VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Data Structures using C
Submitted by

Sakshi Shetty(1BM22CS234)

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 **SAKSHI SHETTY(1BM22CS234),** 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.

Radhika A D   Dr. Jyothi S Nayak
Assistant Professor   Professor and Head
Department of CSE   Department of CSE
BMSCE, Bengaluru   BMSCE,Bengaluru

Index Sheet

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	31-38
7	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 7b) Program - Leetcode platform	39-43
8	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. 8b) Program - Leetcode platform	44-48
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.	49-52

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.

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

CODE:

```c
#include <stdio.h>
#define n 5

void push();
void pop();
void display();

int top=-1;
int stack[n];

void main()
{
    int ch;
    printf("Santosh  B\n");
    printf("1BM22CS243\n\n");
    while(1)
    {
        printf("Stack menu:\n");
        printf("\n1.Push \n2.Pop \n3.Display \n4.exit\n\n");
        printf("Select your choice:");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:push();
                break;
            case 2:pop();
                break;
            case 3:display();
                break;
            case 4:exit(0);
```

```c
        default:printf("Invalid choice");
      }
    }
}
void push()
{
    int val;
    if(top==n-1)
    {
        printf("Stack is full");
    }
    else
    {
        printf("Enter the element:");
        scanf("%d",&val);
        top++;
        stack[top]=val;
    }
}
void pop()
{
    if(top==-1)
    {
        printf("Stack is empty");
    }
    else
    {
        printf("Deleted element is %d",stack[top]);
        top--;
    }
}
void display()
{
    if(top==-1)
        printf("Stack is empty");
    else
    {
        int i;
```

```
    for(i=top;i>=0;i--)
        printf("%d\n",stack[i]);
    }
}
```

OUTPUT:

```
Stack menu:

1.Push
2.Pop
3.Display
4.exit

Select your choice:1
Enter the element:2
Stack menu:

1.Push
2.Pop
3.Display
4.exit

Select your choice:1
Enter the element:2
Stack menu:

1.Push
2.Pop
3.Display
4.exit

Select your choice:3
2
2
Stack menu:

1.Push
2.Pop
3.Display
4.exit

Select your choice:4

Process returned 0 (0x0)    execution time : 13.218 s
Press any key to continue.
```

LAB PROGRAM 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)

CODE:
```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <stdlib.h>
#define MAX 100
char st[MAX];
int top = -1;
void push(char st[], char);
char pop(char st[]);
void InfixtoPostfix(char source[], char target[]);
int getpri(char);

void main()
{
    char infix[100], postfix[100];
    printf("\n Enter any infix expression : ");
    gets(infix);
    strcpy(postfix,"");
    InfixtoPostfix(infix, postfix);
    printf("\n The corresponding postfix expression is : ");
    puts(postfix);
}

void InfixtoPostfix(char source[], char target[])
```

```c
{
    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] == '%' || source[i] == '^')
        {
            while ((top != -1) && (st[top] != '(') && (getpri(st[top]) >
getpri(source[i])))
            {
```

```c
            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';
}
int getpri(char op)
{
  if (op == '^')
    return 2;
  else if (op == '/' || op == '*' || op == '%')
    return 1;
  else if (op == '+' || op == '-')
    return 0;
}
void push(char st[], char val)
{
  if (top == MAX - 1)
    printf("\n STACK OVERFLOW");
  else
  {
    top++;
    st[top] = val;
  }
}
```

```
char pop(char st[])
{
    char val = ' ';
    if (top == -1)
        printf("\n STACK UNDERFLOW");
    else
    {
        val = st[top];
        top--;
    }
    return val;
}
```

OUTPUT:


```
 Enter any infix expression : (A-(B/C+(D%E*F)/G)*H)

 The corresponding postfix expression is : ABC/DEF*%G/+H*-

Process returned 0 (0x0)   execution time : 95.709 s
Press any key to continue.
```

LAB PROGRAM 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:

```c
#include <stdio.h>
#define N 5

int q[N];
int front = -1, rear = -1;
void insert(int);
int delete();
void display();
void main()
{
    int n, choice;
    do
    {
        printf("\n1.Insert\n2.Delete\n3.Display\n4.Exit\n");
        printf("Enter your option : \n");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            printf("Enter the number to be inserted in the queue : \n");
            scanf("%d", &n);
            insert(n);
```

```c
            break;
        case 2:
            n = delete ();
            if (n != -1)
                printf("\n The number deleted is : %d\n", n);
            break;
        case 3:
            display();
            break;
        case 4:
            exit(0);
            break;
        default:
            printf("Invalid option\n");
            exit(0);
            break;
        }
    } while (choice != 4);
}
void insert(int num)
{

    if (rear == N - 1)
        printf("\n OVERFLOW");
    else if (front == -1 && rear == -1)
        front = rear = 0;
    else
        rear++;
    q[rear] = num;
}
int delete()
{
    int val;
    if (front == -1 || front > rear)
    {
        printf("\n UNDERFLOW");
        return -1;
    }
```

```c
    else
    {
        val = q[front];
        front++;
        if (front > rear)
            front = rear = -1;
        return val;
    }
}
void display()
{
    int i;
    printf("\n");
    if (front == -1 || front > rear)
        printf("\n QUEUE IS EMPTY");
    else
    {
        for (i = front; i <= rear; i++)
            printf("\t %d", q[i]);
    }
}
```

OUTPUT:

```
1.Insert
2.Delete
3.Display
4.Exit
Enter your option :1
Enter the number to be inserted in the queue :1

1.Insert
2.Delete
3.Display
4.Exit
Enter your option :1
Enter the number to be inserted in the queue :2

1.Insert
2.Delete
3.Display
4.Exit
Enter your option :1
Enter the number to be inserted in the queue :3

1.Insert
2.Delete
3.Display
4.Exit
Enter your option :2

  The number deleted is : 1

1.Insert
2.Delete
3.Display
4.Exit
Enter your option :3

         2         3
1.Insert
2.Delete
3.Display
4.Exit
Enter your option :4

Process returned 0 (0x0)    execution time : 18.579 s
Press any key to continue.
```

b)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:

```c
#include <stdio.h>
#define N 5
int q[N];
int front = -1, rear = -1;
void insert(int);
int delete();
void display();
void main()
{
    int n, choice;
    do
    {
        printf("\n1.Insert\n2.Delete\n3.Display\n4.Exit\n");
        printf("Enter your option : \n");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            printf("Enter the number to be inserted in the queue : \n");
            scanf("%d", &n);
            insert(n);
            break;
        case 2:
            n = delete ();
            if (n != -1)
                printf("\n The number deleted is : %d\n", n);
            break;
        case 3:
            display();
            break;
        case 4:
            exit(0);
```

```c
            break;
        default:
            printf("Invalid option\n");
            exit(0);
            break;
        }
    } while (choice != 4);
}
void insert(int num)
{
    if ((front == 0 && rear == N - 1) || (rear == (front - 1)))
        printf("\n OVERFLOW");
    else if (front == -1 && rear == -1)
    {
        front = rear = 0;
        q[rear] = num;
    }
    else if (rear == N - 1 && front != 0)
    {
        rear = 0;
        q[rear] = num;
    }
    else
    {
        rear++;
        q[rear] = num;
    }
}
int delete()
{
    int val;
    if (front == -1 && rear == -1)
    {
        printf("\n UNDERFLOW");
        return -1;
    }
    val = q[front];
    if (front == rear)
```

```c
        front = rear = -1;
    else
    {
        if (front == N - 1)
            front = 0;
        else
            front++;
    }
    return val;
}
void display()
{
    int i;
    printf("\n");
    if (front == -1 && rear == -1)
        printf("\n QUEUE IS EMPTY");
    else
    {
        if (front < rear)
        {
            for (i = front; i <= rear; i++)
                printf("\t %d", q[i]);
        }
        else
        {
            for (i = front; i < N; i++)
                printf("\t %d", q[i]);
            for (i = 0; i <= rear; i++)
                printf("\t %d", q[i]);
        }

    }
}
```

OUTPUT:

```
1.Insert
2.Delete
3.Display
4.Exit
Enter your option :1
Enter the number to be inserted in the queue : 1

1.Insert
2.Delete
3.Display
4.Exit
Enter your option :1
Enter the number to be inserted in the queue : 2

1.Insert
2.Delete
3.Display
4.Exit
Enter your option :1
Enter the number to be inserted in the queue : 3

1.Insert
2.Delete
3.Display
4.Exit
Enter your option :1
Enter the number to be inserted in the queue : 3

1.Insert
2.Delete
3.Display
4.Exit
Enter your option :3

        1       2       3       3
1.Insert
2.Delete
3.Display
4.Exit
Enter your option :
4

Process returned 0 (0x0)   execution time : 20.489 s
Press any key to continue.
```

LAB PROGRAM 4:

a)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.

CODE:

```c
#include <stdio.h>
#include<stdlib.h>
typedef struct Node {
int data;
struct Node *next;
}Node;
void InsertAtBeginning( Node **head_ref,int new_data);
void InsertAtEnd( Node **head_ref,int new_data);
void Insert( Node **prev_node,int new_data,int pos);
void PrintList(Node * next);
void InsertAtBeginning( Node **head_ref,int new_data)
{
        struct Node* new_node=(struct Node*)malloc(sizeof( Node));
        new_node->data=new_data;
        new_node->next=*head_ref;
        *head_ref=new_node;
}
void InsertAtEnd(Node **head_ref,int new_data)
{
        struct 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;
```

```c
        return ;
        }
        while (last->next!=NULL)
        last=last->next;
        last->next=new_node;
}
void Insert(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 PrintList(Node *node)
        {
        while (node!=NULL)
        {
        printf("%d\n",node->data);
        node=node->next;
        }
}
int main()
{
        int ch,new,pos;
        Node* head=NULL;
        printf("Santosh B\n");
        printf("1BM22CS243\n\n\n");
        while(ch!=5)
```

```c
{
        printf("Menu\n");
        printf("1.Insert at beginning\n");
        printf("2.Insert at a specific position\n");
        printf("3.Insert at 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);
        InsertAtBeginning(&head,new);
        break;
}
case 2:
{
        printf("Enter the data and position at which you want to
insert \n");
        scanf("%d%d",&new,&pos);
        Insert(&head,new,pos);
        break;
}
case 3:
{
        printf("Enter the data you want to insert at end\n");
        scanf("%d",&new);
        InsertAtEnd(&head,new);
        break;
}
case 4:
{printf("Created linked list is:\n");
PrintList(head);
break;
}
```

```c
case 5:
{
return 0;
break;
}
case 6:
{
printf("Invalid data!");
break;
}
}
    }
    return 0;
    }
```

OUTPUT:

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

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

b)LEETCODE:WAP for Valid Parenthesis

CODE:

```c
#define MAX 10
char stack[MAX];
int top = -1;

void push(char );
char pop(void );
char peek(void );
bool isValid(char* s) {
    int i = 0;
    while(s[i] != '\0') {
        if(s[i] == '(' || s[i] == '[' || s[i] == '{') {
            push(s[i]);
        }
        else if(s[i] == ')'){
            char ch = pop();
            if(ch != '(') return false;
        }
        else if(s[i] == ']'){
            char ch = pop();
            if(ch != '[') return false;
        }
        else if(s[i] == '}'){
            char ch = pop();
            if(ch != '{') return false;
        }
        i++;
    }
    if(top != -1) return false;
    return true;
}

void push(char ch) {
    if(top == MAX - 1) return;
    stack[++top] = ch;
```

```
}

char pop() {
    if(top == -1) return 'n';
    return stack[top--];
}
```

OUTPUT :

LAB PROGRAM 5:

 a) 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.

CODE:

```
#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)
{
        struct 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;
```

```c
        *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;
        }
```

```c
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;
    printf("Santosh B\n");
    printf("1BM22CS243\n\n\n");
    while(ch!=6)
    {
    printf("Menu\n");
    printf("1.Create a linked list\n");
```

```c
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;
}
```

```c
case 6:
{
return 0;
break;
}
default:
{
printf("Invalid data!");
break;
}
}
}
return 0;
}
```

OUTPUT:

```
 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
5
Created linked list is:
4
3
2
1
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

 Deleted Node from the last ...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:
4
3
2
```

```
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
3
Enter the position at which you want to delete
3

Deleted node with position 3Menu
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:
4
3
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
```

b)LEETCODE: Reverse the LinkedList

CODE:

```
struct ListNode* reverse(struct ListNode* node);
struct ListNode* reverseList(struct ListNode* head) {
   if(head == NULL) return NULL;
   return reverse(head);
}

struct ListNode* reverse(struct ListNode *node) {
   if(node->next == NULL) {
      return node;
   }
   struct ListNode* newHead = reverse(node->next);
   node->next->next = node;
   node->next = NULL;
   return newHead;
}
```

OUTPUT:

PROGRAM 6:

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

CODE:
```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node
{
int data;
struct Node *next;
} Node;
struct Node *createNode(int value)
{
        struct Node *newNode = (struct Node *)malloc(sizeof(struct
Node));
        if (newNode == NULL)
        {
        printf("Memory allocation failed.\n");
        exit(1);
        }
        newNode->data = value;
        newNode->next = NULL;
        return newNode;
}
struct Node *insertAtBeginning(struct Node *head, int value)
{
        struct Node *newNode = createNode(value);
        newNode->next = head;
        return newNode;
}
struct Node *concat(Node *head1, Node *head2)
{
        Node *temp = head1;
        while (temp->next != NULL)
        temp = temp->next;
        temp->next = head2;
        return head1;
```

```
}
struct Node *sort(Node *head)
{
        Node *temp, *current;
        int t;
        current = head;
        while (current != NULL)
        {
        temp = head;
        while (temp->next != NULL)
        {
        if (temp->data > temp->next->data)
        {
        t = temp->data;temp->data = temp->next->data;
        temp->next->data = t;
        }
        temp = temp->next;
        }
        current = current->next;
        }
        return head;
}
struct Node *reverse(Node *head)
{
        Node *prev, *temp, *next;
        temp = head;
        prev = NULL;
        while (temp != NULL)
        {
        next = temp->next;
        temp->next = prev;
        prev = temp;
        temp = next;
        }
        head = prev;
        return head;
        }
        void displayLinkedLists(struct Node *head1, struct Node *head2)
```

```c
{
printf("Linked List 1: ");
while (head1 != NULL)
{
printf("%d -> ", head1->data);
head1 = head1->next;
}
printf("NULL\n");
printf("Linked List 2: ");
while (head2 != NULL)
{
printf("%d -> ", head2->data);
head2 = head2->next;
}
printf("NULL\n");
}
int main()
{
        printf("Santosh B");
        printf("1BM22CS243");
        struct Node *list1 = NULL;
        struct Node *list2 = NULL;
        int choice, data;
        list1 = insertAtBeginning(list1, 1);
        list1 = insertAtBeginning(list1, 2);
        list1 = insertAtBeginning(list1, 3);
        list2 = insertAtBeginning(list2, 4);list2 = insertAtBeginning(list2, 5);
        list2 = insertAtBeginning(list2, 6);
        displayLinkedLists(list1, list2);
        printf("After Sorting:\n");
        list1 = sort(list1);
        list2 = sort(list2);
        displayLinkedLists(list1,list2);
        printf("After concatenation:\n");
        list1 = concat(list1, list2);
        displayLinkedLists(list1, list2);
        printf("After reversing:\n");
        list1 = reverse(list1);
```

```
        displayLinkedLists(list1, list2);
        return 0;
}
```

OUTPUT:

```
Linked List 1: 3 -> 2 -> 1 -> NULL
Linked List 2: 6 -> 5 -> 4 -> NULL
After Sorting:
After concatenation:
Linked List 1: 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> NULL
Linked List 2: 4 -> 5 -> 6 -> NULL
After reversing:
Linked List 1: 6 -> 5 -> 4 -> 3 -> 2 -> 1 -> NULL
Linked List 2: 4 -> 3 -> 2 -> 1 -> NULL

Process returned 0 (0x0)   execution time : 0.463 s
Press any key to continue.
```

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

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

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

void push(node **head, int new_data)
{
   node *new_node = (node *)malloc(sizeof(node));
   new_node->data = new_data;
   new_node->next = NULL;

   if (*head == NULL)
   {
      *head = new_node;
   }
   else
   {
      node *temp = *head;
      while (temp->next != NULL)
      {
         temp = temp->next;
      }
      temp->next = new_node;
   }
}

void pop(node **head)
{
   if (*head == NULL)
   {
```

```c
        printf("Stack is empty\n");
    }
    else
    {
        node *temp = *head;
        node *prev = NULL;

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

        if (prev == NULL)
        {
            // Only one element in the list
            *head = NULL;
        }
        else
        {
            prev->next = NULL;
        }

        printf("Popped element: %d\n", temp->data);
        free(temp);
    }
}

void enqueue(node **front, int new_data)
{
    node *new_node = (node *)malloc(sizeof(node));
    new_node->data = new_data;
    new_node->next = NULL;

    if (*front == NULL)
    {
        *front = new_node;
    }
```

```c
        else
        {
            node *temp = *front;
            while (temp->next != NULL)
            {
                temp = temp->next;
            }
            temp->next = new_node;
        }
    }

    void dequeue(node **front)
    {
        if (*front == NULL)
        {
            printf("Queue is empty\n");
        }
        else
        {
            node *temp = *front;
            *front = temp->next;

            printf("Dequeued element: %d\n", temp->data);
            free(temp);
        }
    }

    void display(node *list)
    {
        node *current = list;
        while (current != NULL)
        {
            printf("%d ", current->data);
            current = current->next;
        }
        printf("\n");
    }
```

```c
int main()
{
    node *stack = NULL;
    node *queue = NULL;

    // Stack operations
    push(&stack, 1);
    push(&stack, 2);
    push(&stack, 3);

    // Display the stack
    printf("Stack: ");
    display(stack);

    // Pop elements from the stack
    pop(&stack);
    pop(&stack);
    pop(&stack);

    // Queue operations
    enqueue(&queue, 4);
    enqueue(&queue, 5);
    enqueue(&queue, 6);

    // Display the queue
    printf("Queue: ");
    display(queue);

    // Dequeue elements from the queue
    dequeue(&queue);
    dequeue(&queue);
    dequeue(&queue);

    return 0;
}
```

OUTPUT:


```
Stack: 1 2 3
Popped element: 3
Popped element: 2
Popped element: 1
Queue: 4 5 6
Dequeued element: 4
Dequeued element: 5
Dequeued element: 6

Process returned 0 (0x0)    execution time : 0.876 s
Press any key to continue.
```

LAB PROGRAM 7:

a)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

CODE:
```
#include <stdio.h>
#include <stdlib.h>
struct node{
int data;
struct node* prev;
struct node* next;
};
void insertatbeg(struct node**head_ref,int new_data)
{
        struct node* new_node=(struct node*)malloc(sizeof(struct node));
        new_node->data=new_data;
        new_node->prev=NULL;
        struct node *temp;
        if((*head_ref)==NULL)
        {
        new_node->next=NULL;
        (*head_ref)=new_node;
        }
        else
```

```c
        {
        new_node->next=*head_ref;
        (*head_ref)->prev=new_node;
        (*head_ref)=new_node;
        }
}
void deleteatspec(struct node**head_ref,int value)
{
        struct node* temp;
        temp=*head_ref;
        if(temp==NULL)
        {
        printf("Its empty");
        }
        else
        {
        while(temp->data!=value)
        {
        temp=temp->next;
        }
        temp->next->prev=temp->prev;temp->prev->next=temp->next;
        free(temp);
        }
}
void display(struct node**head_ref)
{
        if(*head_ref==NULL)
        {
        printf("The list is empty");
        }
        else
        {
        struct node*ptr;
        ptr=*head_ref;
        while(ptr!=NULL)
        {
        printf("%d\n",ptr->data);
        ptr=ptr->next;
```

```c
            }
        }
}
void main()
{
        struct node *head=NULL;
        int ch;
        int n;
        printf("1.insert\n2.delete\n3.display\nEnter your choice\n");
        while(1){
        scanf("%d",&ch);
        switch(ch)
        {
        case 1:printf("enter the data to insert");
        scanf("%d",&n);
        insertatbeg(&head,n);
        break;
        case 2:printf("enter the element to delete:");
        scanf("%d",&n);
        deleteatspec(&head,n);
        break;
        case 3:printf("the list is:");
        display(&head);
        break;
        case 4:exit(0);
        }
    }
}
```

OUTPUT:

```
1.insert
2.delete
3.display
Enter your choice1
enter the data to insert1
Enter your choice1
enter the data to insert2
Enter your choice1
enter the data to insert3
Enter your choice3
the list is:3
2
1
Enter your choice2
enter the element to delete:2
Enter your choice3
the list is:3
1
Enter your choice
```

c) LEETCODE:

CODE: Leaf similar trees

```c
int sum1 = 0;
int sum2 = 0;

void preOrder1(struct TreeNode* node);
void preOrder2(struct TreeNode* node);
bool leafSimilar(struct TreeNode* root1, struct TreeNode* root2) {
    preOrder1(root1);
    preOrder2(root2);

    return sum1 == sum2;
}
void preOrder1(struct TreeNode* node) {
    if (node == NULL) return;

    if (node->left == NULL && node->right == NULL) {
        sum1 = (sum1*10) + node->val;
        return;
    }

    preOrder1(node->left);
    preOrder1(node->right);
}

void preOrder2(struct TreeNode* node) {
    if (node == NULL) return;

    if (node->left == NULL && node->right == NULL) {
        sum2 = (sum2*10) + node->val;
        return;
    }

    preOrder2(node->left);
    preOrder2(node->right);
}
```
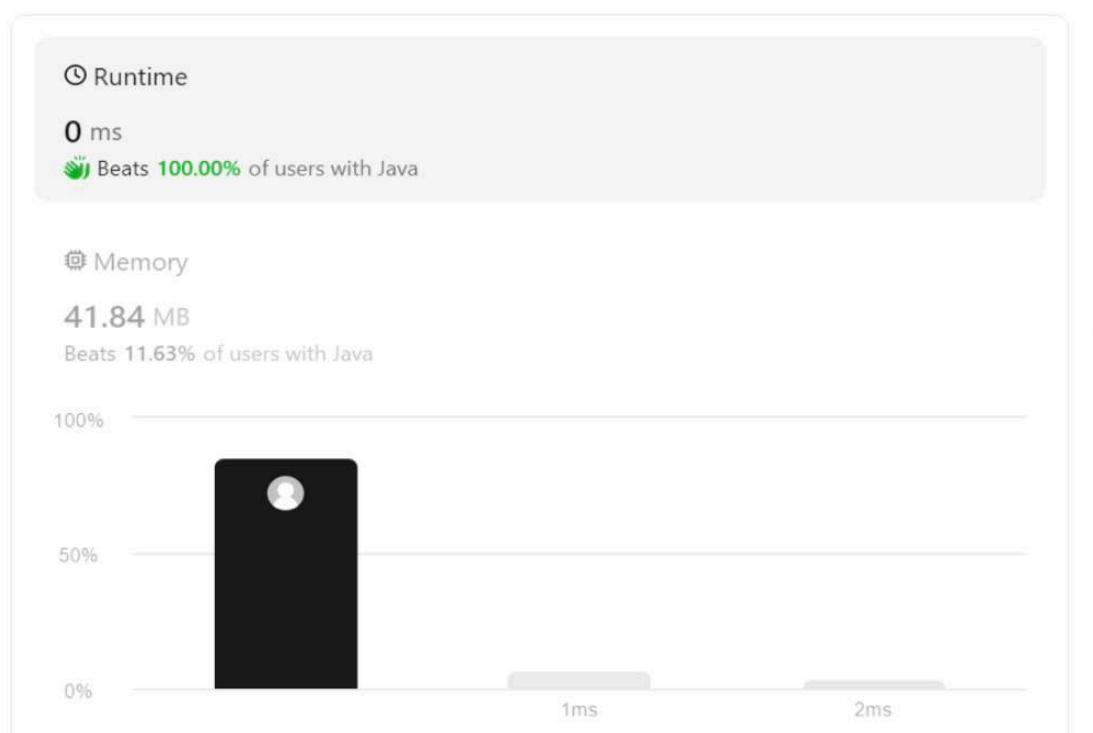
PROGRAM 8:

a)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>
typedef struct NODE
{
int info;
struct NODE *lchild;
struct NODE *rchild;
}NODE;
NODE *root=NULL;
void create();
void insert(int);
void inorder(NODE *);
void preorder(NODE *);
void postorder(NODE *);
int main()
{
int ch,key;
do
{
        printf("1.create\t2.inorder\t3.preorder\t4.postorder\t5.exit\n");
        printf("Enter your choice\n");
        scanf("%d",&ch);
        switch(ch)
        {
        case 1 : create();
        break;
        case 2 : inorder(root);
        break;
        case 3 : preorder(root);
        break;
```

```c
case 4 : postorder(root);
break;
case 5 : exit(0);
default : printf("Invalid choice");
}
}while(ch!=6);
return 0;
}
void create()
{
int n,i,e;
printf("enter the number of elements\n");
scanf("%d",&n);
printf("enter the elements one by one\n");
for(i=1;i<=n;i++)
{
scanf("%d",&e);
insert(e);
}
printf("tree constructed\n");
}
void insert(int e)
{
        NODE *nn,*temp,*prev;
        nn=(NODE *)malloc(sizeof(NODE));
        nn->info=e;
        nn->lchild=NULL;
        nn->rchild=NULL;
        if(root==NULL)
        {
        root=nn;
        return;
        }
        temp=root;
        while(temp!=NULL)
        {
        prev=temp;
        if(e<temp->info)
```

```c
        temp=temp->lchild;
        else if(e>temp->info)
        temp=temp->rchild;
        else
        {
        printf("its a duplicate node");
        return;
        }
        }
        if(e<prev->info)
        prev->lchild=nn;
        else
        prev->rchild=nn;
}
void inorder(NODE *tree)
{
        if(tree!=NULL)
        {
        inorder(tree->lchild);
        printf("%d\n",tree->info);
        inorder(tree->rchild);
        }
}
void preorder(NODE *tree)
{
        if(tree!=NULL){
        printf("%d\n",tree->info);
        preorder(tree->lchild);
        preorder(tree->rchild);
        }
}
void postorder(NODE *tree)
{
        if(tree!=NULL)
        {
        postorder(tree->lchild);
        postorder(tree->rchild);
        printf("%d\n",tree->info);
```

```
            }
}
 OUTPUT:
```

```
1.create          2.inorder       3.preorder      4.postorder     5.exit
Enter your choice
1
enter the number of elements
5
enter the elemeents one by one
10
50
30
90
100
tree constructed
1.create          2.inorder       3.preorder      4.postorder     5.exit
Enter your choice
2
10
30
50
90
100
1.create          2.inorder       3.preorder      4.postorder     5.exit
Enter your choice
3
10
50
30
90
100
1.create          2.inorder       3.preorder      4.postorder     5.exit
Enter your choice
4
30
100
90
50
10
1.create          2.inorder       3.preorder      4.postorder     5.exit
Enter your choice
5

Process returned 0 (0x0)   execution time : 20.002 s
Press any key to continue.
```

b)HACKER RANK: Reverse a double linked list

CODE:
```
DoublyLinkedListNode* reverse(DoublyLinkedListNode* llist) {
    DoublyLinkedListNode* current = llist;
    DoublyLinkedListNode* temp = NULL;

    // Traverse the list and swap prev and next pointers for each node
    while (current != NULL) {
        temp = current->prev;
        current->prev = current->next;
        current->next = temp;

        // Move to the next node
        current = current->prev;
    }

    // Update the head pointer to the last node (previous head becomes
the new tail)
    if (temp != NULL) {
        llist = temp->prev;
    }

    return llist;
}
```

LAB PROGRAM 9:

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

CODE:

```c
#include <stdio.h>
#include <stdlib.h>
struct node{
    int data;
    struct node *next;
}*front=NULL,*rear=NULL;
void enquque(int x){
    struct node t=(struct node) malloc(sizeof(struct node));
    if(t==NULL){
        printf("queue is overflow");
    }
    else{
        t->data=x;
        t->next=NULL;
    if(front==NULL){
        front=rear=t;


    }
    else{
        rear->next=t;
        rear=t;
    }
    }
}
int dequque(){
    struct node *t;
    int x=-1;
    if(front==NULL){
        printf("queque is empty");
        return x;
```

```c
        }
        else{
            t=front;
          x=t->data;
          front=front->next;
        free(t);
        return x;

        }
}
int isempt(){
    if(front==NULL){
        return 1;
    }
    return 0;
}
//traverse a graph using BFS
void bfs(int i,int visited[],int a[][20],int n){
    int u;
    printf("bfs traversal:");
    printf("%d ",i);
    visited[i-1]=1;
    enquque(i-1);
    while(!isempt()){
        u=dequque();
        for(int v=0;v<n;v++){
            if(a[u][v]==1 && visited[v]==0){
                printf("%d ",v+1);
                visited[v]=1;
                enquque(v);
            }
        }

    }
}
//connected or not using DFS
void dfs(int i,int visited[],int a[][20],int n){
    if(visited[i-1]==0){
```

```c
        printf("%d ",i);
        visited[i-1]=1;
        for(int j=0;j<n;j++){
            if(a[i-1][j]==1 && visited[j]==0){
                dfs(j+1,visited,a,n);
            }
        }
    }
}
void main(){
    int visited[20]={0};
    int a[20][20];
    int n,first;
    int count=0;
    printf("enter the number of vertices:");
    scanf("%d",&n);
    printf("enter the adjacency matrix:");
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            scanf("%d",&a[i][j]);
        }
    }
    printf("the adjacency matrix:\n");
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            printf("%d\t",a[i][j]);
        }
        printf("\n");
    }
    printf("enter the starting vertex: ");
    scanf("%d",&first);
    bfs(first,visited,a,n);
    for(int i=0;i<20;i++){
        visited[i]=0;
    }
    printf("\ndfs traversal:");
    dfs(first,visited,a,n);
    for(int i=0;i<n;i++){
```

```
        if(visited[i]==1){
            count++;
        }
    }
    if(count==n){
        printf("\ngraph is connected");
    }
    else{
        printf("\ngraph is not connected");
    }
}
```

OUTPUT:

```
enter the number of vertices:7
enter the adjacency matrix:
0 1 0 1 0 0 0
1 0 1 1 0 1 1
0 1 0 1 1 1 0
1 1 1 0 1 0 0
0 0 1 1 0 0 1
0 1 1 0 0 0 0
0 1 0 0 1 0 0
the adjacency matrix:
0       1       0       1       0       0       0
1       0       1       1       0       1       1
0       1       0       1       1       1       0
1       1       1       0       1       0       0
0       0       1       1       0       0       1
0       1       1       0       0       0       0
0       1       0       0       1       0       0
enter the starting vertex: 4
bfs traversal:4 1 2 3 5 6 7
dfs traversal:4 1 2 3 5 7 6
graph is connected
```