAtBeginning(&head); printf("stack

elements after deletion:\n"); display(head);

   return 0;

}
```

OUTPUT:



```
stack elements:
50 -> 40 -> 30 -> 20 -> 10 -> NULL
stack elements after deletion:
20 -> 10 -> NULL

Process returned 0 (0x0)   execution time : 0.031 s
Press any key to continue.
```

```
#include <stdio.h>

#include <stdlib.h> struct

Node

{    int data;     struct

Node *next;

};

void insertAtEnd(struct Node **head, int value)

{

   struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));

struct Node *temp = *head;     newNode->data = value;     newNode-

>next = NULL;

   if (*head == NULL)

   {

      *head = newNode;

      return;

   }
```

```
    while (temp->next != NULL)
    {
        temp = temp->next;
    }
    temp->next = newNode;
}
void deleteAtBeginning(struct Node **head)
{
    if (*head == NULL)
    {
        printf("Linked list is already empty.\n");
return;
    }
    struct Node *temp = *head;
*head = (*head)->next;
free(temp);

}
void display(struct Node *head)
{
    struct Node *temp = head;
if (temp == NULL)
```

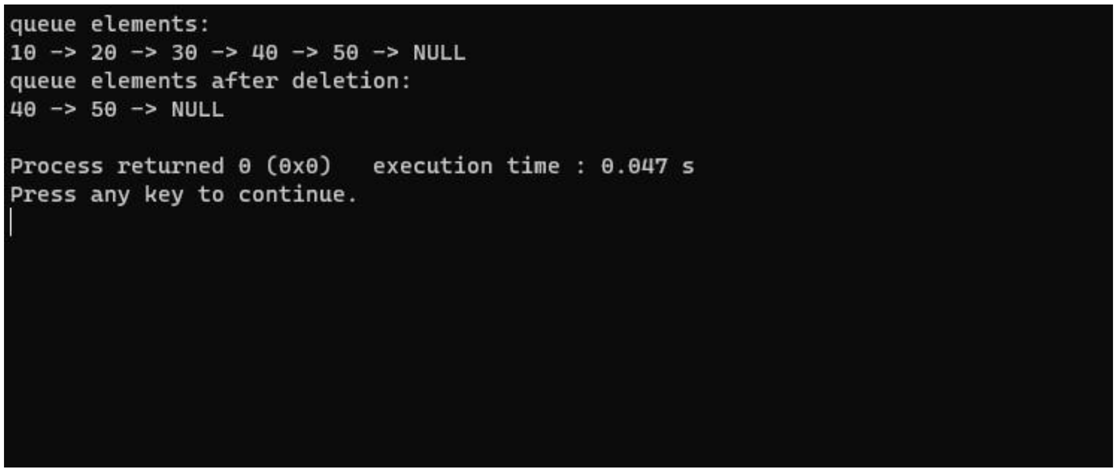
```
   {
       printf("Linked list is empty.\n");
return;
   }
   while (temp != NULL)
   {
       printf("%d -> ", temp->data);
temp = temp->next;
   }
   printf("NULL\n");
}
int main()
{
   struct Node *head = NULL;
insertAtEnd(&head, 10);     insertAtEnd(&head,
20); insertAtEnd(&head, 30);
insertAtEnd(&head, 40);     insertAtEnd(&head,
50);     printf("queue elements:\n");
display(head);     deleteAtBeginning(&head);
deleteAtBeginning(&head);
deleteAtBeginning(&head);     printf("queue
```

elements after deletion:\n");   display(head);

return 0;

}

OUTPUT:



```
queue elements:
10 -> 20 -> 30 -> 40 -> 50 -> NULL
queue elements after deletion:
40 -> 50 -> NULL

Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.
```

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

Display the contents of the list

#include <stdio.h>

#include <stdlib.h> struct

Node

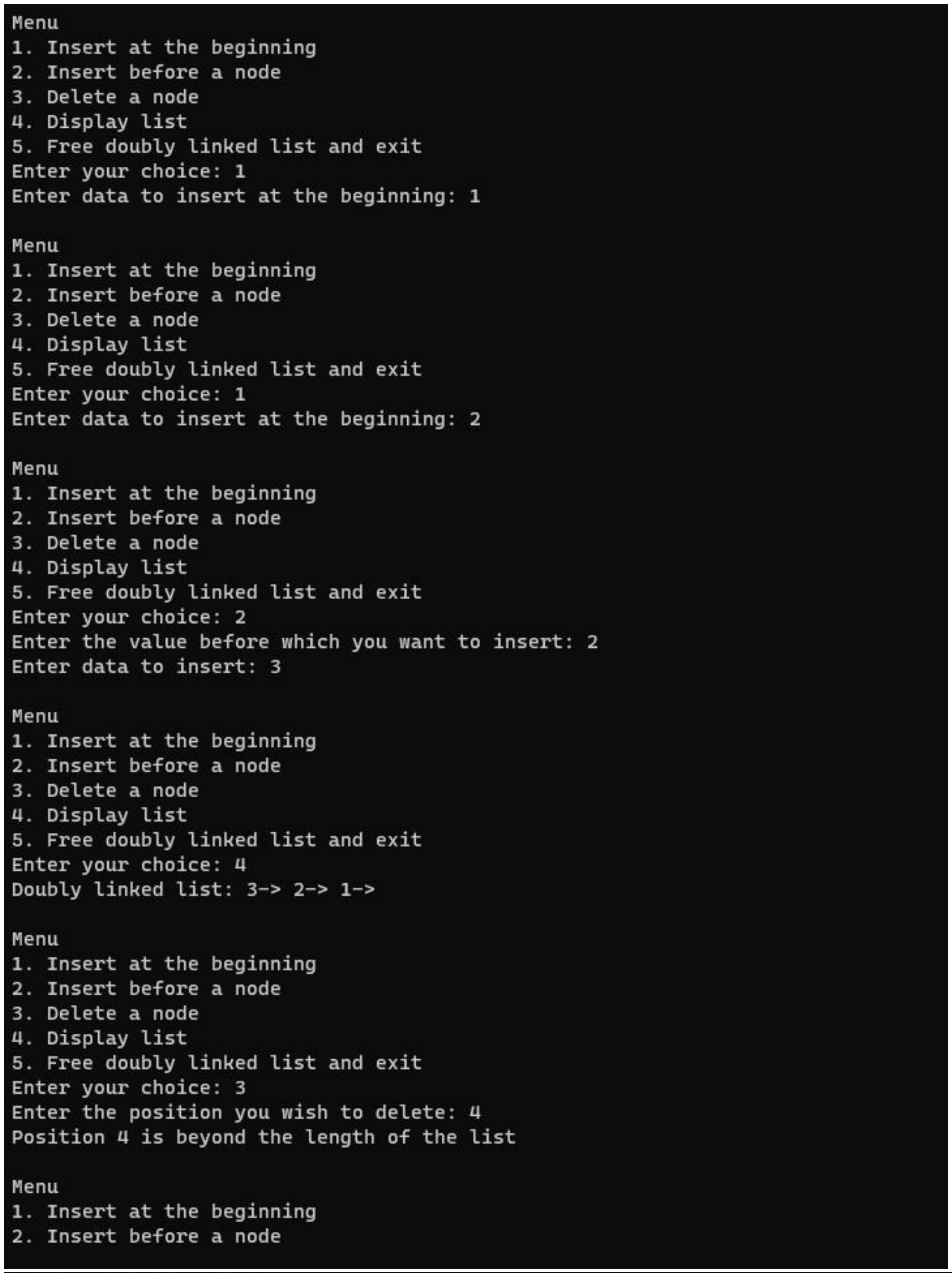
```
{    int data;    struct
Node *prev;    struct
Node *next;
};
struct Node *createNode(int data)
{
   struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
if (newNode == NULL)
   {
       printf("Memory allocation failed\n");
exit(EXIT_FAILURE);
   }
   newNode->data = data;
newNode->prev = NULL;
newNode->next = NULL;    return
newNode;
}
void insertAtBeginning(struct Node **head, int data)
{
   struct Node *newNode = createNode(data);
if (*head == NULL)
   {
       *head = newNode;
   }
   else
   {
       newNode->next = *head;
       (*head)->prev = newNode;
       *head = newNode;
   }
}
void insertBeforeNode(struct Node **head, int key, int data)
{
   if (*head == NULL)
   {
       printf("List is empty\n");
return;
   }
   struct Node *newNode = createNode(data);
struct Node *current = *head;     while
(current)
   {
       if (current->data == key)
       {
           if (current->prev)
           {
               current->prev->next = newNode;
newNode->prev = current->prev;
           }
else
           {
               *head = newNode;
           }
           newNode->next = current;             current->prev = newNode;
           return;
       }
       current = current->next;
   }
   printf("Key not found in the list\n");
}
void deleteNode(struct Node **head, int pos)
{
   if (*head == NULL)
   {
       printf("List is empty\n");
return;
   }
   struct Node *current = *head;
int count = 1;     while (current
&& count < pos)
   {
       current = current->next;
count++;
   }
   if (current == NULL)
   {
       printf("Position %d is beyond the length of the list\n", pos);
return;
   }
   if (current->prev)
   {
       current->prev->next = current->next;
   }
   else
   {
       *head = current->next;
   }
   if (current->next)
   {
       current->next->prev = current->prev;
   }
   free(current);     printf("Node at position %d
deleted\n", pos);
}
void displayList(struct Node *head)
{
   if (head == NULL)
   {
       printf("List is empty\n");
return;
   }
   struct Node *current = head;
while (current)
   {
       printf("%d-> ", current->data);
current = current->next;
   }
   printf("\n");
```

```
}
void freeList(struct Node *head)
{
   struct Node *current = head;
struct Node *nextNode;     while
(current)
   {
       nextNode = current->next;
      free(current);        current =
                  nextNode;
   }
}
int main()
{
   struct Node *head = NULL;
int ch, newData, pos, key;
while (1)
   {
       printf("\nMenu\n");        printf("1. Insert at
the beginning\n");        printf("2. Insert before a
node\n");        printf("3. Delete a node\n");
```

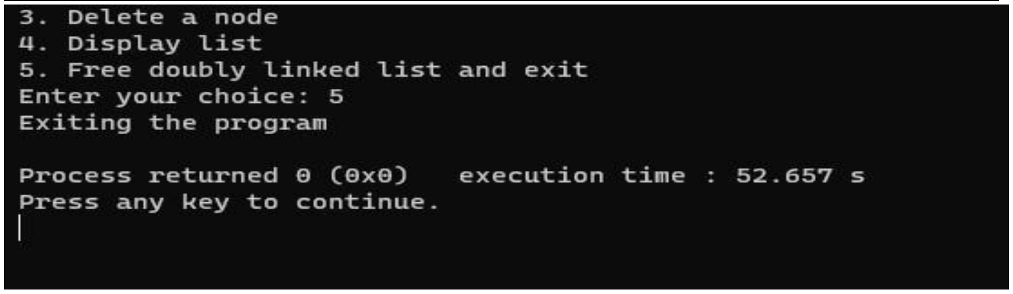
```
printf("4. Display list\n");        printf("5. Free

doubly linked list and exit\n");        printf("Enter

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

switch (ch)

      {

      case 1:

         printf("Enter data to insert at the beginning: ");

scanf("%d", &newData);           insertAtBeginning(&head,

newData);

         break;

case 2:

         printf("Enter the value before which you want to insert: ");

scanf("%d", &key);           printf("Enter data to insert: ");

scanf("%d", &newData);           insertBeforeNode(&head, key,

newData);

         break;

case 3:

         printf("Enter the position you wish to delete: ");

scanf("%d", &key);           deleteNode(&head, key);

         break;

case 4:
```

```
            printf("Doubly linked list: ");
displayList(head);             break;
        case 5:
            freeList(head);
printf("Exiting the program\n");
return 0;         default:
            printf("Invalid choice\n");
        }
    }
    return 0;
}
```

OUTPUT:



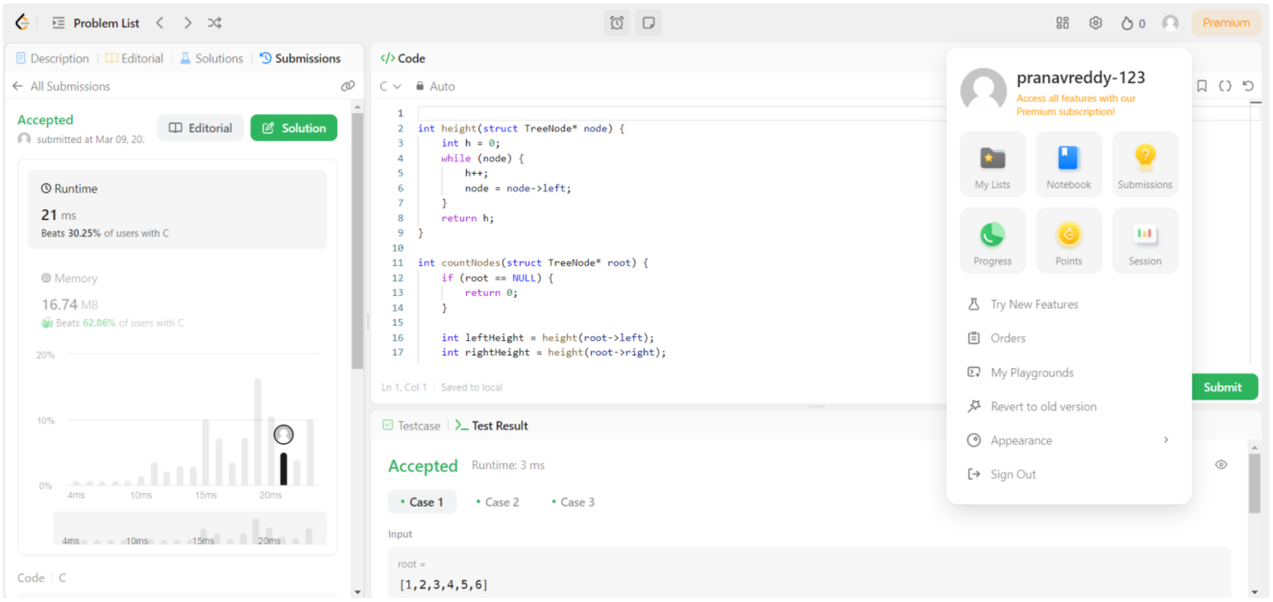
```
Menu
1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. Free doubly linked list and exit
Enter your choice: 1
Enter data to insert at the beginning: 1

Menu
1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. Free doubly linked list and exit
Enter your choice: 1
Enter data to insert at the beginning: 2

Menu
1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. Free doubly linked list and exit
Enter your choice: 2
Enter the value before which you want to insert: 2
Enter data to insert: 3

Menu
1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. Free doubly linked list and exit
Enter your choice: 4
Doubly linked list: 3-> 2-> 1->

Menu
1. Insert at the beginning
2. Insert before a node
3. Delete a node
4. Display list
5. Free doubly linked list and exit
Enter your choice: 3
Enter the position you wish to delete: 4
Position 4 is beyond the length of the list

Menu
1. Insert at the beginning
2. Insert before a node
```



```
3. Delete a node
4. Display list
5. Free doubly linked list and exit
Enter your choice: 5
Exiting the program

Process returned 0 (0x0)   execution time : 52.657 s
Press any key to continue.
```

7b) Program - Leetcode platform

OUTPUT:


Accepted
submitted at Mar 09, 20

Runtime
21 ms
Beats 30.25% of users with C

Memory
16.74 MB
Beats 62.86% of users with C

pranavreddy-123
Access all features with our Premium subscription!

```c
int height(struct TreeNode* node) {
    int h = 0;
    while (node) {
        h++;
        node = node->left;
    }
    return h;
}

int countNodes(struct TreeNode* root) {
    if (root == NULL) {
        return 0;
    }

    int leftHeight = height(root->left);
    int rightHeight = height(root->right);
```

Ln 1, Col 1 | Saved to local

Accepted Runtime: 3 ms

Case 1 Case 2 Case 3

Input

root =
[1,2,3,4,5,6]

8a) 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

To display the elements in the tree.

#include <stdio.h>

#include <stdlib.h> struct

node

{ int data; struct

node *left_side; struct

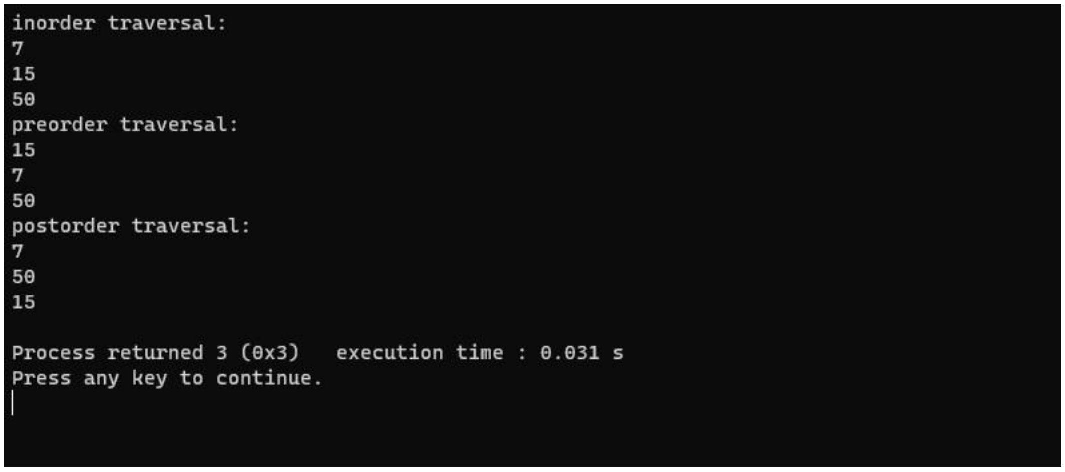
node *right_side

};

struct node *newnode(int x)

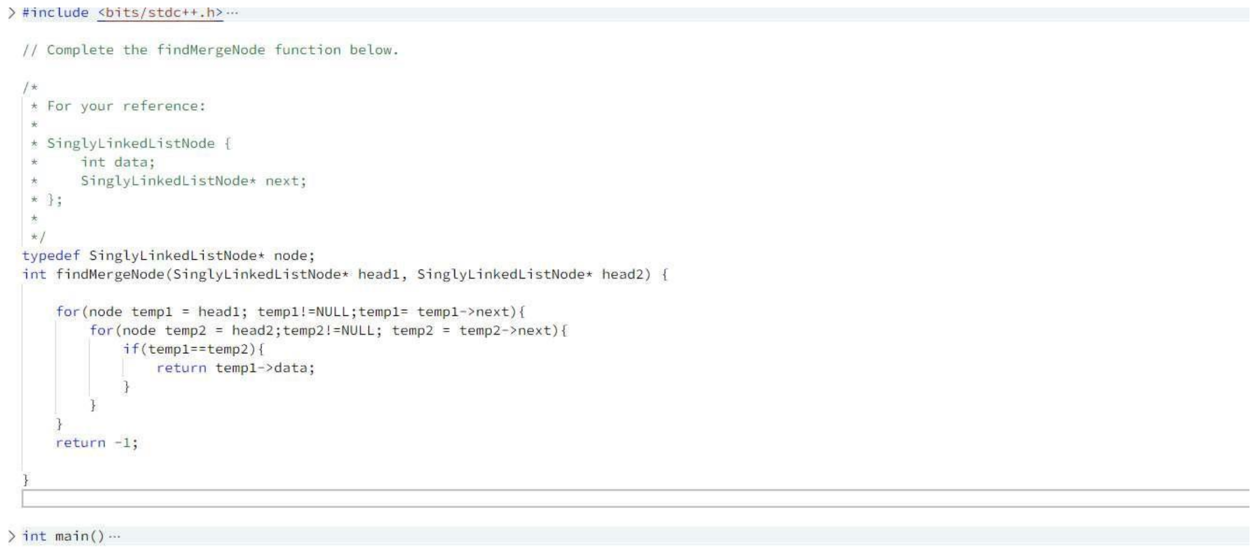
```
{
   struct node *temp = malloc(sizeof(struct node));
temp->data = x;     temp->left_side = NULL;
temp->right_side = NULL;     return temp;
}
struct node *insert(struct node *root, int x)
{
   if (root == NULL)
   {
      return newnode(x);
   }
   else if (x > root->data)
   {
      root->right_side = insert(root->right_side, x);
   }
   else
   {
      root->left_side = insert(root->left_side, x);
   }
   return root;
}
void inorder(struct node *root)
{
   if (root != NULL)
   {
        inorder(root->left_side);
printf("%d\n", root->data);
inorder(root->right_side);
    }
}
void postorder(struct node *root)
{
    if (root != NULL)
    {
        postorder(root->left_side);
postorder(root->right_side);         printf("%d\n",
root->data);
    }
}
void preorder(struct node *root)
{
    if (root != NULL)
    {
        printf("%d\n", root->data);         preorder(root-
>left_side);         preorder(root->right_side);
    }
}
void main()
{
   struct node *root = NULL;
root = insert(root, 15);     root =
insert(root, 7);     root =
insert(root, 50);
printf("inorder traversal:\n");
inorder(root); printf("preorder
traversal:\n");
   preorder(root);
   printf("postorder traversal:\n");
postorder(root);
}
```

OUTPUT:



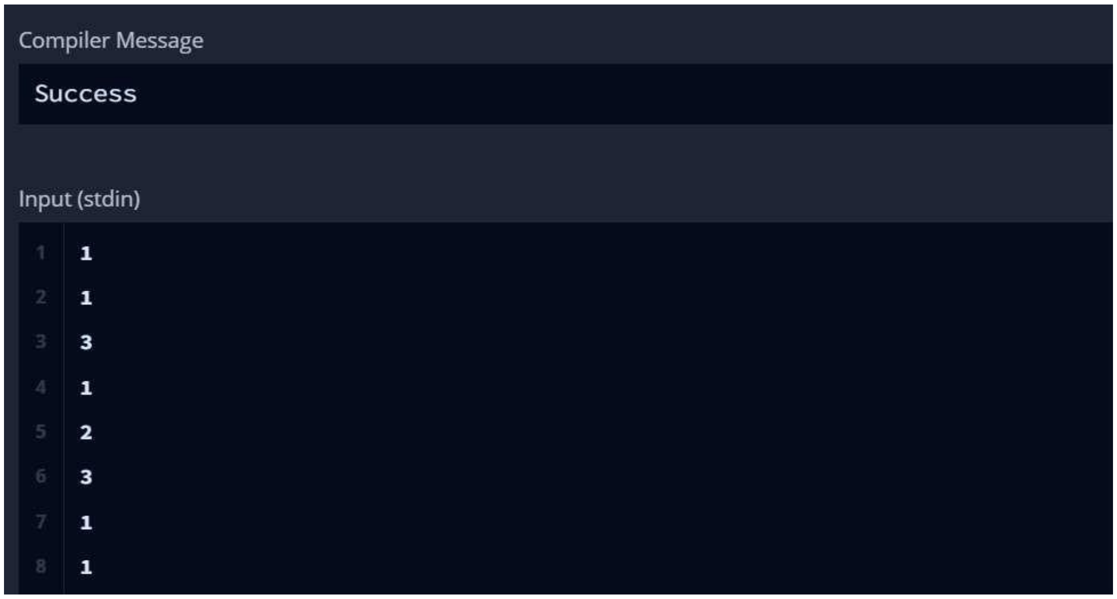
```
inorder traversal:
7
15
50
preorder traversal:
15
7
50
postorder traversal:
7
50
15

Process returned 3 (0x3)   execution time : 0.031 s
Press any key to continue.
```

8b) Program - Leetcode platform



```
> #include <bits/stdc++.h>···

// Complete the findMergeNode function below.

/*
 * For your reference:
 *
 * SinglyLinkedListNode {
 *     int data;
 *     SinglyLinkedListNode* next;
 * };
 *
 */
typedef SinglyLinkedListNode* node;
int findMergeNode(SinglyLinkedListNode* head1, SinglyLinkedListNode* head2) {

    for(node temp1 = head1; temp1!=NULL;temp1= temp1->next){
        for(node temp2 = head2;temp2!=NULL; temp2 = temp2->next){
            if(temp1==temp2){
                return temp1->data;
            }
        }
    }
    return -1;

}

> int main()···
```

OUTPUT:

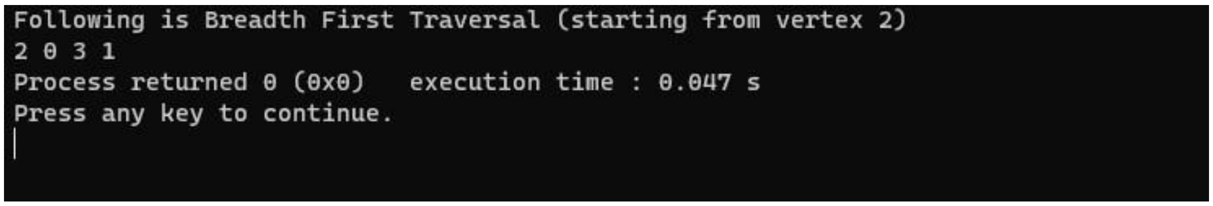


```
Compiler Message
Success

Input (stdin)
1
1
3
1
2
3
1
1
```

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

```
#include <stdbool.h>

#include <stdio.h>

#include <stdlib.h>

#define MAX_VERTICES 50 typedef

struct Graph_t

{    int

V;

   bool adj[MAX_VERTICES][MAX_VERTICES];

} Graph;

Graph *Graph_create(int V)

{

   Graph *g = malloc(sizeof(Graph));

g->V = V;     for (int i = 0; i < V; i++)

   {

      for (int j = 0; j < V; j++)

      {

         g->adj[i][j] = false;

      }

   }
   return g;
}
void Graph_destroy(Graph *g) { free(g); }
void Graph_addEdge(Graph *g, int v, int w)
{
   g->adj[v][w] = true;
}
void Graph_BFS(Graph *g, int s)
{
   bool visited[MAX_VERTICES];
for (int i = 0; i < g->V; i++)
   {
      visited[i] = false;
   }
   int queue[MAX_VERTICES];
int front = 0, rear = 0;
visited[s] = true;
queue[rear++] = s;     while
(front != rear)
   {
      s = queue[front++];
printf("%d ", s);
      for (int adjacent = 0; adjacent < g->V;
adjacent++)
      {
            if (g->adj[s][adjacent] && !visited[adjacent])
            {
                visited[adjacent] = true;
queue[rear++] = adjacent;
            }
        }
    }
}
int main()
{
    Graph *g = Graph_create(4);
    Graph_addEdge(g, 0, 1);
Graph_addEdge(g, 0, 2);     Graph_addEdge(g,
1, 2);     Graph_addEdge(g, 2, 0);
    Graph_addEdge(g,                  2,                  3);
Graph_addEdge(g, 3, 3);     printf("Following is
Breadth First Traversal "
           "(starting from vertex 2) \n");
    Graph_BFS(g, 2);
Graph_destroy(g);     return
0;
}
```

OUTPUT:

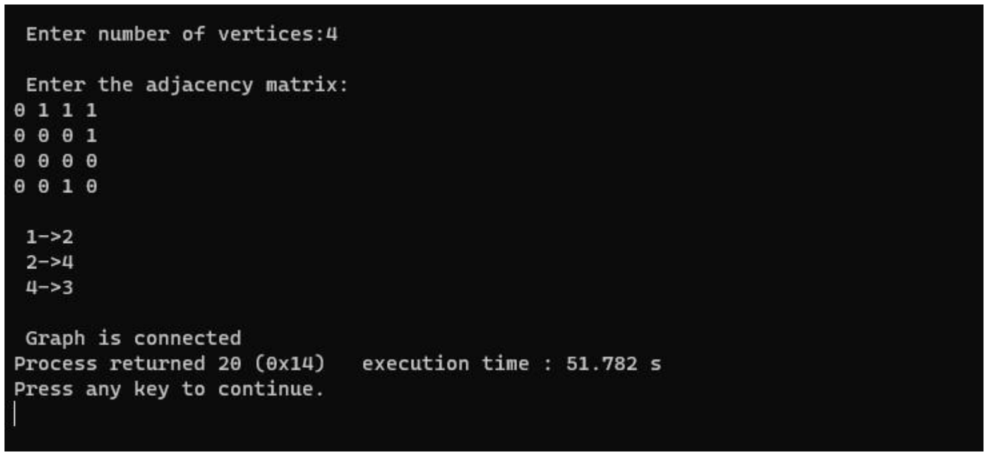


```
Following is Breadth First Traversal (starting from vertex 2)
2 0 3 1
Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.
```

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

```
#include <stdio.h> int
a[20][20], reach[20], n;
void dfs(int v)
{    int
i;
   reach[v] = 1;     for (i = 1;
i <= n; i++)        if (a[v][i]
&& !reach[i])
      {
         printf("\n %d->%d", v, i);
dfs(i);
      }
}
void main()
{
   int i, j, count = 0;
   printf("\n Enter number of vertices:");
scanf("%d", &n);     for (i = 1; i <= n; i++)
   {
        1reach [i] = 0;
for (j = 1; j <= n; j++)
a[i][j] = 0;
    }
    printf("\n Enter the adjacency matrix:\n");
for (i = 1; i <= n; i++)        for (j = 1; j <= n;
j++)            scanf("%d", &a[i][j]);     dfs(1);
printf("\n");     for (i = 1; i <= n; i++)
    {        if
(reach[i])
count++;
    }
    if (count == n)        printf("\n
Graph is connected");     else
        printf("\n Graph is not connected");
}
```

OUTPUT:



```
 Enter number of vertices:4

 Enter the adjacency matrix:
0 1 1 1
0 0 0 1
0 0 0 0
0 0 1 0

 1->2
 2->4
 4->3

 Graph is connected
Process returned 20 (0x14)   execution time : 51.782 s
Press any key to continue.
```