VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

RACHANA H D
1BM22CS212

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
Dec 2023- March 2024

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

This is to certify that the Lab work entitled **"DATA STRUCTURES"** carried out by RACHANA H D **(1BM22CS212)**, who is a 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 2023-24. The Lab report has been approved as it satisfies the academic requirements in respect of Data structures Lab - **(23CS3PCDST) work** prescribed for the said degree.

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

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

Index Sheet

Course outcomes:

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

Lab program 1:

Write a program to simulate the working of stack using an array with the following:
a) Push
b) Pop
c) Display
The program should print appropriate messages for stack overflow, stack underflow.

```
#include <stdio.h>
#include<stdlib.h>
#define STACK_SIZE 5
void push(int st[],int *top)
{
	int item;
	if(*top==STACK_SIZE-1)
		printf("Stack overflow\n");
	else
	{
		printf("\nEnter an item :");
		scanf("%d",&item);
		(*top)++;
		st[*top]=item;
	}
}
void pop(int st[],int *top)
{
	if(*top==-1)
		printf("Stack underflow\n");
	else
	{
		printf("\n%d item was deleted",st[(*top)--]);
	}
}
void display(int st[],int *top)
{
	int i;
	if(*top==-1)
		printf("Stack is empty\n");
	for(i=0;i<=*top;i++)
		printf("%d\t",st[i]);
}
void main()
{
	int st[10],top=-1, c,val_del;
	while(1)
	{
		printf("\n1. Push\n2. Pop\n3. Display\n");
		printf("\nEnter your choice :");
		scanf("%d",&c);
		switch(c)
		{
```

```
                    case 1: push(st,&top);
                            break;
                    case 2: pop(st,&top);
                            break;
                    case 3: display(st,&top);
                            break;
                    default: printf("\nInvalid choice!!!");
                            exit(0);
                }
        }
}
```

Output:

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

PS D:\jyothika\DST> cd "d:\jyothika\DST\" ; if ($?) { gcc 1.c -o 1 } ; if ($?) { .\1 }

1. Push
2. Pop
3. Display

Enter your choice :1

Enter an item :12

1. Push
2. Pop
3. Display

Enter your choice :1

Enter an item :65

1. Push
2. Pop
3. Display

Enter your choice :1

Enter an item :45

1. Push
2. Pop
3. Display

Enter your choice :1
Stack overflow
```

```
1. Push
2. Pop
3. Display

Enter your choice :2

45 item was deleted
1. Push
2. Pop
3. Display

Enter your choice :2

65 item was deleted
1. Push
2. Pop
3. Display

Enter your choice :3
12
1. Push
2. Pop
3. Display

Enter your choice :2

12 item was deleted
1. Push
2. Pop
3. Display

Enter your choice :2
Stack underflow

1. Push
2. Pop
3. Display

Enter your choice :4

Invalid choice!!!
```

WEEK 2:

Program 2: Write a program 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).

Solution:

```
#include <stdio.h>
#include <ctype.h>
#define SIZE 50
char stack[SIZE];
int top=-1;

void push(char elem)
{
stack [++top]=elem;
}

char pop()
{
return(stack [top--]);
}
int pr(char symbol)
{
if (symbol == '^')
{
return(3);
```

```
}
else if(symbol=='*' || symbol=='/')
{
return(2);
}
else if(symbol == '+' || symbol == '-')
{
return(1);
}
else
{
return(0);
}
}
void main()
{
char infix[50], postfix[50],ch,elem;
int i=0,k=0;
printf("Enter Infix Expression : ");
scanf("%s", infix);
push('#');
while ((ch=infix[i++]) != '\0')
{
if( ch == '(')
push(ch);
else if(isalnum(ch))
postfix[k++]=ch;
```

```
else
{
if( ch == ')')
{
      while (stack[top] != '(')
      postfix[k++]=pop();
      elem=pop();
}
else
{
    while( pr(stack [top]) >= pr(ch) )
    postfix [k++]=pop();
    push(ch);
}
}
}
while (stack[top] != '#')
postfix[k++]=pop();
postfix[k]='\0';
printf("\nPostfix Expression = %s\n", postfix);
}
```

Output:

```
Enter Infix Expression : 2+2*(2*2)

Postfix Expression = 2222**+
```

```
Enter Infix Expression : (4/2)+8*(6/3)

Postfix Expression = 42/863/*+
```

WEEK 3:

Program 3a: Write a program 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

Solution:

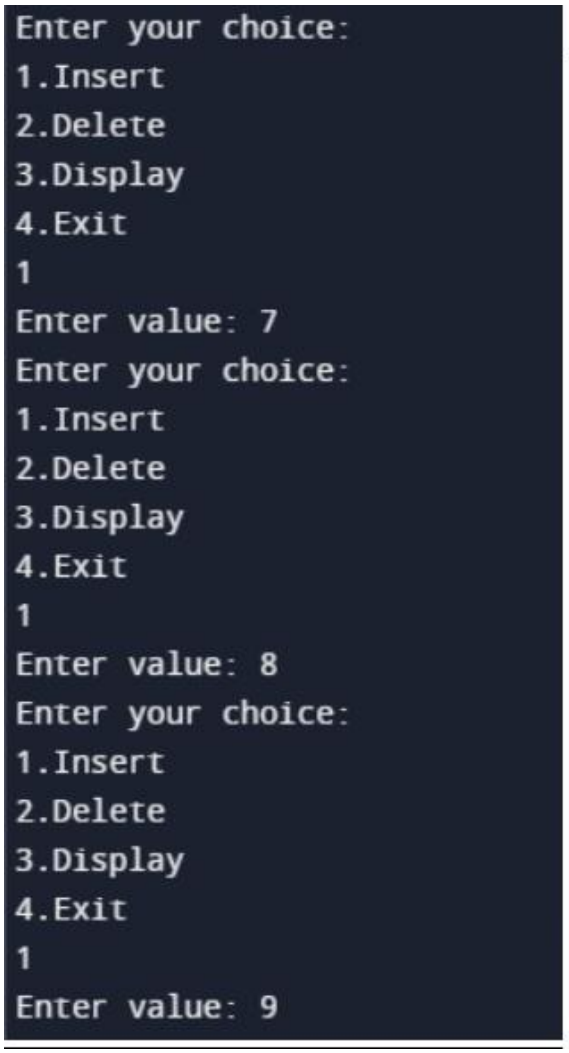
```
#include<stdio.h>
#include<ctype.h>
#include <stdlib.h>
#define s 5
int queue[s], f=-1, r=-1;

int isfull()
{
   if(f==(s-1))
      return 1;
   else
      return 0;
}
int isempty()
{
   if(f==-1||r==-1)
      return 1;
   else
      return 0;
}
void insert()
{
```

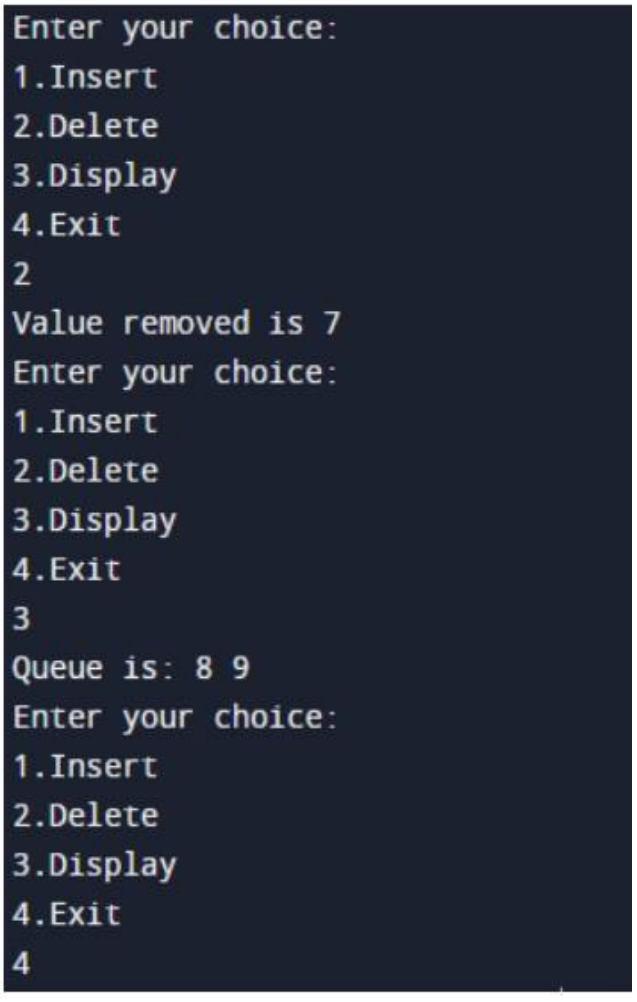
```
    int i;
    printf("Enter value: ");
    scanf("%d",&i);
    if(isfull())
        printf("Queue is full");
    else if(f==-1)
    {
        f=0;
        r=0;
    }
    else
        r=r+1;
    queue[r]=i;
}
void qdelete()
{
    if(isempty())
        printf("Queue is empty");
    else if(f==r)
    {
        f=-1;
        r=-1;
    }
    else
    {
        printf("Value removed is %d\n",queue[f]);
        f=f+1;
```

```
    }
}
void display()
{
    int i;
    if(isempty())
        printf("Queue is empty");
    else
    {
        printf("Queue is: ");
        for(i=f; i<=r;i++)
            printf("%d\t",queue[i]);
        printf("\n");
    }
}
int main()
{
    int ch;
    while(1)
    {
        printf("Enter your choice:\n1.Insert\n2.Delete\n3.Display\n4.Exit\n");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1: insert(); break;
            case 2: qdelete(); break;
            case 3: display(); break;
```

```
        case 4: {
                printf("Name: RACHANA H D\tUSN:1BM22CS212");
                exit(0);
               }
      }
   }
   return 0;
}
```

Output:





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

Solution:

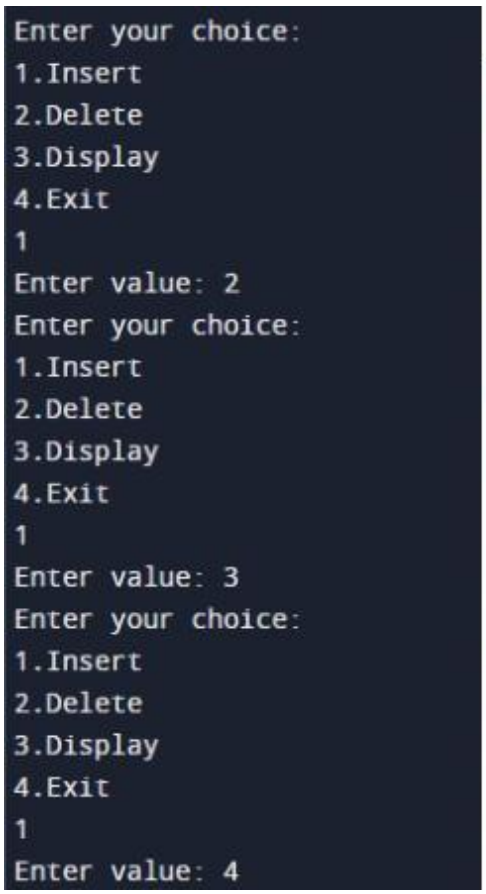
```
#include<stdio.h>
#include<ctype.h>
#include <stdlib.h>
#define s 5
int queue[s], f=-1, r=-1;

int isfull()
{
   if(f==(r+1)||f==0 && r==(s-1))
      return 1;
   else
      return 0;
}
int isempty()
{
   if(f==-1||f>r)
      return 1;
   else
      return 0;
}
void insert()
{
   int i;
```

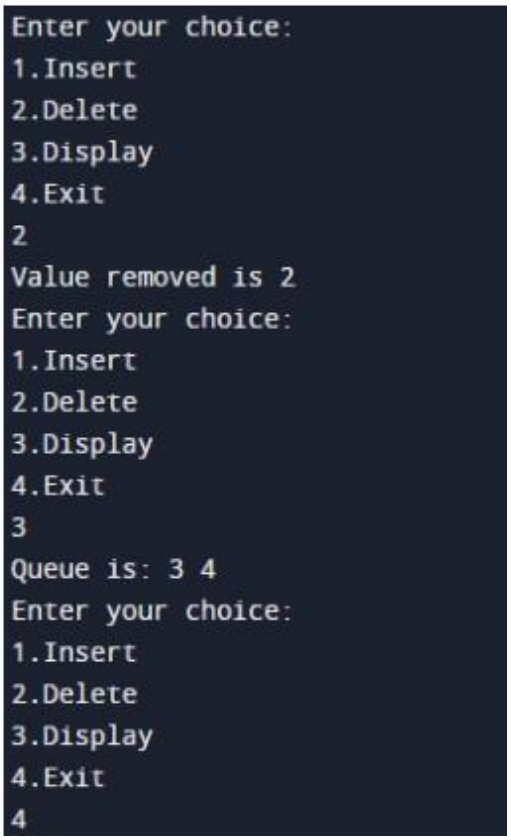
```
    printf("Enter value: ");
    scanf("%d",&i);
    if(isfull())
        printf("Queue is full");
    else if(isempty())
    {
        f=0;
        r=0;
    }
    else
        r=(r+1)%s;
    queue[r]=i;
}
void qdelete()
{
    if(isempty())
        printf("Queue is empty");
    else if(f==r)
    {
        f=-1;
        r=-1;
    }
    else
    {
        printf("Value removed is %d\n",queue[f]);
        f=(f+1)%s;
    }
```

```
}
void display()
{
   int i;
   if(isempty())
      printf("Queue is empty");
   else
   {
      printf("Queue is: ");
      for(i=f; i!=r;i=(i+1)%s)
         printf("%d\t",queue[i]);
      printf("%d",queue[i]);
      printf("\n");
   }
}
int main()
{
   int ch;
   while(1)
   {
      printf("Enter your choice:\n1.Insert\n2.Delete\n3.Display\n4.Exit\n");
      scanf("%d",&ch);
      switch(ch)
      {
         case 1: insert(); break;
         case 2: qdelete(); break;
         case 3: display(); break;
```

```
        case 4: {
                printf("Name: RACHANA H D\tUSN:1BM22CS212");
                exit(0);
               }
      }
   }
   return 0;
}
```

OUTPUT:





Lab Program 4: Write a program to implement Singly Linked List with the following operations

1. **Create a linked list.**
2. **Insertion of a node at the first position, at any position, and at the end of the list.**
3. **Display the contents of the linked list.**

Solution:

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
   int data;
   struct Node *next;
};
struct Node *head;
void insert_begin()
{
   struct Node *ptr;
   int n;
   ptr=(struct Node *)malloc(sizeof(struct Node));
   if(ptr==NULL)
   {
      printf("\nOverflow");
   }
   else
   {
      printf("Enter value: ");
```

```
        scanf("%d",&n);
        ptr->data=n;
        ptr->next=head;
        head=ptr;
    }
}
void insert_end()
{
    struct Node *ptr, *temp;
    int n;
    ptr=(struct Node*)malloc(sizeof(struct Node));
    if(ptr==NULL)
    {
        printf("\nOverflow");
    }
    else
    {
        printf("Enter value: ");
        scanf("%d",&n);
        ptr->data=n;
        if(head==NULL)
        {
            ptr->next=NULL;
            head=ptr;
        }
        else
        {
```

```
            temp=head;
            while(temp->next!=NULL)
               temp=temp->next;
            temp->next=ptr;
            ptr->next=NULL;
        }
    }
}
void insert_random()
{
    int i, loc, n;
    struct Node *ptr, *temp;
    ptr=(struct Node*)malloc(sizeof(struct Node));
    if(ptr==NULL)
    {
        printf("\nOverflow");
    }
    else
    {
        printf("Enter value: ");
        scanf("%d",&n);
        ptr->data=n;
        printf("Enter position: ");
        scanf("%d",&loc);
        if(loc == 0)
        {
            ptr->next = head;
```

```
            head = ptr;
            return;
        }
        temp=head;
        for(i=0; i<loc-1; i++)
        {
            if(temp==NULL)
            {
                printf("Cannot insert at position\n");
                return;
            }
            temp=temp->next;
        }
        ptr->next=temp->next;
        temp->next=ptr;
        }
}
void display()
{
    struct Node * ptr;
    ptr=head;
    printf("List is: ");
    while(ptr!=NULL)
    {
        printf("%d\t", ptr->data);
        ptr=ptr->next;
    }
```

```
    printf("\n");
}
int main()
{
    int ch=0;
    while(1)
    {
        printf("Enter your choice:\n1.Insert at the beginning \n2.Insert at the end \n3.Insert at specified position\n4.Display\n5.Exit\n");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:insert_begin(); break;
            case 2:insert_end(); break;
            case 3:insert_random(); break;
            case 4:display(); break;
            case 5:{
                      printf("Name: RACHANA H D\tUSN:1BM22CS212");
                       exit(0);
                   }
        }
    }return 0;
}
```

Output:

```
Enter your choice:
1.Insert at the beginning
2.Insert at the end
3.Insert at specified position
4.Display
5.Exit
1
Enter value: 2
Enter your choice:
1.Insert at the beginning
2.Insert at the end
3.Insert at specified position
4.Display
5.Exit
4
List is: 2
Enter your choice:
1.Insert at the beginning
2.Insert at the end
3.Insert at specified position
4.Display
5.Exit
1
Enter value: 2
Enter your choice:
1.Insert at the beginning
2.Insert at the end
3.Insert at specified position
4.Display
5.Exit
2
Enter value: 3
```

```
Enter your choice:
1.Insert at the beginning
2.Insert at the end
3.Insert at specified position
4.Display
5.Exit
4
List is: 2  2  3
Enter your choice:
1.Insert at the beginning
2.Insert at the end
3.Insert at specified position
4.Display
5.Exit
3
Enter value: 1
Enter position: 0
Enter your choice:
1.Insert at the beginning
2.Insert at the end
3.Insert at specified position
4.Display
5.Exit
4
List is: 1  2  2  3
Enter your choice:
1.Insert at the beginning
2.Insert at the end
3.Insert at specified position
4.Display
5.Exit
5
```

Program - Leetcode platform

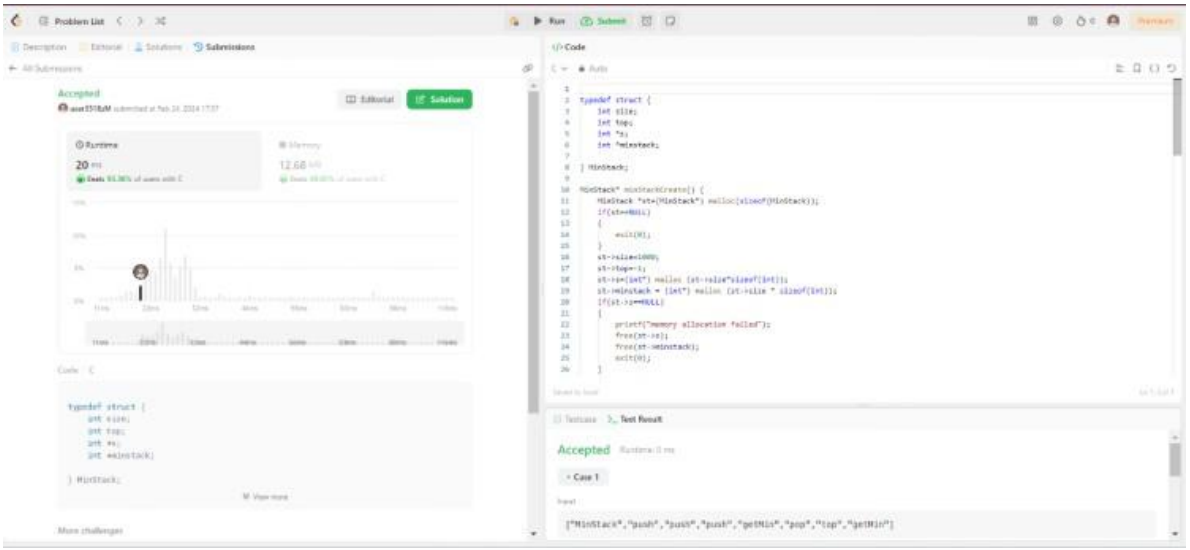
```
typedef struct {
    int size;
    int top;
    int *s;
    int *minstack;

} MinStack;


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

```
    }
    return st;
}

void minStackPush(MinStack* obj, int val) {
    if(obj->top==obj->size-1)
        printf("stack is overflow");
    else{
        obj->top++;
        obj->s[obj->top]=val;
        if (obj->top == 0 || val < obj->minstack[obj->top - 1]) {
            obj->minstack[obj->top] = val;
        } else {
            obj->minstack[obj->top] = obj->minstack[obj->top - 1];
        }
    }
}

void minStackPop(MinStack* obj) {
    int value;
    if(obj->top==-1)
        printf("underflow");
    else
    {
        value=obj->s[obj->top];
```

```
        obj->top--;
    }
}

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

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

```
    }
}

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

Output:



Lab Program 5:

Write a program to implement a Singly Linked List with the following operations: deletion of the first element,specified element, and last element in the list.

Solution:

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
   int data;
   struct Node *next;
};
struct Node *head;
void insert_begin(int n)
{
   struct Node *ptr;
   ptr=(struct Node *)malloc(sizeof(struct Node));
   if(ptr==NULL)
   {
      printf("\nOverflow");
   }
   else
   {
      ptr->data=n;
      ptr->next=head;
      head=ptr;
   }
```

```
}

void delete_begin()
{
   struct Node *ptr;
   if(head==NULL)
   {
      printf("List is empty\n");
   }
   else
   {
      ptr=head;
      head=ptr->next;
      free(ptr);
      printf("Deleted at the start\n");
   }
}
void delete_end()
{
   struct Node *ptr,*ptr1;
   if(head == NULL)
   {
      printf("List is empty\n");
   }
   else if(head -> next == NULL)
   {
      free(head);
```

```
        head = NULL;
        printf("Only node of the list deleted\n");
    }
    else
    {
        ptr = head;
        while(ptr->next != NULL)
        {
            ptr1 = ptr;
            ptr = ptr ->next;
        }
        ptr1->next = NULL;
        free(ptr);
        printf("Deleted Node from the last\n");
    }
}
void delete_specified()
{
    struct Node *ptr, *ptr1;
    int loc,i;
    printf("Enter position: ");
    scanf("%d",&loc);
    ptr=head;
    for(i=0;i<loc;i++)
    {
        ptr1 = ptr;
        ptr = ptr->next;
```

```
        if(ptr == NULL)
        {
            printf("There are less than %d elements in the list\n",loc);
            return;
        }
    }
    ptr1 ->next = ptr ->next;
    free(ptr);
    printf("Deleted %d node\n",loc);
}
void display()
{
    struct Node * ptr;
    ptr=head;
    printf("List is: ");
    while(ptr!=NULL)
    {
        printf("%d\t", ptr->data);
        ptr=ptr->next;
    }
    printf("\n");
}
int main()
{
    insert_begin(8);
    insert_begin(10);
    insert_begin(12);
```

```
    insert_begin(14);
    insert_begin(16);
    display();
    int ch=0;
    while(1)
    {
        printf("Enter your choice:\n1.Delete at the beginning \n2.Delete at End\n3.Delete at specified position \n4.Display\n5.Exit\n");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:delete_begin(); display(); break;
            case 2:delete_end(); display(); break;
            case 3:delete_specified(); display(); break;
            case 4:display(); break;
            case 5:{
                                printf("Name: RACHANA H D\tUSN:1BM22CS212");
                                exit(0);
                }
        }
    }return 0;
}
```

Output:

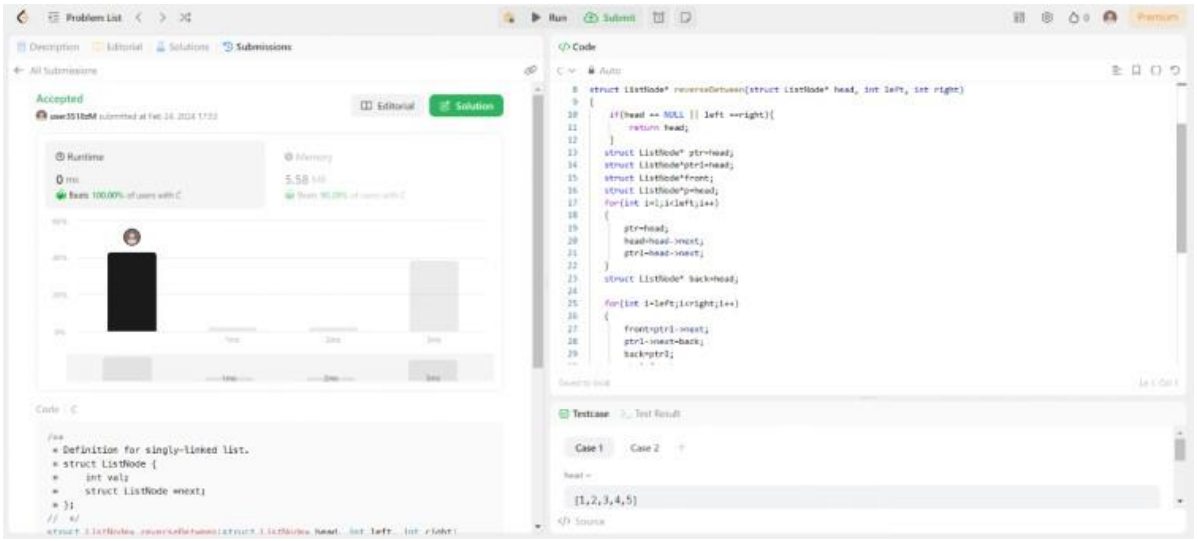
```
List is: 16 14  12  10  8
Enter your choice:
1.Delete at the beginning
2.Delete at End
3.Delete at specified position
4.Display
5.Exit
1
Deleted at the start
List is: 14 12  10  8
Enter your choice:
1.Delete at the beginning
2.Delete at End
3.Delete at specified position
4.Display
5.Exit
2
Deleted Node from the last
List is: 14 12  10
```

```
Enter your choice:
1.Delete at the beginning
2.Delete at End
3.Delete at specified position
4.Display
5.Exit
3
Enter position: 2
Deleted 2 node
List is: 14 12
Enter your choice:
1.Delete at the beginning
2.Delete at End
3.Delete at specified position
4.Display
5.Exit
5
```

Program - Leetcode platform

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

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


Problem List
Run
Submit
Premium
Description
Editorial
Solutions
Submissions
All Submissions
Accepted
Editorial
Solution
Runtime
0 ms
Memory
5.58 MB
Code
C
Auto
struct ListNode* reverseBetween(struct ListNode* head, int left, int right)
{
if(head == NULL || left ==right){
return head;
}
struct ListNode* ptr=head;
struct ListNode*ptr1=head;
struct ListNode*front;
struct ListNode*p=head;
for(int i=1;i<left;i++)
{
ptr=head;
head=head->next;
ptr1=head->next;
}
struct ListNode* back=head;
for(int i=left;i<right;i++)
{
front=ptr1->next;
ptr1->next=back;
back=ptr1;
Testcase
Test Result
Case 1
Case 2
[1,2,3,4,5]
Source
/**
* Definition for singly-linked list.
* struct ListNode {
* int val;
* struct ListNode *next;
* };
*/

Lab Program 6a

Write a program to implement a Single Link List with following operations: Sort the linked list, Reverse the linked list, Concatenation of two linked lists.

Solution:

```
#include<stdio.h>
#include<stdlib.h>
struct node {
   int data;
   struct node *next;
};
void display(struct node *head)
{
   struct node *ptr = head;
   while (ptr != NULL)
   {
      printf("%d\t", ptr->data);
      ptr = ptr->next;
   }
   printf("\n");
}
void sort(struct node **head)
{
   if (*head == NULL)
      return;
   struct node *current, *next;
   int temp;
```

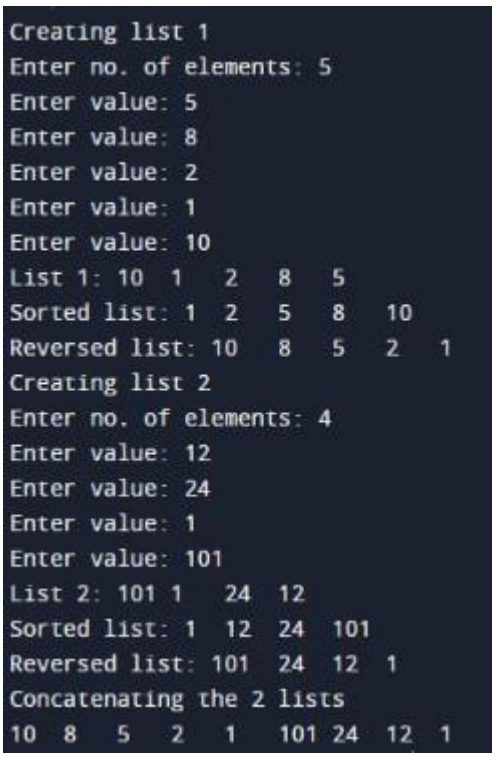
```
    current = *head;
    while (current->next != NULL)
    {
        next = current->next;
        while (next != NULL)
        {
            if (current->data > next->data)
            {
                temp = current->data;
                current->data = next->data;
                next->data = temp;
            }
            next = next->next;
        }
        current = current->next;
    }
}
void reverse(struct node **head)
{
    struct node *cur=*head, *prev=NULL, *next=NULL;
    while(cur!=NULL)
    {
        next=cur->next;
        cur->next=prev;
        prev=cur;
        cur=next;
    }
```

```
    *head=prev;
}
struct node *concatenate(struct node **head1, struct node **head2)
{
    if (*head1 == NULL)
    {
        *head1 = *head2;
        return *head1;
    }
    if (*head2 == NULL)
        return *head1;
    struct node *temp = *head1;
    while (temp->next != NULL)
        temp = temp->next;
    temp->next = *head2;
    return *head1;
}
void PUSH(struct node **head)
{
    struct node *node = (struct node*)malloc(sizeof(struct node));
    if (node == NULL)
    {
        printf("Overflow\n");
        exit(1);
    }
    int n;
    printf("Enter value: ");
```

```
    scanf("%d", &n);
    node->data = n;
    node->next = *head;
    *head = node;
}
int main()
{
    struct node *head1 = NULL, *head2 = NULL;
    int ch;
    printf("Creating list 1\nEnter no. of elements: ");
    int n, i;
    scanf("%d",  &n);
    for (i = 0; i < n; i++)
        PUSH(&head1);
    printf("List 1: ");
    display(head1);
    sort(&head1);
    printf("Sorted list: ");
    display(head1);
    reverse(&head1);
    printf("Reversed list: ");
    display(head1);
    printf("Creating list 2\nEnter no. of elements: ");
    int n1, i1;
    scanf("%d", &n1);
    for (i1 = 0; i1 < n1; i1++)
        PUSH(&head2);
```

```
    printf("List 2: ");
    display(head2);
    sort(&head2);
    printf("Sorted list: ");
    display(head2);
    reverse(&head2);
    printf("Reversed list: ");
    display(head2);
    printf("Concatenating the 2 lists \n");
    struct node *h = concatenate(&head1, &head2);
    display(h);
    return 0;
}
```

Output:



Program 6b: Write a program to implement a Single Link List to simulate Stack & Queue Operations

Solution;

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

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

/* Stack implementation */

void PUSH(struct node **head) {
   struct node *node = (struct node*)malloc(sizeof(struct node));
   int n;
   if (node == NULL) {
      printf("Overflow\n");
   }
   else {
      printf("Enter value: ");
      scanf("%d", &n);
      node->data = n;
      node->next = *head;
      *head = node;
   }
```

```
}

void POP(struct node **head) {
   struct node *node;
   if (*head == NULL) {
      printf("Stack is empty\n");
   }
   else {
      node = *head;
      *head = node->next;
      free(node);
      printf("Node deleted\n");
   }
}

/* Queue implementation */

void enqueue(struct node **head) {
   struct node *node = (struct node*)malloc(sizeof(struct node));
   int n;
   if (node == NULL) {
      printf("Overflow\n");
   }
   else {
      printf("Enter value: ");
      scanf("%d", &n);
      node->data = n;
```

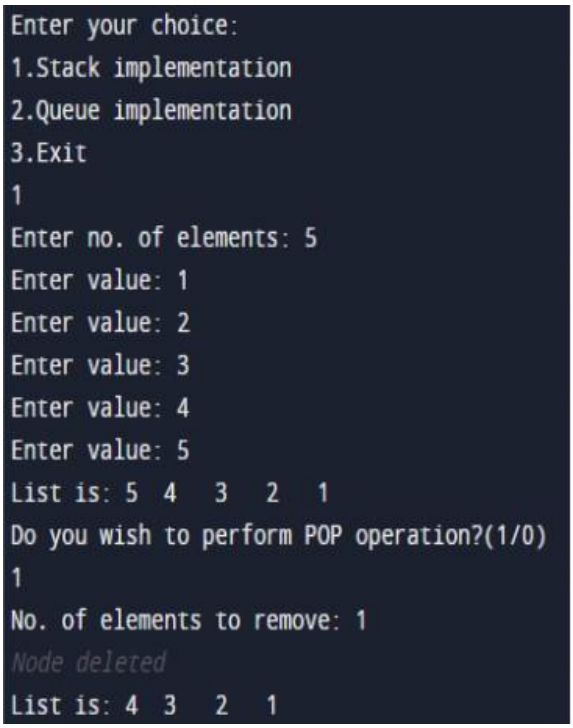
```
        node->next = NULL;
        if (*head == NULL) {
            *head = node;
        }
        else {
            struct node *t;
            t = *head;
            while (t->next != NULL) {
                t = t->next;
            }
            t->next = node;
        }
    }
}

void dequeue(struct node **head) {
    struct node *node;
    if (*head == NULL) {
        printf("Queue is empty\n");
    }
    else {
        node = *head;
        *head = node->next;
        free(node);
        printf("Node deleted\n");
    }
}
```

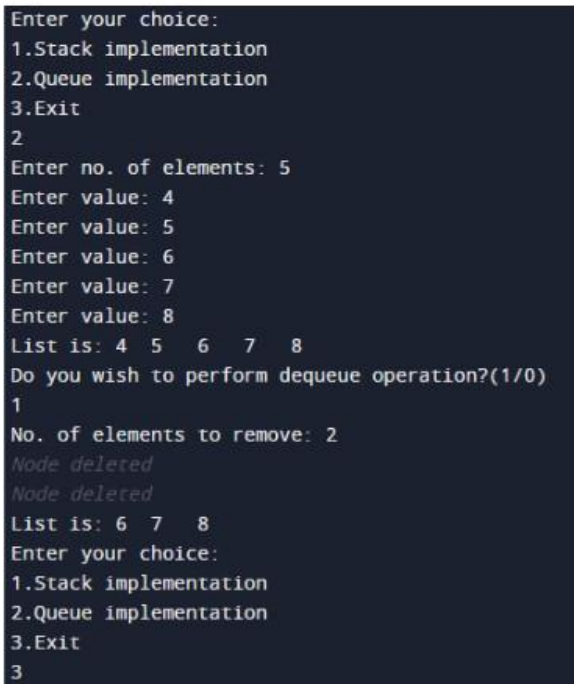
```
void display(struct node *head) {
    struct node *ptr;
    ptr =  head;
    printf("List is: ");
    while (ptr != NULL) {
        printf("%d\t", ptr->data);
        ptr = ptr->next;
    }
    printf("\n");
}

int main() {
    struct node *head1 = NULL, *head2 = NULL;
    int ch;
    while (1) {
        printf("Enter your choice:\n1.Stack implementation\n2.Queue
implementation\n3.Exit\n");
        scanf("%d", &ch);
        switch (ch) {
            case 1: {
                printf("Enter no. of elements: ");
                int n, i, c;
                scanf("%d",  &n);
                for (i = 0; i < n; i++)
                    PUSH(&head1);
                display(head1);
```

```
    printf("Do you wish to perform POP operation?(1/0)\n");
    scanf("%d", &c);
    if (c == 1) {
        printf("No. of elements to remove: ");
        scanf("%d", &n);
        for (i = 0; i < n; i++)
            POP(&head1);
        display(head1);
    }
}
break;
case 2: {
    printf("Enter no. of elements: ");
    int n, i, c;
    scanf("%d",  &n);
    for (i = 0; i < n; i++)
        enqueue(&head2);
    display(head2);
    printf("Do you wish to perform dequeue operation?(1/0)\n");
    scanf("%d", &c);
    if (c == 1) {
        printf("No. of elements to remove: ");
        scanf("%d", &n);
        for (i = 0; i < n; i++)
            dequeue(&head2);
        display(head2);
    }
```

```
        }
        break;
        case 3: {
                printf("Name:RACHANA H D \tUSN:1BM22CS212");
                exit(0);
              }
      }
   }
   return 0;
}
```

Output:





Program 7: Write a program to Implement doubly link list with primitive operations

1. **Create a doubly linked list.**
2. **Insert a new node to the left of the node.**
3. **Delete the node based on a specific value**
4. **Display the contents of the list**

Solution:

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

struct node
{
   struct node *next, *prev;
   int data;
};
struct node *head;
void display()
{
   struct node *temp = head;
   if (temp == NULL)
   {
      printf("List is empty.\n");
      return;
   }
   while (temp != NULL)
   {
      printf("%d ", temp->data);
```

```
        temp = temp->next;
    }
    printf("\n");
}
void push()
{
    struct node *new_node = (struct node *)malloc(sizeof(struct node));
    int data;
    if(new_node==NULL)
        printf("Overflow\n");
    else
    {
        printf("Enter the data: ");
        scanf("%d", &data);
        new_node->data = data;
        if (head == NULL)
        {
            new_node->next = NULL;
            new_node->prev = NULL;
            head = new_node;
        }
        else
        {
            head->prev = new_node;
            new_node->next = head;
            new_node->prev = NULL;
            head = new_node;
```

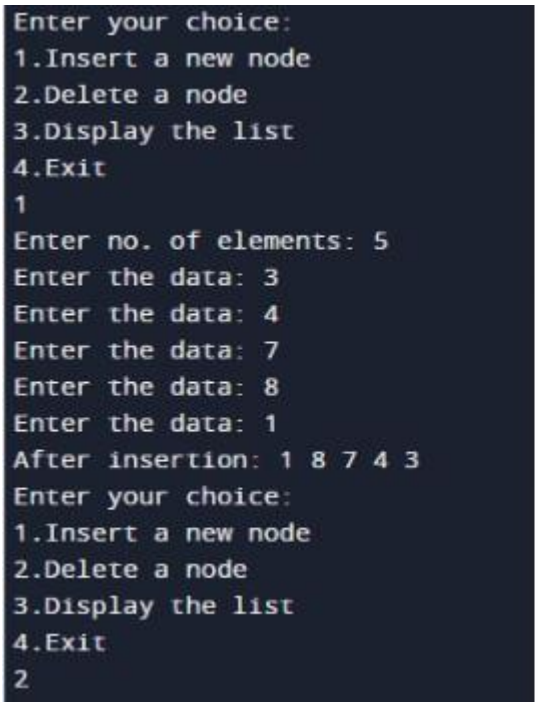
```
        }
    }
}

void delete_specified()
{
    int loc=1,val;
    printf("Enter the value to delete: ");
    scanf("%d", &val);
    struct node *temp = head;
    if (temp == NULL)
    {
        printf("List is empty. Nothing to delete.\n");
        return;
    }
    while(temp->data!=val)
    {
        loc++;
        temp=temp->next;
    }
    temp=head;
    if (loc == 1)
    {
        head = temp->next;
        if (head != NULL)
            head->prev = NULL;
        free(temp);
```

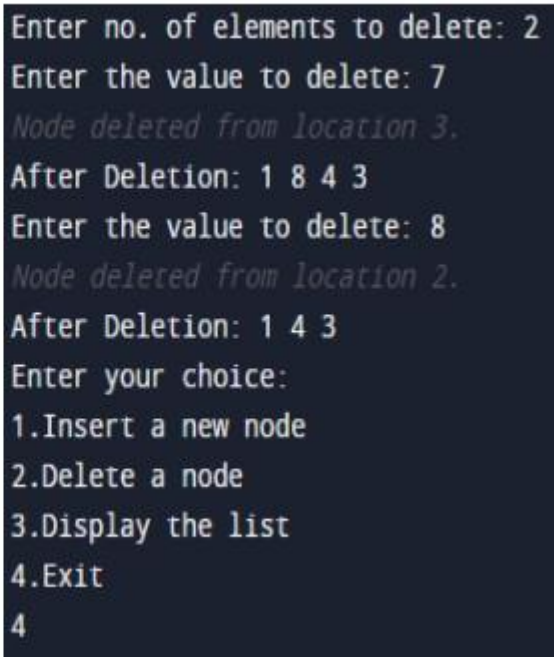
```
        printf("Node deleted from the beginning.\n");
        return;
    }
    for (int i = 1; i < loc; i++)
    {
        temp = temp->next;
        if (temp == NULL)
        {
            printf("Specified position does not exist.\n");
            return;
        }
    }
    if (temp->next == NULL)
    {
        temp->prev->next = NULL;
        free(temp);
        printf("Node deleted from the end.\n");
        return;
    }
    temp->prev->next = temp->next;
    temp->next->prev = temp->prev;
    free(temp);
    printf("Node deleted from location %d.\n", loc);
    printf("After Deletion: ");
    display();
}
int main()
```

```
{
   int ch;
   while (1)
   {
      printf("Enter your choice:\n1.Insert a new node\n2.Delete a node\n3.Display the
list\n4.Exit\n");
      scanf("%d", &ch);
      switch (ch)
      {
         case 1:{
                  printf("Enter no. of elements: ");
                  int n, i;
                  scanf("%d",&n);
                  for(i=0; i<n; i++)
                     push();
                  printf("After insertion: ");
                  display();
               }break;
         case 2:{
                  printf("Enter no. of elements to delete: ");
                  int n,i;
                  scanf("%d",&n);
                  for(i=0; i<n; i++)
                     delete_specified();
               }break;
         case 3:display(); break;
         case 4:{
```

```
                printf("Name: RACHANA H D\tUSN:1BM22CS212");
                exit(0);
            }
        }
    }
    return 0;
}
```

OUTPUT:



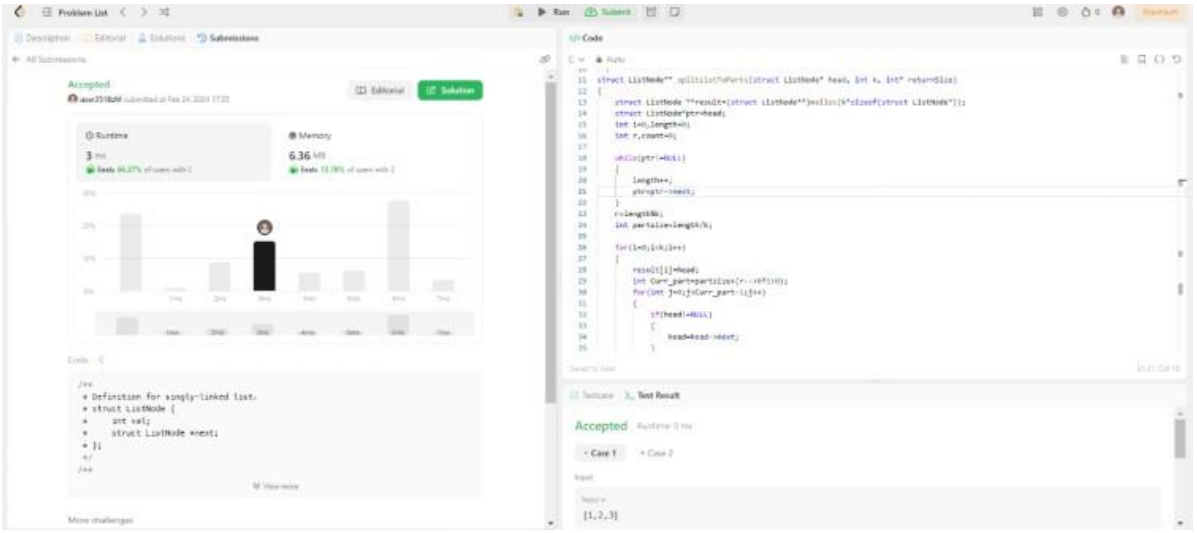


Program - Leetcode platform

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

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

```
int partsize=length/k;


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

Output:



Program 8: Write a program

1. To construct a binary Search tree.

2. To traverse the tree using all the methods i.e., in-order, preorder and post order To display the elements in the tree.

Solution:

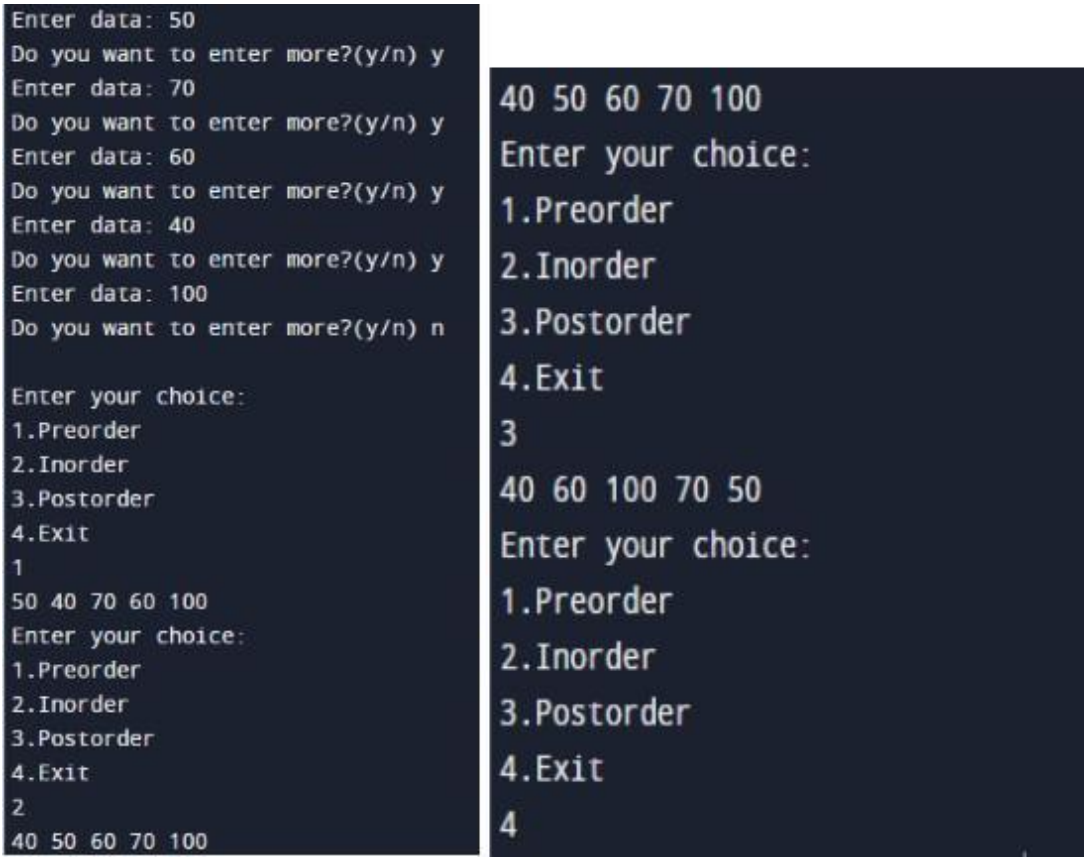
```
#include<stdio.h>
#include<stdlib.h>
typedef struct BST
{
   int data;
   struct BST *left;
   struct BST *right;
}node;
node *create()
{
   node  *t;
   printf("Enter data: ");
```

```
    t=(node*)malloc(sizeof(node));
    scanf("%d",&t->data);
    t->left=t->right=NULL;
    return t;
}
void insert(node *root,node*t)
{
    if(t->data<root->data)
    {
        if(root->left!=NULL)
            insert(root->left,t);
        else
            root->left=t;
    }
    if(t->data>root->data)
    {
        if(root->right!=NULL)
            insert(root->right,t);
        else
            root->right=t;
    }
}
void preorder(node *root)
{
    if(root!=NULL)
    {
        printf("%d ",root->data);
```

```
        preorder(root->left);
        preorder(root->right);
    }
}
void inorder(node *root)
{
    if(root!=NULL)
    {
        inorder(root->left);
        printf("%d ",root->data);
        inorder(root->right);
    }
}
void postorder(node *root)
{
    if(root!=NULL)
    {
        postorder(root->left);
        postorder(root->right);
        printf("%d ",root->data);
    }
}
int main()
{
    char ch;
    node *root=NULL,*t;
    do{
```

```
        t=create();
        if(root==NULL)
            root=t;
        else
            insert(root,t);
        printf("Do you want to enter more?(y/n) ");
        getchar();
        scanf("%c",&ch);
    }while(ch=='y'||ch=='Y');
    int c;
    while(1)
    {
        printf("\nEnter your choice:\n1.Preorder\n2.Inorder\n3.Postorder\n4.Exit\n");
        scanf("%d",&c);
        switch(c)
        {
            case 1:{preorder(root);}break;
            case 2:{inorder(root);}break;
            case 3:{postorder(root);}break;
            case 4:{
                    printf("Name: RACHANA H D\tUSN:1BM22CS212");
                       exit(0);
                   }
        }
    }
    return 0;
}
```

Output:

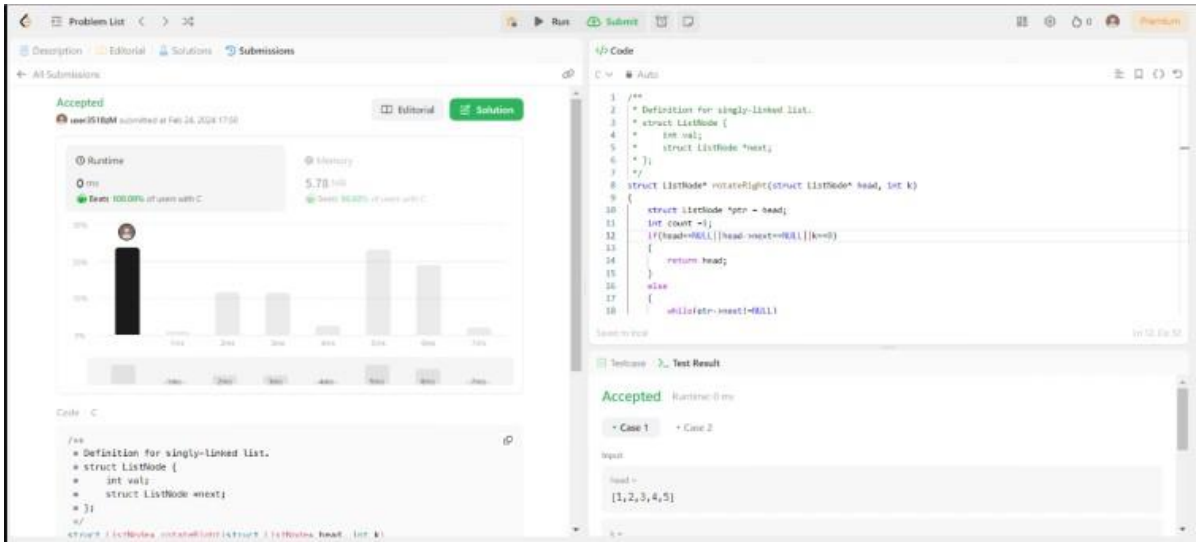

Enter data: 50
Do you want to enter more?(y/n) y
Enter data: 70
Do you want to enter more?(y/n) y
Enter data: 60
Do you want to enter more?(y/n) y
Enter data: 40
Do you want to enter more?(y/n) y
Enter data: 100
Do you want to enter more?(y/n) n
Enter your choice:
1.Preorder
2.Inorder
3.Postorder
4.Exit
1
50 40 70 60 100
Enter your choice:
1.Preorder
2.Inorder
3.Postorder
4.Exit
2
40 50 60 70 100
40 50 60 70 100
Enter your choice:
1.Preorder
2.Inorder
3.Postorder
4.Exit
3
40 60 100 70 50
Enter your choice:
1.Preorder
2.Inorder
3.Postorder
4.Exit
4

Program - Leetcode platform

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

```
        }
        ptr->next=head;
        for(int i=k%count;i<count;i++)
        {
            ptr=ptr->next;
        }
        head=ptr->next;
        ptr->next=NULL;
        return head;

    }
}
```

Output:



Program 9a: Write a program to traverse a graph using the BFS method.

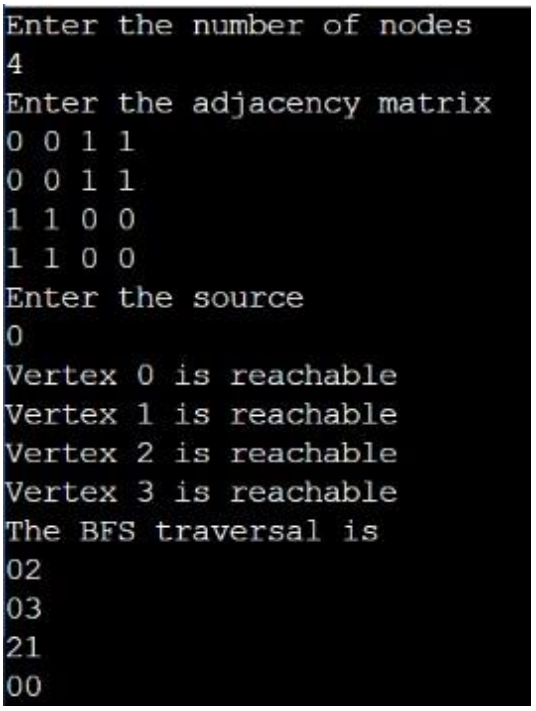
Solution:

```
#include<stdio.h>
#include<conio.h>
void bfs(int a[20][20], int n, int src, int t[20][2], int s[])
{
	int f,r,q[20],u,v,k=0,i;
	for(i=1;i<=n;i++)
		s[i]=0;
	f=r=k=0;
	q[r]=src;
	s[src]=1;
	while(f<=r)
	{
		u=q[f++];
		for(v=1;v<=n;v++)
		{
			if(a[u][v]==1 && s[v]==0)
			{
				s[v]=1;
				q[++r]=v;
				t[k][0]=u;
				t[k][1]=v;
				k++;
			}
		}
	}
```

```
}
void main()
{
	int n,a[20][20],src,t[20][2],flag,s[20],i,j;
	printf("Enter the number of nodes\n");
	scanf("%d", &n);
	printf("Enter the adjacency matrix\n");
	for(i=0;i<n;i++)
	{
		for(j=0;j<n;j++)
			scanf("%d", &a[i][j]);
	}
	printf("Enter the source\n");
	scanf("%d", &src);
	bfs(a,n,src,t,s);
	flag=0;
	for(i=0;i<n;i++)
	{
		if(s[i]==0)
		{
			printf("Vertex %d is not reachable\n", i);
			flag=1;
		}
		else
			printf("Vertex %d is reachable\n", i);
	}
	if(flag==1)
```

```
            printf("Some nodes are not visited\n");
        else
        {
            printf("The BFS traversal is\n");
            for(i=0;i<n;i++)
                printf("%d%d\n", t[i][0], t[i][1]);
        }
        getch();
}
```

Output:



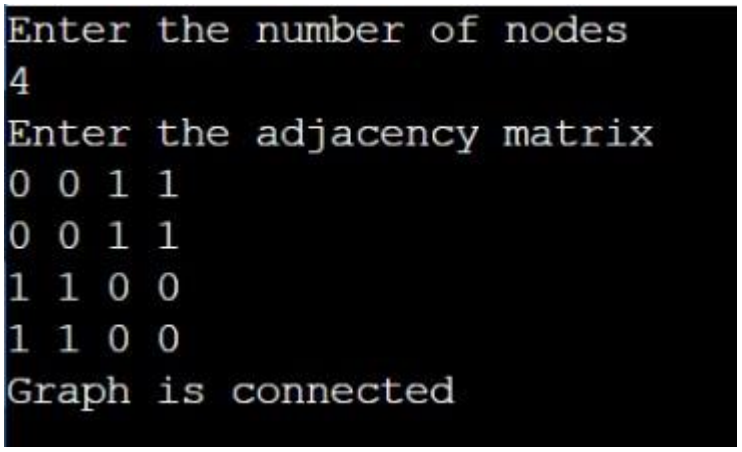
Program 9b: Write a program to check whether a graph is connected or not using the DFS method.

Solution:

```
#include<stdio.h>
#include<conio.h>
int a[1][10];
void dfs(int n, int cost[10][10], int u, int s[])
{
        int v;
        s[u]=1;
        for(v=0;v<n;v++)
        {
                if((cost[u][v]==1) && (s[v]==0))
                        dfs(n,cost,v,s);
        }
}
void main()
{
        int n,i,j,cost[10][10],s[10],con,flag;
        printf("Enter the number of nodes\n");
        scanf("%d", &n);
        printf("Enter the adjacency matrix\n");
        for(i=0;i<n;i++)
        {
                for(j=0;j<n;j++)
                        scanf("%d", &cost[i][j]);
```

```
        }
        con=0;
        for(j=0;j<n;j++)
        {
                for(i=0;i<n;i++)
                s[i]=0;
                dfs(n,cost,j,s);
                flag=0;
                for(i=0;i<n;i++)
                {
                        if(s[i]==0)
                                flag=1;
                }
                if(flag==0)
                        con=1;
        }
        if(con==1)
                printf("Graph is connected\n");
        else
                printf("Graph is not connected\n");
        getch();
}
```

Output:


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

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 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 -> Las 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 100 // Size of hash table

#define EMPTY -1 // Indicates empty cell in hash table


// Employee structure

struct Employee {

    int key; // Unique key

    // Add other employee data here

};


// Hash table structure

struct HashTable {

    struct Employee* table[TABLE_SIZE];

};


// Hash function using remainder method

int hash(int key, int m) {
```

```
    return key % m;
}

// Function to initialize hash table
void initHashTable(struct HashTable* ht) {
    for (int i = 0; i < TABLE_SIZE; i++) {
        ht->table[i] = NULL;
    }
}

// Function to insert employee record into hash table
void insert(struct HashTable* ht, struct Employee* emp) {
    int index = hash(emp->key, TABLE_SIZE);

    // Linear probing to resolve collisions
    while (ht->table[index] != NULL && ht->table[index]->key != EMPTY) {
        index = (index + 1) % TABLE_SIZE;
    }

    ht->table[index] = emp;
}

// Function to search for an employee record using key
struct Employee* search(struct HashTable* ht, int key) {
    int index = hash(key, TABLE_SIZE);
```

```
    while (ht->table[index] != NULL) {
        if (ht->table[index]->key == key) {
            return ht->table[index];
        }
        index = (index + 1) % TABLE_SIZE;
    }

    return NULL; // Employee not found
}

// Function to display hash table contents
void displayHashTable(struct HashTable* ht) {
    printf("Hash Table:\n");
    for (int i = 0; i < TABLE_SIZE; i++) {
        if (ht->table[i] != NULL && ht->table[i]->key != EMPTY) {
            printf("Index %d: Key %d\n", i, ht->table[i]->key);
        } else {
            printf("Index %d: Empty\n", i);
        }
    }
}

int main() {
    struct HashTable ht;
```

```
    initHashTable(&ht);

    // Example employee records
    struct Employee emp1 = {1234}; // Key: 1234
    struct Employee emp2 = {5678}; // Key: 5678

    // Insert employee records into hash table
    insert(&ht, &emp1);
    insert(&ht, &emp2);

    // Display hash table
    displayHashTable(&ht);

    // Search for an employee
    int keyToSearch = 1234;
    struct Employee* foundEmp = search(&ht, keyToSearch);
    if (foundEmp != NULL) {
        printf("\nEmployee found with key %d\n", foundEmp->key);
    } else {
        printf("\nEmployee with key %d not found\n", keyToSearch);
    }

    return 0;
}
```


Index 74: Empty
Index 75: Empty
Index 76: Empty
Index 77: Empty
Index 78: Key 5678
Index 79: Empty
Index 80: Empty
Index 81: Empty
Index 82: Empty
Index 83: Empty
Index 84: Empty
Index 85: Empty
Index 86: Empty
Index 87: Empty
Index 88: Empty
Index 89: Empty
Index 90: Empty
Index 91: Empty
Index 92: Empty
Index 93: Empty
Index 94: Empty
Index 95: Empty
Index 96: Empty
Index 97: Empty
Index 98: Empty
Index 99: Empty
Employee found with key 1234
Process returned 0 (0x0) execution time : 13.766 s

Output:

HACKER RANK:

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

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

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

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

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

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

    TreeNode* nodes[indexes_rows + 1];
    nodes[1] = root;

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

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

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

```
            nodes[indexes[i][1]] = curr->right;
        }
    }

    return root;
}

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

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

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

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

    return result;
}
```