VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
on

Data Structures using C

Submitted by

S Rakhal
1BM22CS229

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 byS Rakhal**(1BM22CS229),** 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.

Lakshmi Neelima	Dr. Jyothi S Nayak
Assistant professor	Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

Index Sheet

	Display the contents of the linked list. 5b) Program - Leetcode platform	
6	6a) WAP to Implement Single Link List with following operations: Sort the linked list, Reverse the linked list, Concatenation of two linked lists. 6b) WAP to Implement Single Link List to simulate Stack & Queue Operations.	**40-53**
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	**54-68**
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	**69-74**
9	9a) Write a program to traverse a graph using BFS method. 9b) Write a program to check whether given graph is connected or not using DFS method.	**75-79**
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

Code:

```
#include  <stdio.h>
#include <stdlib.h>
#define N 3
void push();
void pop();
void display();
int top=-1;
int stack[N];

void main()
{
        int choice;
        printf("Enter 1:push 2:pop 3:display 4:exit\n");
        while(1)
        {
```

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

void push()
{
        int x;
        if(top>=N){
```

```
        printf("Stack is full, overflow\n");
        }
        else{
        top++;
        printf("Enter the element\n");
        scanf("%d",&x);
        stack[top]=x;
        printf("Element %d is pushed in stack\n",x);
        }
}

void pop()
{
        if(top==-1){
        printf("Stack is empty, underflow\n");
        }
        else{
        int data=stack[top];
        printf("Element %d is popped from stack\n",stack[top]);
        top=top-1;
        }
```

```
}


void display()
{
        if(top<=N && top>=0){
        printf("The elements of stack are\n");
        for(int i=top;i>=0;i--){
        printf("%d\t",stack[i]);
        }
        printf("\n");
        }
        else{
        printf("Stack is empty\n");
        }
}
```

Output :

```
Enter 1:push 2:pop 3:display 4:exit
1
Enter the element
10
Element 10 is pushed in stack
1
Enter the element
20
Element 20 is pushed in stack
1
Enter the element
30
Element 30 is pushed in stack
1
Stack is full, overflow
3
The elements of stack are
30      20      10
2
Element 30 is popped from stack
2
Element 20 is popped from stack
2
Element 10 is popped from stack
2
Stack is empty, underflow
4

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)

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

```
#include  <stdio.h>
#include <stdlib.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+D)-(D*E)
The corresponding postfix expression is:
AB+CD+/DE*-

Press any key to continue . . .
```

3.a) 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

Code :

```
#include  <stdio.h>
#include <stdlib.h>
#define N 3

int q[N];
int front=-1;
int rear=-1;

void enque();
int deque();
void display();

void enque(){
   int c;
   if(rear==N-1){
      printf("Overflow\n");
      return;
   }
   printf("Enque : ");
   scanf("%d",&c);
   if(front==-1 && rear==-1){
      front=rear=0;
   }
   else{
      rear++;
   }
```

```
    q[rear]=c;
}

int deque(){
    int a;
    if(front==-1 && rear == -1){
        printf("UnderFlow");
        exit(0);
    }
    if(front == rear && rear!=0){
        a=q[front];
        front=rear=-1;
        return a;
    }
    else if(front==0 && rear==0){
        a=q[front];
        front=rear=-1;
        return a;
    }
    else if(front!=-1 && rear>front){
        a=q[front];
        front++;
        return a;
    }
}

void display(){
        for(int i=front;i<=rear;i++){
        printf("%d\t",q[i]);
        }
        printf("\n");
}

int main(){
    int ch;
```

```
    int b;
    while(1){
        printf("Enter 1:Enque 2:Deque 3:Display 4:Exit \n");
        scanf("%d",&ch);
        switch(ch){
            case 1:
                enque()
                ; break;
            case 2:
                b=deque();
                printf("Dequed element : %d\n",b);
                break;
            case 3:
                display();
                break;
            case 4:
                exit(0);
            default:
                printf("invalid choice");
        }
    }
    return 0;
}
```

Output :

```
Enter 1:Enque  2:Deque  3:Display  4:Exit
1
Enque : 10
Enter 1:Enque  2:Deque  3:Display  4:Exit
1
Enque : 20
Enter 1:Enque  2:Deque  3:Display  4:Exit
1
Enque : 30
Enter 1:Enque  2:Deque  3:Display  4:Exit
1
Overflow
Enter 1:Enque  2:Deque  3:Display  4:Exit
3
10      20      30
Enter 1:Enque  2:Deque  3:Display  4:Exit
2
Dequed element : 10
Enter 1:Enque  2:Deque  3:Display  4:Exit
2
Dequed element : 20
Enter 1:Enque  2:Deque  3:Display  4:Exit
2
Dequed element : 30
Enter 1:Enque  2:Deque  3:Display  4:Exit
2
UnderFlow
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 & Display
The program should print appropriate messages for queue empty and queue overflow conditions.

Code:

```
#include <stdio.h>

#define N 3


int q[N];

int front=-1,rear=-1;


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

```
            delete();
            break;
        case 3:
            display();
            break;
        case 4:
            exit(0);
        }
    }
}

void insert(){
    int x;
    if(((rear+1)%N) == front){
        printf("Overflow\n");
        return;
    }

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

    if( front==-1 && rear==-1){
        front=rear=0;
```

```
        q[rear]=x;
    }

    else{
        rear=(rear+1)%N;
        q[rear]=x;
    }
}

void delete(){
    int val;
    if( front==-1 && rear==-1)
        printf("Underflow\n");
    else if( front==rear){
        val=q[front];
        front=-1;
        rear=-1;
        printf("%d deleted\n",val);
    }
    else {
        val=q[ front];
         front=(
```

```
 front+1)%N;

printf("%d deleted\n",val)
```

```
}
    }

    void display(){
       int i= front;
       if(front==-1 && rear==-1){
          printf("Underflow\n");}
       else{
          while(i!=rear)
          {
             printf("%d\t",q[i]);
             i=(i+1)%N;
          }
          printf("%d\n",q[rear]);
       }
    }
```

Output :

```
1. insert 2.delete 3.display 4.exit
1
Enter :10
1. insert 2.delete 3.display 4.exit
1
Enter :20
1. insert 2.delete 3.display 4.exit
1
Enter :30
1. insert 2.delete 3.display 4.exit
1
Overflow
1. insert 2.delete 3.display 4.exit
3
10      20      30
1. insert 2.delete 3.display 4.exit
2
10 deleted
1. insert 2.delete 3.display 4.exit
3
20      30
1. insert 2.delete 3.display 4.exit
1
Enter :40
1. insert 2.delete 3.display 4.exit
3
20      30      40
1. insert 2.delete 3.display 4.exit
2
20 deleted
1. insert 2.delete 3.display 4.exit
2
30 deleted
1. insert 2.delete 3.display 4.exit
2
40 deleted
1. insert 2.delete 3.display 4.exit
2
Underflow
1. insert 2.delete 3.display 4.exit
4
```

4a) WAP to Implement Singly Linked List with following operations
Create a linked list.
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> typedef struct Node { int data;
struct Node *next;
}Node;
void Ins_At_Beginning( Node **head_ref,int new_data); void Ins_At_End( Node
**head_ref,int new_data);
void Ins( Node **prev_node,int new_data,int pos); void Print_list(Node * next);
void Ins_At_Beginning( Node **head_ref,int new_data)
{
Node *new_node=(struct Node*)malloc(sizeof( Node)); new_node->data=new_data;
new_node->next=*head_ref;

*head_ref=new_node;
}
void Ins_At_End(Node **head_ref,int new_data)



{
Node *new_node=(struct Node*)malloc(sizeof( Node)); Node *last=*head_ref;
new_node->data=new_data; new_node->next=NULL;
if (*head_ref==NULL)
{
*head_ref=new_node; return ;
}
while (last->next!=NULL) last=last->next;
last->next=new_node;
```

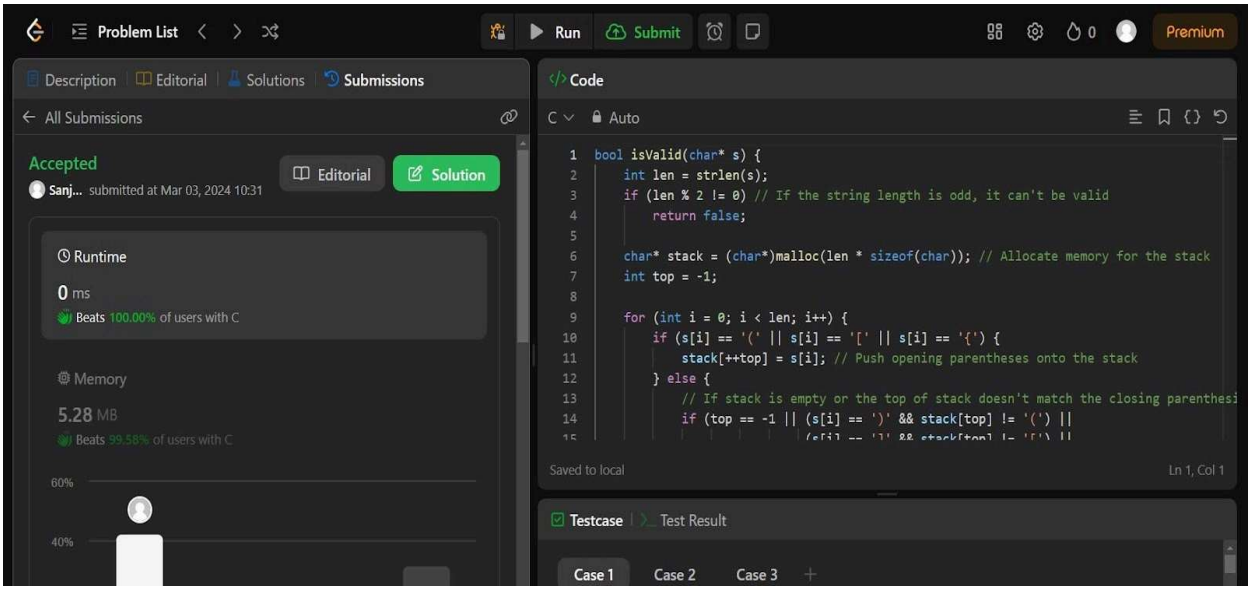
```
}
void Ins(Node **head_ref,int new_data,int pos)
{
if (*head_ref ==NULL)
{

printf("Cannot be NULL\n"); return;
}
Node *temp = *head_ref;
Node *newNode = ( Node *) malloc (sizeof ( Node)); newNode->data = new_data;

newNode->next = NULL; while (--pos>0)
{
temp = temp->next;
}
newNode->next = temp->next; temp->next = newNode;
}


void Print_list(Node *node)
{
while (node!=NULL)
{
printf("%d\n",node->data); node=node->next;
}

}


int main()
{
int ch,new,pos; Node* head=NULL; while(ch!=5)

{
printf("Menu\n");
```

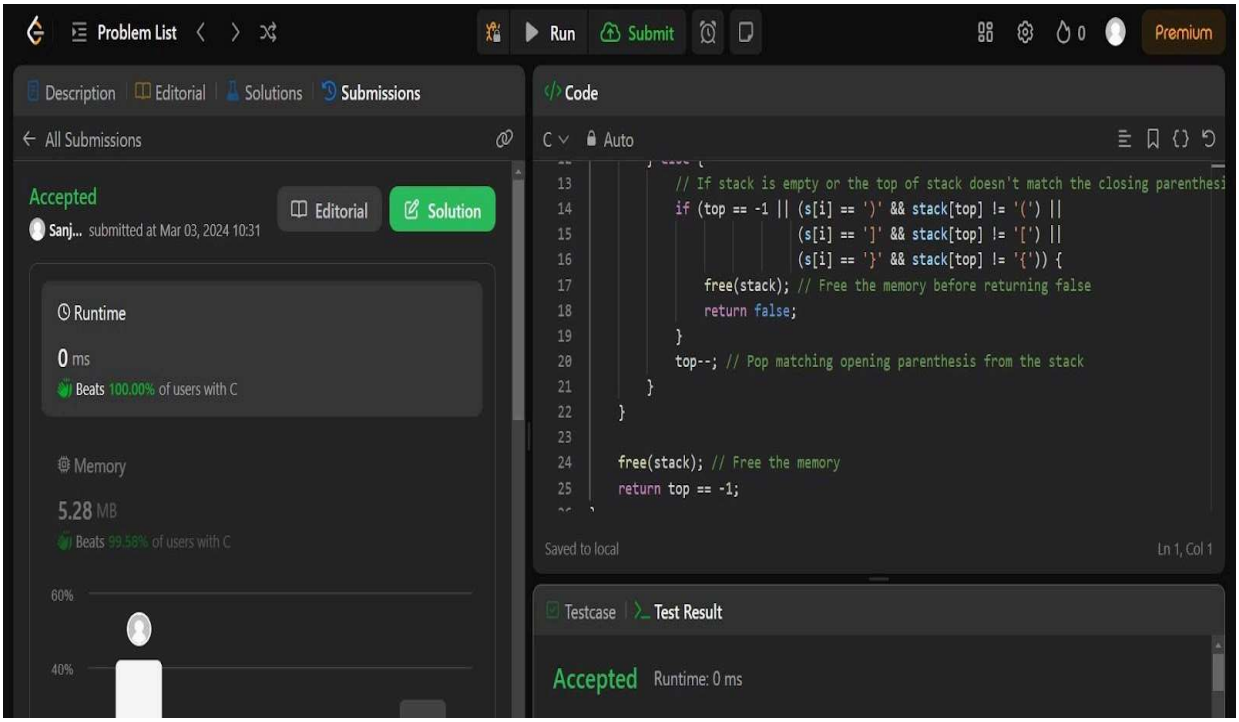
```
printf("1.Insert at the beginning\n"); printf("2.Insert at a specific position\n");
printf("3.Insert at the end\n"); printf("4.Display linked list\n"); printf("5.Exit\n");
printf("Enter your choice\n"); scanf("%d",&ch);
switch(ch)
{
case 1:
{
printf("Enter the data you want to insert at beginning\n"); scanf("%d",&new);

Ins_At_Beginning(&head,new); break;
}
case 2:
{
printf("Enter the data and position at which you want to insert \n");
scanf("%d%d",&new,&pos);
Ins(&head,new,pos); break;
}
case 3:



{
printf("Enter the data you want to insert at end\n"); scanf("%d",&new);
Ins_At_End(&head,new); break;
}
case 4:
{
printf("Created linked list is:\n"); Print_list(head);
break;

}
case 5:
{
return 0; break;
}
case 6:
```

```
{
printf("Invalid data!"); break;
}
}
}
return 0;
}
```

```
Menu
1.Insert at the beginning
2.Insert at a specific position
3.Insert at the end
4.Display linked list
5.Exit
Enter your choice
2
Enter the data and position at which you want to insert
20
1
Menu
1.Insert at the beginning
2.Insert at a specific position
3.Insert at the end
4.Display linked list
5.Exit
Enter your choice
4
Created linked list is:
10
20
30
Menu
1.Insert at the beginning
2.Insert at a specific position
3.Insert at the end
4.Display linked list
5.Exit
Enter your choice
5
```

4b) Program - Leetcode platform





5a) WAP to Implement Singly Linked List with following operations
Create a linked list.
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> typedef struct Node { int data;
struct Node *next;
}Node;
void InsertAtBeginning( Node **head_ref,int new_data); void DeleteAtBeginning(
Node **head_ref);
void DeleteAtEnd( Node **head_ref); void Delete( Node **prev_node,int pos); void
PrintList(Node * next);
void InsertAtBeginning( Node **head_ref,int new_data)
{
Node *new_node=(struct Node*)malloc(sizeof( Node)); new_node->data=new_data;
new_node->next=*head_ref;

*head_ref=new_node;
}
void DeleteAtBeginning( Node **head_ref)
{
Node *ptr;


if(head_ref == NULL)
{
printf("\nList is empty");
}
else
{
ptr = *head_ref;
*head_ref = ptr->next; free(ptr);
printf("\n Node deleted from the beginning ...");
}
```

```
}
void DeleteAtEnd(Node **head_ref)
{
Node *ptr,*ptr1; if(*head_ref == NULL)
{

printf("\nlist is empty");
}


else if((*head_ref)-> next == NULL)
{
free(*head_ref);
*head_ref= NULL;
printf("\nOnly node of the list deleted ...");
}
else
{
ptr = *head_ref; while(ptr->next != NULL)
{
ptr1 = ptr;
ptr = ptr ->next;
}


ptr1->next = NULL; free(ptr);
printf("\n Deleted Node from the last ...");
}
}

void Delete(Node **head_ref, int pos)
{
Node *temp = *head_ref, *prev; if (temp == NULL)
{
printf("\nList is empty"); return;
```

```
}
if (pos == 1)
{
*head_ref = temp->next; free(temp);
printf("\nDeleted node with position %d", pos); return;
}
for (int i = 0; temp != NULL && i < pos - 1; i++)


{
prev = temp;
temp = temp->next;
}
if (temp == NULL)

printf("\nPosition out of range"); return;
}
prev->next = temp->next; free(temp);
printf("\nDeleted node with position %d", pos);
}
void PrintList(Node *node)
{
while (node!=NULL)
{
printf("%d\n",node->data); node=node->next;
}
}


int main()
{
int ch,new,pos;


Node* head=NULL; while(ch!=6)
```

```
printf("Menu\n"); printf("1.Create a linked list\n"); printf("2.Delete at beginning\n");
printf("3.Delete at a specific position\n"); printf("4..Delete at end\n"); printf("5..Display
linked list\n"); printf("6..Exit\n");
printf("Enter your choice\n"); scanf("%d",&ch);
switch(ch)
{
case 1:
{
printf("Enter the data you want to insert at beginning\n"); scanf("%d",&new);
InsertAtBeginning(&head,new); break;
}
case 2:
{
DeleteAtBeginning(&head); break;
}

case 3:
{


printf("Enter the position at which you want to delete \n"); scanf("%d",&pos);
Delete(&head,pos); break;
}
case 4:
{
DeleteAtEnd(&head); break;
}
case 5:
{
printf("Created linked list is:\n"); PrintList(head);
break;
}
case 6:
{
```

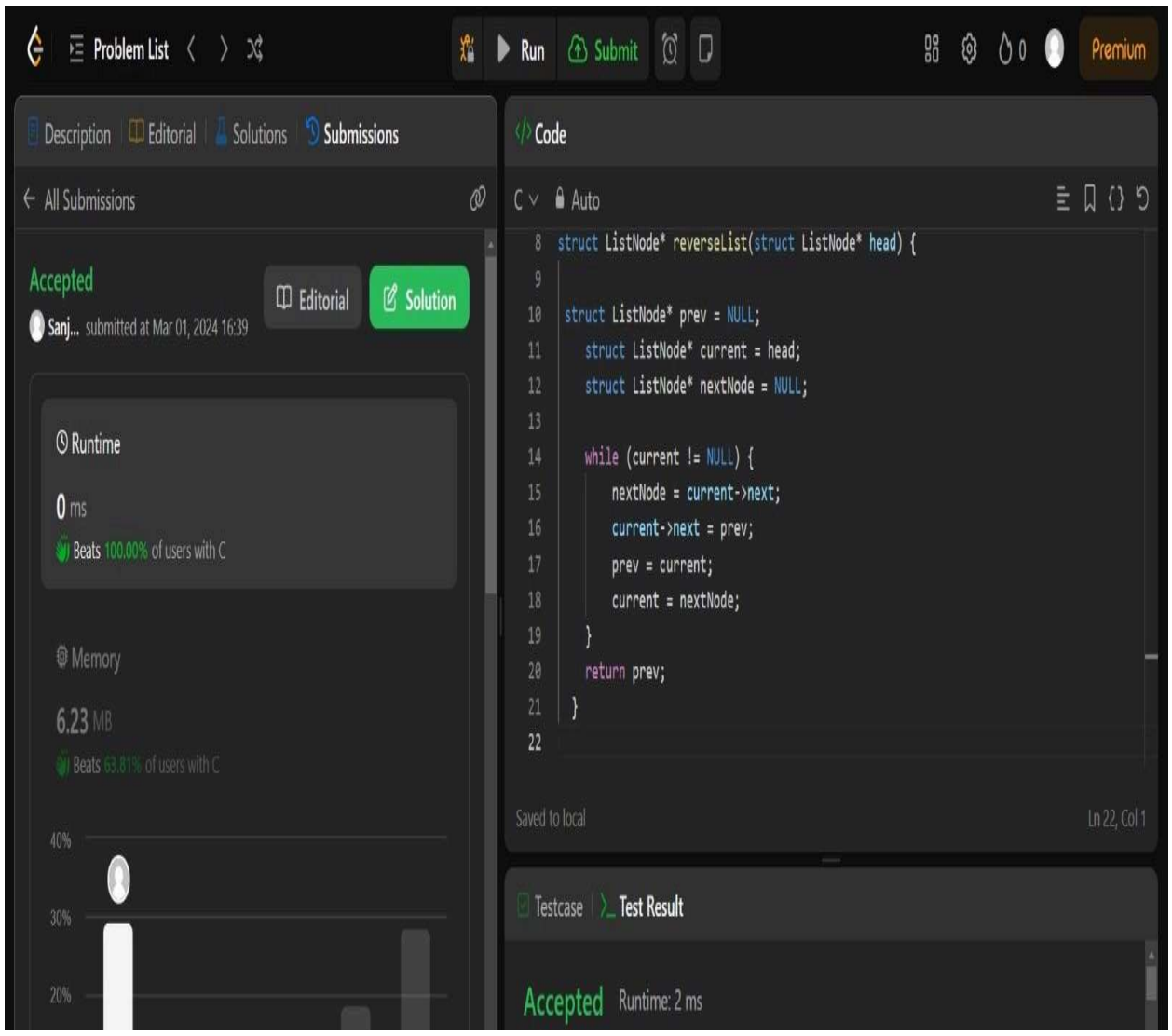
```
return 0; break;

}
default:
{
printf("Invalid data!"); break;
}
}
}


return 0;
}
```

```
Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
Enter your choice
1
Enter the data you want to insert at beginning
10
Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
Enter your choice
1
Enter the data you want to insert at beginning
20
Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
Enter your choice
5
Created linked list is:
20
10
Menu
1.Create a linked list
2.Delete at beginning
```

```
Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
Enter your choice
2

 Node deleted from the beginning ...Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
Enter your choice
4

Only node of the list deleted ...Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
Enter your choice
1
Enter the data you want to insert at beginning
30
Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
```

```
4

Only node of the list deleted ...Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
Enter your choice
1
Enter the data you want to insert at beginning
30
Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
Enter your choice
5
Created linked list is:
30
Menu
1.Create a linked list
2.Delete at beginning
3.Delete at a specific position
4..Delete at end
5..Display linked list
6..Exit
Enter your choice
6
```

5b) Program - Leetcode platform



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

```
#include <stdio.h> #include <stdlib.h> struct node{
int data;
struct node *next;
};
struct node* head=NULL; struct node* head2=NULL; void insert1(int val){
struct node* newnode=(struct node*)malloc(sizeof(struct node));
newnode->data=val;
newnode->next=head; head=newnode;
}
void insert2(int val){
struct node* newnode=(struct node*)malloc(sizeof(struct node));
newnode->data=val;
newnode->next=head2; head2=newnode;

}

void sort(){
struct node *current=head; struct node *index=NULL; int temp;
if(head==NULL){ printf("List is empty"); return;
}
else{ while(current!=NULL){ index=current->next; while(index!=NULL){
if(current->data > index->data){ temp=current->data;
current->data=index->data; index->data=temp;
}
index=index->next;
}
current=current->next;
}
}
}
```

```
void reverse(){
struct node* temp=head; struct node* prev=NULL; while(temp!=NULL){
struct node* front=temp->next; temp->next=prev;
prev=temp; temp=front;
}
head=prev;
}
void concat(){
struct node* temp=head; while(temp->next!=NULL){ temp=temp->next;
}
temp->next=head2;
}
void display1(){
struct node*temp=head; if(head==NULL){ printf("List is empty");

return;



}
while(temp!=NULL){ printf("%d\t",temp->data); temp=temp->next;
}
}
void display2(){
struct node*temp=head2; if(head2==NULL){ printf("List is empty"); return;
}
while(temp!=NULL){ printf("%d\t",temp->data); temp=temp->next;
}
}
int main(){ int ch,val;
while(ch!=8){
printf("\nMenu 1:Insert_1 2:Insert_2 3:Display_1 4:Display_2 5:Sort 6:Reverse

7:Concatenate 8:Exit : "); scanf("%d",&ch); switch(ch){
```

```
case 1:
printf("Enter data : "); scanf("%d",&val); insert1(val);
break; case 2:
printf("Enter data : "); scanf("%d",&val); insert2(val);
break; case 3:
printf("Elements of linked list 1:\n"); display1();
break; case 4:
printf("Elements of linked list 2:\n"); display2();
break;
case 5:

sort(); break; case 6: reverse(); break; case 7: concat();

break; case 8:
return 0;
}
}
return 0;
}
```

```
Menu 1:Insert_1 2:Insert_2 3:Display_1 4:Display_2 5:Sort 6:Reverse 7:Concatenate 8:Exit : 1
Enter data : 10

Menu 1:Insert_1 2:Insert_2 3:Display_1 4:Display_2 5:Sort 6:Reverse 7:Concatenate 8:Exit : 1
Enter data : 20

Menu 1:Insert_1 2:Insert_2 3:Display_1 4:Display_2 5:Sort 6:Reverse 7:Concatenate 8:Exit : 2
Enter data : 30

Menu 1:Insert_1 2:Insert_2 3:Display_1 4:Display_2 5:Sort 6:Reverse 7:Concatenate 8:Exit : 2
Enter data : 40

Menu 1:Insert_1 2:Insert_2 3:Display_1 4:Display_2 5:Sort 6:Reverse 7:Concatenate 8:Exit : 3
Elements of linked list 1:
20      10
Menu 1:Insert_1 2:Insert_2 3:Display_1 4:Display_2 5:Sort 6:Reverse 7:Concatenate 8:Exit : 4
Elements of linked list 2:
40      30
Menu 1:Insert_1 2:Insert_2 3:Display_1 4:Display_2 5:Sort 6:Reverse 7:Concatenate 8:Exit : 6

Menu 1:Insert_1 2:Insert_2 3:Display_1 4:Display_2 5:Sort 6:Reverse 7:Concatenate 8:Exit : 3
Elements of linked list 1:
10      20
Menu 1:Insert_1 2:Insert_2 3:Display_1 4:Display_2 5:Sort 6:Reverse 7:Concatenate 8:Exit : 5

Menu 1:Insert_1 2:Insert_2 3:Display_1 4:Display_2 5:Sort 6:Reverse 7:Concatenate 8:Exit : 3
Elements of linked list 1:
10      20
Menu 1:Insert_1 2:Insert_2 3:Display_1 4:Display_2 5:Sort 6:Reverse 7:Concatenate 8:Exit : 7

Menu 1:Insert_1 2:Insert_2 3:Display_1 4:Display_2 5:Sort 6:Reverse 7:Concatenate 8:Exit : 3
Elements of linked list 1:
10      20      40      30
Menu 1:Insert_1 2:Insert_2 3:Display_1 4:Display_2 5:Sort 6:Reverse 7:Concatenate 8:Exit : 8
```

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

```
#include <stdio.h>

struct node{
int data;
struct node* next;
};
void push(struct node**head,int val){

struct node* newnode=(struct node*)malloc(sizeof(struct node)); newnode->data =
val;
newnode->next =*head;
*head = newnode;
}
void pop(struct node**head){ if (*head == NULL) { printf("Stack is empty.\n"); return;
}
struct node* temp = *head;
*head =(*head)->next;
printf("%d is poped.",temp->data); free(temp);
}
void display(struct Node* head) { struct node* temp = head;
if (temp == NULL) { printf("Stack is empty.\n"); return;
}
printf("Elements of Stack are:\n"); while (temp != NULL) {

printf("%d\t",temp->data); temp = temp->next;
}
}
int main(){
struct node* head=NULL; int ch,val;
while(ch!=4){
printf("\nMenu : 1:Push 2:Pop 3:Display 4:Exit\n"); scanf("%d",&ch);
switch(ch){ case 1:
printf("Enter the value : "); scanf("%d",&val); push(&head,val);
```

```
break; case 2:
pop(&head); break;
case 3: display(head); break;
case 4:

return 0; default:
printf("Invalid Choice\n"); break;
}
}
}
```

```
Menu : 1:Push 2:Pop 3:Display 4:Exit
1
Enter the value : 10

Menu : 1:Push 2:Pop 3:Display 4:Exit
1
Enter the value : 20

Menu : 1:Push 2:Pop 3:Display 4:Exit
2
20 is poped.
Menu : 1:Push 2:Pop 3:Display 4:Exit
3
Elements of Stack are:
10
Menu : 1:Push 2:Pop 3:Display 4:Exit
4
```

```
#include <stdio.h>
struct node {
int data;
struct node* next;
};
void insert(struct node** head, int val) {
struct node* newnode = (struct node*)malloc(sizeof(struct node)); struct node* temp
= *head;
newnode->data = val; newnode->next = NULL; if (*head == NULL) {
*head = newnode; return;
}
while (temp->next != NULL) { temp = temp->next;
}
temp->next = newnode;
}
void delete1(struct node** head) { if (*head == NULL) { printf("Queue is empty.\n");

return;
}
struct node* temp = *head;
*head = (*head)->next;
printf("%d is deleted.",temp->data); free(temp);
}
void display(struct Node* head) { struct node* temp = head;
if (temp == NULL) { printf("Queue is empty.\n"); return;
}
printf("Elements of Queue are:\n"); while (temp != NULL) { printf("%d\t",temp->data);
temp = temp->next;
}
}
int main(){
struct node* head=NULL; int ch,val;
while(ch!=4){
```

```
printf("\nMenu : 1:Insert 2:Delete 3:Display 4:Exit\n"); scanf("%d",&ch);
switch(ch){ case 1:
printf("Enter the value : "); scanf("%d",&val); insert(&head,val);
break; case 2:
delete1(&head); break;
case 3: display(head); break;
case 4:
return 0; default:
printf("Invalid Choice\n"); break;
}
}
}
```

```
Menu : 1:Insert 2:Delete 3:Display 4:Exit
1
Enter the value : 10

Menu : 1:Insert 2:Delete 3:Display 4:Exit
1
Enter the value : 20

Menu : 1:Insert 2:Delete 3:Display 4:Exit
2
10 is deleted.
Menu : 1:Insert 2:Delete 3:Display 4:Exit
2
20 is deleted.
Menu : 1:Insert 2:Delete 3:Display 4:Exit
2
Queue is empty.

Menu : 1:Insert 2:Delete 3:Display 4:Exit
3
Queue is empty.

Menu : 1:Insert 2:Delete 3:Display 4:Exit
4
```

7a) WAP to Implement doubly link list with primitive operations
Create a doubly linked list.
Insert a new node to the left of the node.
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 *s1 = NULL;
struct node *insert_begin(struct node *start)
{
struct node *temp;
temp = (struct node *)malloc(sizeof(struct node)); printf("Enter the value to be
inserted\n"); scanf("%d", &temp->data);
temp->next = NULL;

temp->prev = NULL; if (start == NULL)
{
start = temp;



}
else
{
temp->next = start; start->prev = temp; start = temp;
}
return start;
}
struct node *insert_end(struct node *start)
{
struct node *temp;
```

```
temp = (struct node *)malloc(sizeof(struct node)); printf("Enter the value to be
inserted\n"); scanf("%d", &temp->data);
temp->next = NULL; temp->prev = NULL;
if (start == NULL)

{
start = temp;
}
else



{
struct node *ptr; ptr = start;
while (ptr->next != NULL)
{
ptr = ptr->next;
}
ptr->next = temp; temp->prev = ptr;
}
return start;
}
struct node *insert_pos(struct node *start)
{
int n;
struct node *temp; struct node *ptr = start;
temp = (struct node *)malloc(sizeof(struct node));

printf("Enter the value to be inserted\n"); scanf("%d", &temp->data);
temp->next = NULL; temp->prev = NULL;
printf("Enter the position where the node has to be inserted\n"); scanf("%d", &n);

if (n == 1)
{
temp->next = start; start = temp;
}
```

```
else
{
for (int i = 1; i < n - 1; i++)
{
ptr = ptr->next; if (ptr == NULL)
{
printf("the node cant be inserted at position -%d\n", n);
}
}
temp->next = ptr->next;

ptr->next = temp;
}
return start;
}
struct node *delete_begin(struct node *start)
{
if (start == NULL)
{


printf("Empty list\n"); return start;
}
else if (start->next == NULL)
{
printf("Value deleted=%d", start->data); free(start);
start = NULL;
}
else
{
struct node *ptr; ptr = start;
start = start->next;

printf("Value deleted=%d", ptr->data); free(ptr);
start->prev = NULL;
```

```
}
return start;
}
struct node *delete_end(struct node *start)
{
struct node *ptr = start; if (start == NULL)

{
printf("\n the list is empty\n"); return start;
}
else if (start->next == NULL)
{
printf("The value deleted=%d", start->data); free(start);
start = NULL;
}
else
{

while (ptr->next != NULL)
{
ptr = ptr->next;
}
printf("the value deleted=%d", ptr->data); ptr->prev->next = NULL;
free(ptr);
}
return start;
}
struct node *delete_pos(struct node *start)
{
int n;


struct node *ptr = start; if (start == NULL)
{
printf("\n The list is empty\n"); return start;
```

```
}
printf("Enter the position to be deleted\n"); scanf("%d", &n);
if (n == 1)

{
printf("The value deleted=%d", start->data); start = start->next;
start->prev = NULL; free(ptr);
}
else
{
for (int i = 1; i < n - 1; i++)
{
ptr = ptr->next;
}
printf("The value deleted=%d", ptr->next->data); ptr->next = ptr->next->next;
free(ptr->next->prev); ptr->next->prev = ptr;

}
return start;
}
void display(struct node *start)
{
struct node *ptr;

ptr = start;
if (start == NULL)
{
printf("\n list is empty\n");
}
else
{
while (ptr != NULL)
{
printf("%d\n", ptr->data); ptr = ptr->next;
}
```

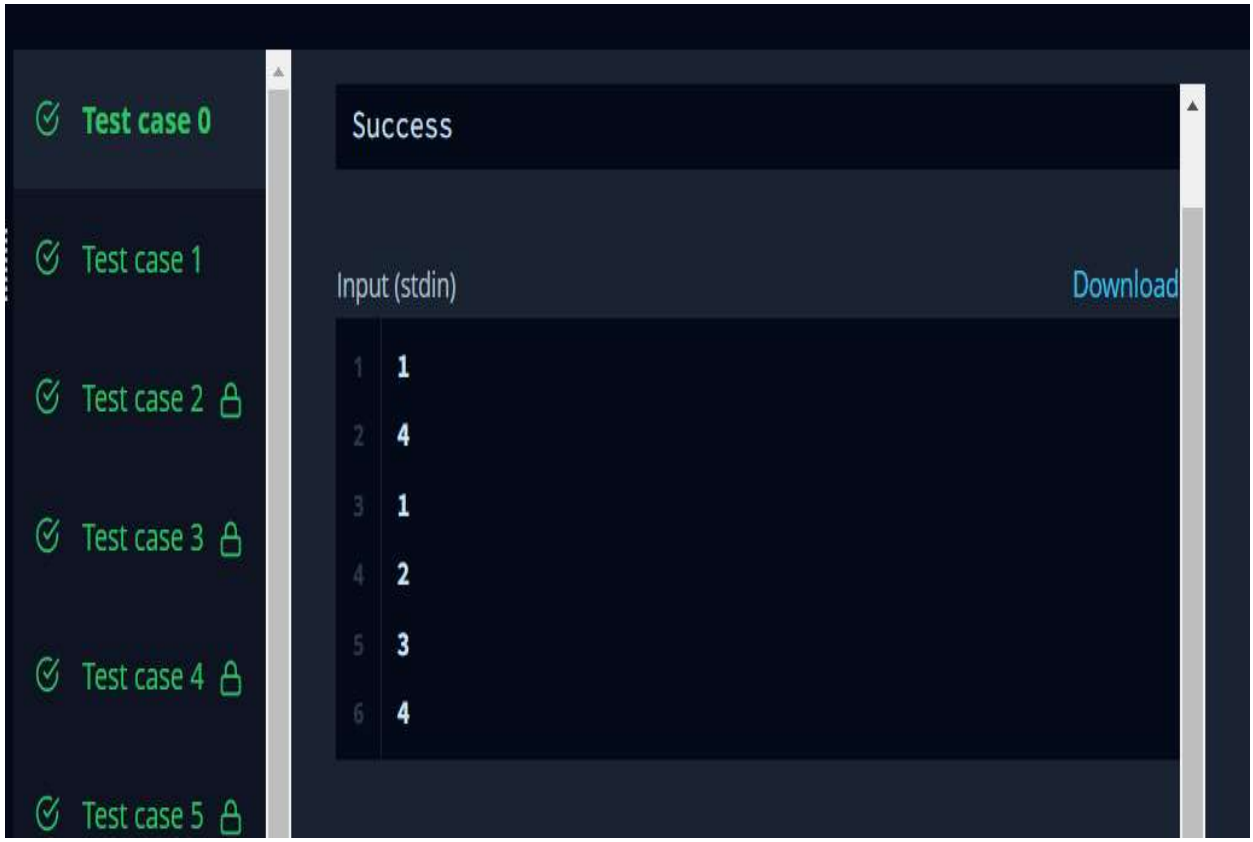
```
}
}
int main()
{
int choice; while (1)

{
printf("\n 1. to add in the beginning \n 2. to add at the end\n 3. to insert at the given position\n 4.to delete at the beginning\n 5.to delete at the end\n 6.to delete at a specific position\n 7.display \n 8.Exit\n");

scanf("%d", &choice); switch (choice)
{
case 1:
s1 = insert_begin(s1); break;
case 2:
s1 = insert_end(s1); break;
case 3:
s1 = insert_pos(s1); break;
case 4:
s1 = delete_begin(s1); break;
case 5:
s1 = delete_end(s1); break;
case 6:
s1 = delete_pos(s1); break;
case 7:
display(s1);


break; case 8: exit(0);
}
}
}
```

```
 1. to add in the beginning
 2. to add at the end
 3. to insert at the given position
 4.to delete at the beginning
 5.to delete at the end
 6.to delete at a specific position
 7.display
 8.Exit
1
Enter the value to be inserted
10

 1. to add in the beginning
 2. to add at the end
 3. to insert at the given position
 4.to delete at the beginning
 5.to delete at the end
 6.to delete at a specific position
 7.display
 8.Exit
1
Enter the value to be inserted
20

 1. to add in the beginning
 2. to add at the end
 3. to insert at the given position
 4.to delete at the beginning
 5.to delete at the end
 6.to delete at a specific position
 7.display
 8.Exit
4
Value deleted=20
 1. to add in the beginning
 2. to add at the end
 3. to insert at the given position
```

```
 2. to add at the end
 3. to insert at the given position
 4.to delete at the beginning
 5.to delete at the end
 6.to delete at a specific position
 7.display
 8.Exit
7
10

 1. to add in the beginning
 2. to add at the end
 3. to insert at the given position
 4.to delete at the beginning
 5.to delete at the end
 6.to delete at a specific position
 7.display
 8.Exit
1
Enter the value to be inserted
40

 1. to add in the beginning
 2. to add at the end
 3. to insert at the given position
 4.to delete at the beginning
 5.to delete at the end
 6.to delete at a specific position
 7.display
 8.Exit
7
40
10

 1. to add in the beginning
 2. to add at the end
 3. to insert at the given position
 4.to delete at the beginning
```

```
10

 1. to add in the beginning
 2. to add at the end
 3. to insert at the given position
 4.to delete at the beginning
 5.to delete at the end
 6.to delete at a specific position
 7.display
 8.Exit
6
Enter the position to be deleted
1
The value deleted=40
 1. to add in the beginning
 2. to add at the end
 3. to insert at the given position
 4.to delete at the beginning
 5.to delete at the end
 6.to delete at a specific position
 7.display
 8.Exit
7
10

 1. to add in the beginning
 2. to add at the end
 3. to insert at the given position
 4.to delete at the beginning
 5.to delete at the end
 6.to delete at a specific position
 7.display
 8.Exit
8
```

7b) Program - HackerRank - Reverse Linked List



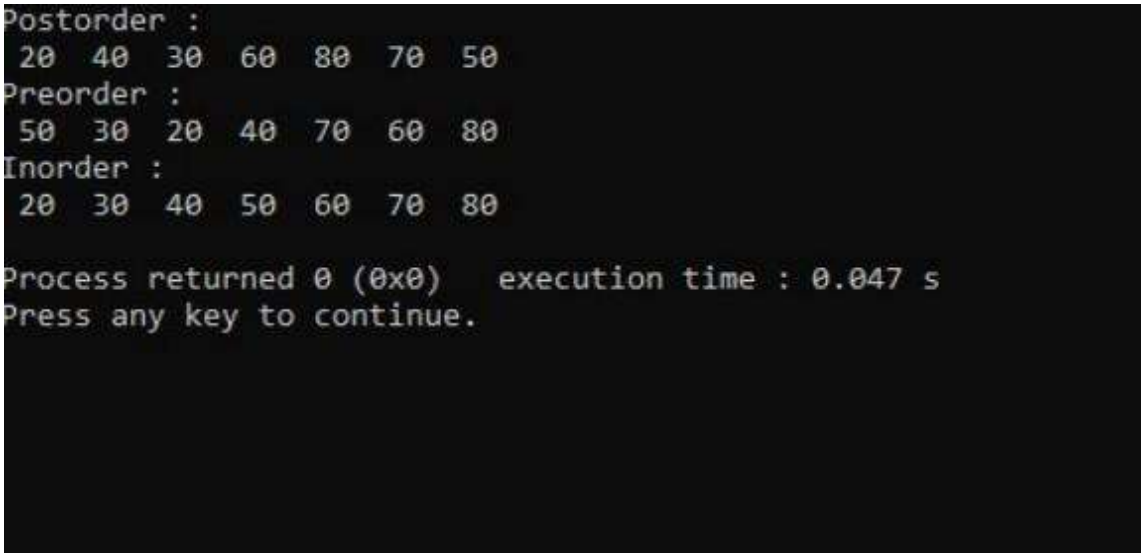
8a) Write a program
To construct a binary Search tree.
To traverse the tree using all the methods i.e., in-order, preorder and post order
To display the elements in the tree.

Code:

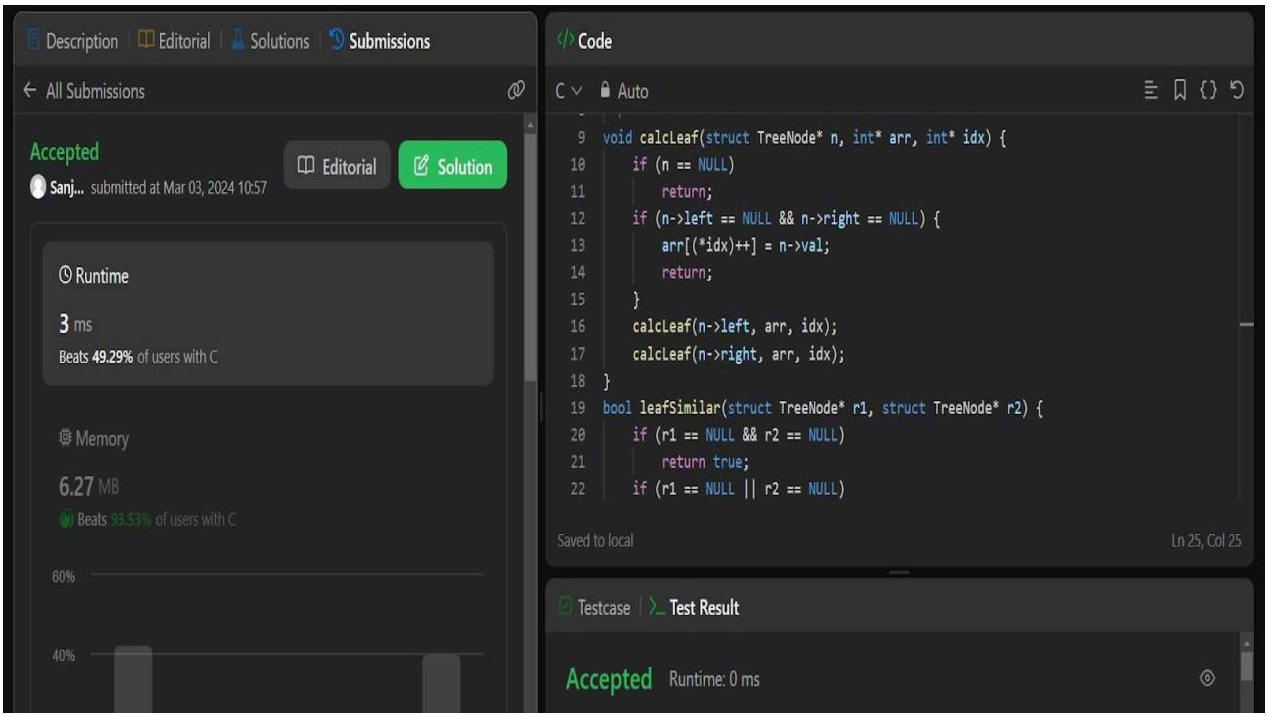
```
#include <stdio.h> #include <stdlib.h> struct Node{
int data;
struct Node *left, *right;
};
struct Node* newnode(int value)
{
struct Node* temp= (struct Node*)malloc(sizeof(struct Node)); temp->data = value;
temp->left = temp->right = NULL; return temp;
}
struct Node* insertNode(struct Node* node, int value)
{
if (node == NULL) {

return newnode(value);
}
if (value < node->data) {


node->left = insertNode(node->left, value);
}
else if (value > node->data) {
node->right = insertNode(node->right, value);
}
return node;
}
void postOrder(struct Node* root)
{
```

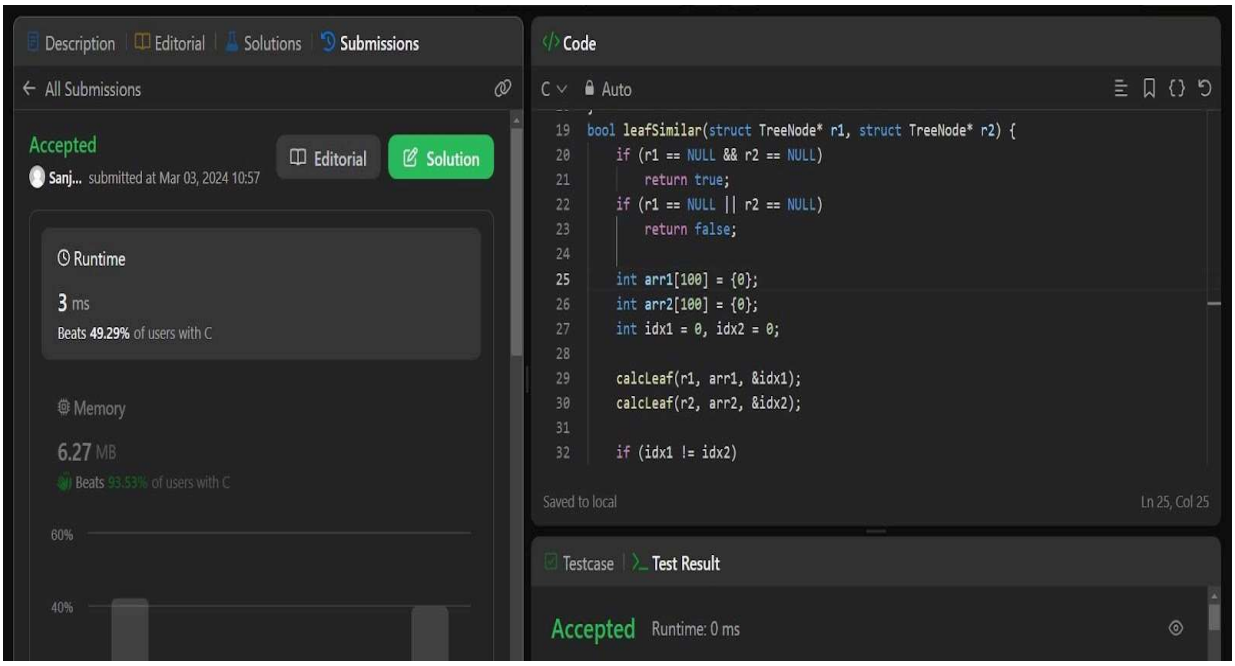
```
if (root != NULL) { postOrder(root->left); postOrder(root->right); printf(" %d ",
root->data);
}
}
void inOrder(struct Node* root)
{
if (root != NULL) { inOrder(root->left); printf(" %d ", root->data);

inOrder(root->right);
}
}


void preOrder(struct Node* root)
{
if (root != NULL) {
printf(" %d ", root->data); preOrder(root->left); preOrder(root->right);
}
}
int main()
{
struct Node* root = NULL; root = insertNode(root, 50); insertNode(root, 30);
insertNode(root, 20);
insertNode(root, 40);
insertNode(root, 70);
insertNode(root, 60);
insertNode(root, 80); printf("Postorder :\n"); postOrder(root);

printf("\n");


printf("Preorder :\n"); preOrder(root); printf("\n"); printf("Inorder :\n"); inOrder(root);
printf("\n");
return 0;
}
```

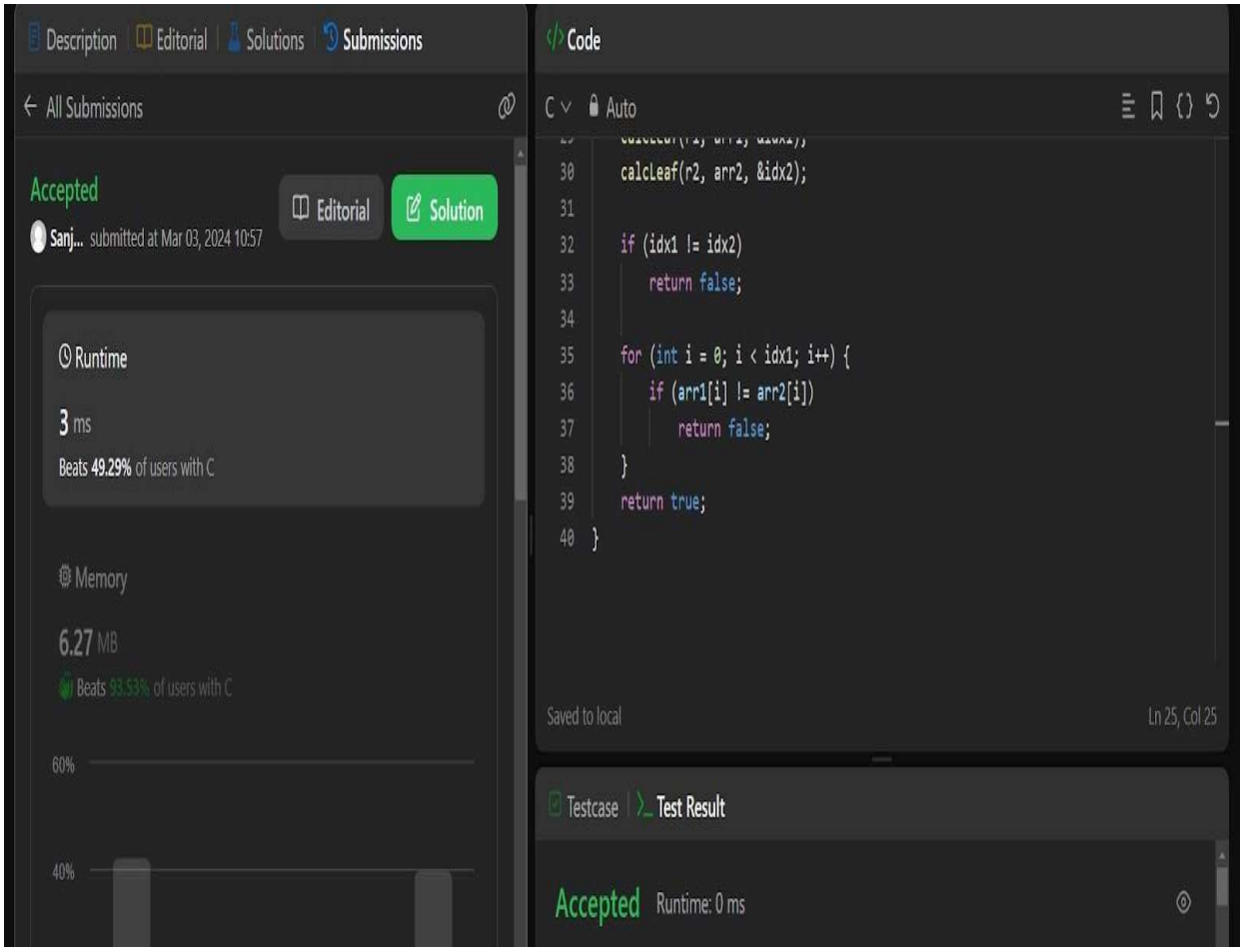
Output:


Postorder :
20 40 30 60 80 70 50
Preorder :
50 30 20 40 70 60 80
Inorder :
20 30 40 50 60 70 80
Process returned 0 (0x0) execution time : 0.047 s
Press any key to continue.

8b) Program - Leetcode platform - Leaf-Similar Trees






Description
Editorial
Solutions
Submissions
All Submissions
Accepted
Sanj... submitted at Mar 03, 2024 10:57
Editorial
Solution
Runtime
3 ms
Beats 49.29% of users with C
Memory
6.27 MB
Beats 93.53% of users with C
60%
40%
Code
C
Auto
30 calcLeaf(r2, arr2, &idx2);
31
32 if (idx1 != idx2)
33 return false;
34
35 for (int i = 0; i < idx1; i++) {
36 if (arr1[i] != arr2[i])
37 return false;
38 }
39 return true;
40 }
Saved to local
Ln 25, Col 25
Testcase
Test Result
Accepted
Runtime: 0 ms

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
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);
}
}
int main() {
int i, j, count = 0;
printf("\n Enter number of vertices:"); scanf("%d", &n);
for (i = 1; i <= n; i++) { reach[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"); return 0;
}
```

Output:

```
 Enter number of vertices:5

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

 1->2
 2->3
 3->5
 5->4

 Graph is connected
Press any key to continue . . .
```

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 TABLE_SIZE 10

int h[TABLE_SIZE]={NULL};

void insert()
{

 int key,index,i,flag=0,hkey;
 printf("\nenter a value to insert into hash table\n");
 scanf("%d",&key);
 hkey=key%TABLE_SIZE;
 for(i=0;i<TABLE_SIZE;i++)
          {

          index=(hkey+i)%TABLE_SIZE;

          if(h[index] == NULL)
          {
          h[index]=key;
          break;
          }

          }

          if(i == TABLE_SIZE)

          printf("\nelement cannot be inserted\n");
}
void search()
{
```

```
 int key,index,i,flag=0,hkey;
 printf("\nenter search element\n");
 scanf("%d",&key);
 hkey=key%TABLE_SIZE;
 for(i=0;i<TABLE_SIZE; i++)
 {
            index=(hkey+i)%TABLE_SIZE;
            if(h[index]==key)
            {
            printf("value is found at index %d",index);
            break;
            }
  }
  if(i == TABLE_SIZE)
            printf("\n value is not found\n");
}
void display()
{

  int i;

  printf("\nelements in the hash table are \n");

  for(i=0;i< TABLE_SIZE; i++)

  printf("\nat index %d \t value =  %d",i,h[i]);

}
main()
{
            int opt,i;
            while(1)
            {
            printf("\nPress 1. Insert\t 2. Display \t3. Search \t4.Exit \n");
            scanf("%d",&opt);
            switch(opt)
            {
            case 1:
            insert();
            break;
            case 2:
            display();
            break;
            case 3:
            search();
            break;
            case 4:exit(0);
            }
```

```
        }
}
```

```
Press 1. Insert    2. Display    3. Search    4.Exit
1

enter a value to insert into hash table
12

Press 1. Insert    2. Display    3. Search    4.Exit
1

enter a value to insert into hash table
13

Press 1. Insert    2. Display    3. Search    4.Exit
1

enter a value to insert into hash table
22

Press 1. Insert    2. Display    3. Search    4.Exit
2

elements in the hash table are

at index 0      value =  0
at index 1      value =  0
at index 2      value =  12
at index 3      value =  13
at index 4      value =  22
at index 5      value =  0
at index 6      value =  0
at index 7      value =  0
at index 8      value =  0
at index 9      value =  0
Press 1. Insert    2. Display    3. Search    4.Exit
3

enter search element
12
value is found at index 2
Press 1. Insert    2. Display    3. Search    4.Exit
2 3

enter search element
23

 value is not found

Press 1. Insert    2. Display    3. Search    4.Exit
4
```