

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT On

DATA STRUCTURES (23CS3PCDST)

Submitted by

Alok (1BM23CS024)

**in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING**



**B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
September 2024-January 2025**

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



This is to certify that the Lab work entitled “**DATA STRUCTURES**” carried out by **Alok (1BM23CS024)**, who is bona-fide student of **B. M. S. College of Engineering**. It is in partial fulfilment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2024-25. The Lab report has been approved as it satisfies the academic requirements in respect of Data structures Lab - (**23CS3PCDST**)work prescribed for the said degree.

Prof. Basavaraj Jakkali
Associate Professor
Department of CSE
BMSCE, Bengaluru

Dr. Kavitha Sooda
Professor and Head
Department of CSE
BMSCE, Bengaluru

Index

Sl. No.	Experiment Title	Page No.
1	Stack Program	1
2	Infix to Postfix Expression	5
3	Queue Program & LeetCode – Valid Parentheses	8
4	Circular Queue Program & LeetCode – First non-repeating character	14
5	Creation, Insertion and Display of Singly Linked List & Creation, Deletion and Display of Singly Linked List	20
6	Sort, Reverse and Concatenate Singly Linked Lists & Stimulate Stack and Queue using Singly Linked List	32
7	Doubly Linked List Program	51
8	Binary Search Tree Program	56
9	BFS Traversal of Graph & Check if Graph is connected using DFS Traversal	62
10	Hashing Program	67

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.

Q) 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 and stack underflow.

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

int stk[3],top=-1,ch;

char chk(){
    if (top==-1)
        return ('a');
    if (top>(max-1))
        return('b');
}

void push(){
    char ch=chk();
    if (ch=='b'){
        printf("STACK OVERFLOW\n");
        return;
    }
    int element;
    top++;
    printf("Enter element: ");
    scanf("%d",&element);
    stk[top]=element;
    printf("Element pushed into stack \n");
}

void pop(){
    char ch=chk();
    if (ch=='a'){
        printf("STACK UNDERFLOW\n");
        return;
    }
}
```

```

    }
    printf("%d\n",stk[top]);
    top--;
    printf("Element popped from stack \n");
}

void display(){
    char ch=chk();
    if (ch=='a'){
        printf("STACK EMPTY\n");
        return;
    }
    printf("Stack contents:\n");
    for (int i=top; i>=0; i--)
        printf("%d\n",stk[i]);
}

void main(){
    while (1){
        printf("\n1. Push \n2. Pop \n3. Display \n4. Exit\n");
        printf("Enter choice: ");
        scanf("%d",&ch);
        switch(ch){
            case 1: push();
                    break;
            case 2: pop();
                    break;
            case 3: display();
                    break;
            case 4: exit(0);
        }
    }
}

```

Output

```
1. Push
2. Pop
3. Display
4. Exit
Enter choice: 1
Enter element: 11
Element pushed into stack

1. Push
2. Pop
3. Display
4. Exit
Enter choice: 1
Enter element: 22
Element pushed into stack

1. Push
2. Pop
3. Display
4. Exit
Enter choice: 1
Enter element: 33
Element pushed into stack

1. Push
2. Pop
3. Display
4. Exit
Enter choice: 3
Stack contents:
33
22
11
```

```
1. Push
2. Pop
3. Display
4. Exit
Enter choice: 2
33
Element popped from stack

1. Push
2. Pop
3. Display
4. Exit
Enter choice: 2
22
Element popped from stack

1. Push
2. Pop
3. Display
4. Exit
Enter choice: 2
11
Element popped from stack

1. Push
2. Pop
3. Display
4. Exit
Enter choice: 3
STACK EMPTY

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

Q) 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).

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

int pos=0,top=-1,length;
char symbol,temp,infix[20],postfix[20],stack[20];

void push(char symbol){
    top = top +1;
    stack[top]=symbol;
}

char pop(){
    char symb;
    symb = stack[top];
    top = top-1;
    return (symb);
}

int pred(char symbol){
    int p;
    switch(symbol){
        case '^': p=3;
                break;
        case '*':
        case '/': p=2;
                break;
        case '+':
        case '-': p=1;
                break;
        case '(': p=0;
                break;
        case '#': p=-1;
                break;
    }
    return(p);
}
```



```

}

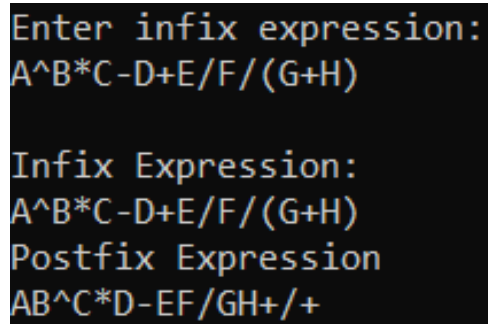
void infixtopostfix(){
    length = strlen(infix);
    int index=0;
    push('#');
    while(index<length){
        symbol = infix[index];
        switch(symbol){
            case '(': push(symbol);
            break;
            case ')': temp = pop();
            while(temp!='('){
                postfix[pos]=temp;
                pos++;
                temp = pop();
            }
            break;
            case '+':
            case '-':
            case '*':
            case '/':
            case '^': while(pred(stack[top])>=pred(symbol)){
                temp = pop();
                postfix[pos++]=temp;
            }
            push(symbol);
            break;
            default : postfix[pos++]=symbol;
        }
        index++;
    }
    while(top>0){
        temp=pop();
        postfix[pos++]=temp;
    }
}

void main(){
    printf("Enter infix expression:\n");

```

```
scanf("%s",infix);
infixtopostfix();
printf("\nInfix Expression: \n%s",infix);
printf("\nPostfix Expression \n%s",postfix);
}
```

Output



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

Infix Expression:
A^B*C-D+E/F/(G+H)
Postfix Expression
AB^C*D-EF/GH+/+

Q) Write a program to simulate the working of queue using an array with the following:

a) Insert

b) Delete

c) Display.

The program should print appropriate messages for queue empty and queue full.

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

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

```
#define MAX 4
```

```
int ch, queue[4], front=-1, rear=-1, element, i;
```

```
void insert(){
```

```
    if (rear==MAX-1){  
        printf("Queue full\n");  
        return;  
    }
```

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

```
    else
```

```
        rear++;
```

```
    printf("Enter element: ");
```

```
    scanf("%d",&element);
```

```
    queue[rear]=element;
```

```
}
```

```
void delete(){
```

```
    if (rear==-1 && front==-1){  
        printf("Queue empty");  
        return;  
    }
```

```
    element=queue[front];
```

```
    if (front==rear){
```

```
        front=-1;
```

```
        rear=-1;
```

```
    }
```

```
    else
```

```

        front++;
        printf("Deleted element = %d",element);
    }

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

void main(){
    while (1){
        printf("\n1.Insert \n2.Delete \n3.Display \n4.Exit");
        printf("\nEnter choice: ");
        scanf("%d",&ch);
        switch(ch){
            case 1: insert();
                    break;
            case 2: delete();
                    break;
            case 3: display();
                    break;
            case 4: exit(0);
        }
    }
}

```

Output

```
1.Insert
2.Delete
3.Display
4.Exit
Enter choice: 1
Enter element: 11
```

```
1.Insert
2.Delete
3.Display
4.Exit
Enter choice: 1
Enter element: 22
```

```
1.Insert
2.Delete
3.Display
4.Exit
Enter choice: 1
Enter element: 33
```

```
1.Insert
2.Delete
3.Display
4.Exit
Enter choice: 1
Enter element: 44
```

```
1.Insert
2.Delete
3.Display
4.Exit
Enter choice: 1
Queue full
```

```
1.Insert
2.Delete
3.Display
4.Exit
Enter choice: 3
Queue contents:
11
22
33
44
```

```
1.Insert
2.Delete
3.Display
4.Exit
Enter choice: 2
Deleted element = 11
1.Insert
2.Delete
3.Display
4.Exit
Enter choice: 2
Deleted element = 22
1.Insert
2.Delete
3.Display
4.Exit
Enter choice: 2
Deleted element = 33
1.Insert
2.Delete
3.Display
4.Exit
Enter choice: 2
Deleted element = 44
1.Insert
2.Delete
3.Display
4.Exit
Enter choice: 2
Queue empty
1.Insert
2.Delete
3.Display
4.Exit
Enter choice: 4
```

LeetCode Problem

Q) Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

```
bool isValid(char* s) {
    int n = strlen(s);
    char stack[n];
    int top = -1;
    int i = 0;
    while (i < n){
        char c = s[i];
        if (c == '(' || c == '{' || c == '[')
            stack[++top] = c;
        else{
            if (top == -1)
                return false;
            char topChar = stack[top--];
            if ((c == ')' && topChar != '(') || (c == '}' && topChar != '{') || (c == ']' &&
topChar != '['))
                return false;
        }
        i++;
    }
    return top == -1;
}
```

Output

• Case 1

• Case 2

• Case 3

• Case 4

Input

s =
"()"

Output

true

Expected

true

• Case 1

• Case 2

• Case 3

• Case 4

Input

s =
"()[]{}"

Output

true

Expected

true

• Case 1

• Case 2

• Case 3

• Case 4

Input

s =
"()"

Output

false

Expected

false

• Case 1

• Case 2

• Case 3

• Case 4

Input

s =
"([])"

Output

true

Expected

true

Q) Write a program to simulate the working of circular queue using an array with the following:

a) Insert

b) Delete

c) Display.

The program should print appropriate messages for queue empty and queue full.

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

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

```
#define MAX 3
```

```
int cq[3], front=-1, rear=-1, ch, element, i;
```

```
void insert();
```

```
void delete();
```

```
void display();
```

```
void main(){
```

```
    while (1){
```

```
        printf("\n1. Insert \n2. Delete \n3. Display \n4. Exit");
```

```
        printf("\nEnter choice: ");
```

```
        scanf("%d",&ch);
```

```
        switch(ch){
```

```
            case 1: insert();
```

```
                break;
```

```
            case 2: delete();
```

```
                break;
```

```
            case 3: display();
```

```
                break;
```

```
            case 4: exit(0);
```

```
        }
```

```
    }
```

```
}
```

```
void insert(){
```

```
    if (front==(rear+1)%MAX){
```

```
        printf("Circular Queue full\n");
```

```
        return;
```

```
    }
```

```
    printf("Enter element: ");
```

```

scanf("%d",&element);
if (rear==-1 && front==-1){
    front=0;
    rear=0;
}
else
    rear=(rear+1)%MAX;
cq[rear]=element;
}

void delete(){
    if (front==-1 && rear==-1){
        printf("Circular Queue empty\n");
        return;
    }
    element=cq[front];
    printf("Deleted element = %d\n",element);
    if (front==rear){
        front=-1;
        rear=-1;
    }
    else
        front=(front+1)%MAX;
}

void display(){
    if (front==-1 && rear==-1){
        printf("Circular Queue empty\n");
        return;
    }
    printf("Contents are:\n");
    if (front<=rear){
        for(i=front;i<=rear;i++)
            printf("%d\n",cq[i]);
    }
    else{
        for(i=front;i<=MAX-1;i++)
            printf("%d\n",cq[i]);
        for(i=0;i<=rear;i++)
            printf("%d\n",cq[i]);
    }
}

```

```
}  
}
```

Output

```
1. Insert  
2. Delete  
3. Display  
4. Exit  
Enter choice: 1  
Enter element: 11  
  
1. Insert  
2. Delete  
3. Display  
4. Exit  
Enter choice: 1  
Enter element: 22  
  
1. Insert  
2. Delete  
3. Display  
4. Exit  
Enter choice: 1  
Enter element: 33  
  
1. Insert  
2. Delete  
3. Display  
4. Exit  
Enter choice: 1  
Circular Queue full  
  
1. Insert  
2. Delete  
3. Display  
4. Exit  
Enter choice: 3  
Contents are:  
11  
22  
33  
  
1. Insert  
2. Delete  
3. Display  
4. Exit  
Enter choice: 2  
Deleted element = 11
```

```
1. Insert
2. Delete
3. Display
4. Exit
Enter choice: 2
Deleted element = 22
```

```
1. Insert
2. Delete
3. Display
4. Exit
Enter choice: 2
Deleted element = 33
```

```
1. Insert
2. Delete
3. Display
4. Exit
Enter choice: 2
Circular Queue empty
```

```
1. Insert
2. Delete
3. Display
4. Exit
Enter choice: 4
```

LeetCode Problem

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

```
int firstUniqChar(char* s) {  
    int freq[26] = {0}; // Array to store frequency of each character ('a' to 'z')  
  
    // Traverse the string to count the frequency of each character  
    for (int i = 0; s[i] != '\0'; i++) {  
        freq[s[i] - 'a']++; // Increment the count for the character s[i]  
    }  
  
    // Traverse the string again to find the first character with frequency 1  
    for (int i = 0; s[i] != '\0'; i++) {  
        if (freq[s[i] - 'a'] == 1) {  
            return i; // Return the index of the first non-repeating character  
        }  
    }  
  
    return -1; // If no non-repeating character, return -1  
}
```

Output



☒ Testcase | [Test Result](#)

• Case 1

• Case 2

• Case 3

Input

s =
"loveleetcode"

Output

2

Expected

2

☒ Testcase | [Test Result](#)

• Case 1

• Case 2

• Case 3

Input

s =
"aabb"

Output

-1

Expected

-1

Q) WAP to Implement Singly Linked List with following operations :

a) Create a linked list.

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

c) Display the contents of the linked list.

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

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

```
struct Node {  
    int data;  
    struct Node *link;  
};
```

```
typedef struct Node node;
```

```
node *start = NULL;
```

```
node *new1, *curr, *ptr;
```

```
void create();
```

```
void display();
```

```
void InsertStart();
```

```
void InsertPosition();
```

```
void InsertEnd();
```

```
void main() {
```

```
    int ch;
```

```
    while (1) {
```

```
        printf("\n1. Create \n2. Display \n3. Insert at Beginning \n4. Insert at Position  
\n5. Insert at End \n6. Exit");
```

```
        printf("\nEnter Your Choice: ");
```

```
        scanf("%d", &ch);
```

```
        switch (ch) {
```

```
            case 1:
```

```
                create();
```

```
                break;
```

```
            case 2:
```

```
                display();
```

```
                break;
```

```
            case 3:
```

```
                InsertStart();
```

```

        break;
    case 4:
        InsertPosition();
        break;
    case 5:
        InsertEnd();
        break;
    case 6:
        exit(0);
    default:
        printf("Enter a Number between 1 and 6.\n");
    }
}
}

```

```

void create() {
    char ch;

    do {
        new1 = (node*)malloc(sizeof(node));
        printf("\nEnter value: ");
        scanf("%d",&new1->data);
        if (start==NULL)
        {
            start=new1;
            curr=new1;
        }
        else {
            curr->link = new1;
            curr=new1;
        }

        printf("Do You Want to Add an Element (Y/N)? ");
        scanf(" %c", &ch);
    } while (ch == 'y' || ch == 'Y');
    curr->link=NULL;
}

```

```

void InsertStart() {
    new1 = (node*)malloc(sizeof(node));

```



```

printf("\nEnter value: ");
scanf("%d",&new1->data);
if(start==NULL)
{
    start=new1;
    new1->link=NULL;
    return;
}
else {
    new1->link=start;
    start=new1;
    return;
}
}

void InsertPosition() {
    new1 = (node*)malloc(sizeof(node));
    printf("\nEnter value: ");
    scanf("%d",&new1->data);
    if(start==NULL)
    {
        start=new1;
        new1->link=NULL;
        return;
    }

    int i=1, pos;
    ptr=start;
    printf("\nEnter position: ");
    scanf("%d",&pos);
    while (ptr!=NULL && i<pos-1)
    {
        ptr=ptr->link;
        i++;
    }
    if(ptr==NULL)
    {
        return;
    }
}

```

```

    new1->link=ptr->link;
    ptr->link=new1;
}

void InsertEnd() {
    new1 = (node*)malloc(sizeof(node));
    printf("\nEnter value: ");
    scanf("%d",&new1->data);
    if(start==NULL)
    {
        start=new1;
        new1->link=NULL;
        return;
    }

    ptr=start;
    while(ptr->link !=NULL)
    {
        ptr=ptr->link;
    }
    ptr->link=new1;
    new1->link=NULL;
    return;
}

void display() {
    if (start == NULL) {
        printf("\nLinked List is Empty.\n");
        return;
    }

    ptr = start;
    printf("\nElements in Linked List: \n");

    while (ptr != NULL) {
        printf("%d ", ptr->data);
        ptr = ptr->link;
    }
    printf("\n");
}

```

Output

```
1. Create
2. Display
3. Insert at Beginning
4. Insert at Position
5. Insert at End
6. Exit
Enter Your Choice: 1

Enter value: 11
Do You Want to Add an Element (Y/N)? Y

Enter value: 22
Do You Want to Add an Element (Y/N)? N

1. Create
2. Display
3. Insert at Beginning
4. Insert at Position
5. Insert at End
6. Exit
Enter Your Choice: 2

Elements in Linked List:
11 22

1. Create
2. Display
3. Insert at Beginning
4. Insert at Position
5. Insert at End
6. Exit
Enter Your Choice: 3

Enter value: 33

1. Create
2. Display
3. Insert at Beginning
4. Insert at Position
5. Insert at End
6. Exit
Enter Your Choice: 4

Enter value: 44

Enter position: 1
```

1. Create
2. Display
3. Insert at Beginning
4. Insert at Position
5. Insert at End
6. Exit

Enter Your Choice: 5

Enter value: 55

1. Create
2. Display
3. Insert at Beginning
4. Insert at Position
5. Insert at End
6. Exit

Enter Your Choice: 2

Elements in Linked List:

33 44 11 22 55

1. Create
2. Display
3. Insert at Beginning
4. Insert at Position
5. Insert at End
6. Exit

Enter Your Choice: 6

Q)WAP to Implement Singly Linked List with following operations:

a) Create a linked list.

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

c) Display the contents of the linked list.

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

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

```
struct Node {  
    int data;  
    struct Node *link;  
};
```

```
typedef struct Node node;  
node *start = NULL;
```

```
void create();  
void display();  
void DeletefromStart();  
void DeleteatPosition();  
void DeleteatEnd();
```

```
void main() {  
    int ch;  
    while (1) {  
        printf("\n1. Create \n2. Display \n3. Delete from Beginning \n4. Delete at  
Position \n5. Delete at End \n6. Exit");  
        printf("\nEnter Your Choice: ");  
        scanf("%d", &ch);  
  
        switch (ch) {  
            case 1:  
                create();  
                break;  
            case 2:  
                display();  
                break;  
            case 3:  
                DeletefromStart();  
                break;
```

```

        case 4:
            DeleteatPosition();
            break;
        case 5:
            DeleteatEnd();
            break;
        case 6:
            exit(0);
        default:
            printf("Enter a Number between 1 and 9.\n");
    }
}
}

```

```

void create() {
    char ch;
    node *new1, *curr;

    do {
        new1 = (node*)malloc(sizeof(node));
        printf("\nEnter value: ");
        scanf("%d",&new1->data);
        if (start==NULL)
        {
            start=new1;
            curr=new1;
        }
        else {
            curr->link = new1;
            curr=new1;
        }

        printf("Do You Want to Add an Element (Y/N)? ");
        scanf(" %c", &ch);
    } while (ch == 'y' || ch == 'Y');
    curr->link=NULL;
}

```

```

void display() {
    if (start == NULL) {

```

```

    printf("\nLinked List is Empty.\n");
    return;
}

node *temp = start;
printf("\nElements in Linked List: \n");

while (temp != NULL) {
    printf("%d ", temp->data);
    temp = temp->link;
}
printf("\n");
}

void DeletefromStart() {
    if (start == NULL) {
        printf("\nLinked List is Empty.\n");
        return;
    }

    node *temp = start;
    start = start->link;
    free(temp);
    printf("\nFirst element deleted successfully.\n");
}

void DeleteatPosition() {
    int pos, i = 1;
    if (start == NULL) {
        printf("\nLinked List is Empty.\n");
        return;
    }

    printf("\nEnter the position to delete: ");
    scanf("%d", &pos);

    node *temp = start;
    node *prev = NULL;

    if (pos == 1) {

```

```

    start = temp->link;
    free(temp);
    printf("\nElement at position %d deleted successfully.\n", pos);
    return;
}

while (temp != NULL && i < pos) {
    prev = temp;
    temp = temp->link;
    i++;
}

if (temp == NULL) {
    printf("\nPosition not found.\n");
    return;
}

prev->link = temp->link;
free(temp);
printf("\nElement at position %d deleted successfully.\n", pos);
}

void DeleteatEnd() {
    if (start == NULL) {
        printf("\nLinked List is Empty.\n");
        return;
    }

    node *temp = start;
    node *prev = NULL;

    if (start->link == NULL) {
        start = NULL;
        free(temp);
        printf("\nLast element deleted successfully.\n");
        return;
    }

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

```



```

    temp = temp->link;
}

prev->link = NULL;
free(temp);
printf("\nLast element deleted successfully.\n");
}

```

Output

```

1. Create
2. Display
3. Delete from Beginning
4. Delete at Position
5. Delete at End
6. Exit
Enter Your Choice: 1

Enter value: 11
Do You Want to Add an Element (Y/N)? Y

Enter value: 22
Do You Want to Add an Element (Y/N)? Y

Enter value: 33
Do You Want to Add an Element (Y/N)? Y

Enter value: 44
Do You Want to Add an Element (Y/N)? Y

Enter value: 55
Do You Want to Add an Element (Y/N)? N

1. Create
2. Display
3. Delete from Beginning
4. Delete at Position
5. Delete at End
6. Exit
Enter Your Choice: 2

Elements in Linked List:
11 22 33 44 55

1. Create
2. Display
3. Delete from Beginning
4. Delete at Position
5. Delete at End
6. Exit
Enter Your Choice: 3

First element deleted successfully.

```

```
1. Create
2. Display
3. Delete from Beginning
4. Delete at Position
5. Delete at End
6. Exit
Enter Your Choice: 4

Enter the position to delete: 2

Element at position 2 deleted successfully.

1. Create
2. Display
3. Delete from Beginning
4. Delete at Position
5. Delete at End
6. Exit
Enter Your Choice: 5

Last element deleted successfully.

1. Create
2. Display
3. Delete from Beginning
4. Delete at Position
5. Delete at End
6. Exit
Enter Your Choice: 2

Elements in Linked List:
22 44

1. Create
2. Display
3. Delete from Beginning
4. Delete at Position
5. Delete at End
6. Exit
Enter Your Choice: 6
```

Q) WAP to Implement Single Link List with following operations:

a) Sort the linked list

b) Reverse the linked list

c) Concatenation of two linked lists.

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

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

```
struct node {  
    int value;  
    struct node *next;  
};
```

```
typedef struct node* NODE;
```

```
NODE newnode() {  
    NODE new_node = (NODE)malloc(sizeof(struct node));  
    return new_node;  
}
```

```
NODE insert_end(int item, NODE first) {  
    NODE new_end = newnode();  
    new_end->value = item;  
    new_end->next = NULL;  
  
    if (first == NULL) {  
        return new_end;  
    }
```

```
    NODE current = first;  
    while (current->next != NULL) {  
        current = current->next;  
    }  
    current->next = new_end;  
    return first;  
}
```

```
NODE reverse(NODE first){  
    NODE current,temp;
```

```

current=NULL;
while (first!=NULL){
    temp=first;
    first=first->next;
    temp->next=current;
    current=temp;
}
return current;
}

```

```

NODE concatenate(NODE first_1, NODE first_2){
    NODE last1;
    if (first_1==NULL && first_2==NULL){
        return NULL;
    }
    else if(first_1==NULL){
        return first_2;
    }
    else if (first_2==NULL){
        return first_1;
    }
    else{
        last1=first_1;
        while(last1->next!=NULL){
            last1=last1->next;
        }
        last1->next=first_2;
    }
    return first_1;
}

```

```

NODE sort(NODE first){
    NODE temp1,temp2,temp3;
    int swap;
    do{
        swap=0;
        temp1=first;
        while (temp1 != NULL && temp1->next!=NULL){
            if (temp1->value > temp1->next->value){
                temp3=temp1->next;

```

```

        temp1->next = temp3->next;
        temp3->next=temp1;
        if (temp1==first)
            first=temp3;
        else
            temp2->next=temp3;
        swap=1;
        temp2=temp3;
    }
    else{
        temp2=temp1;
        temp1=temp1->next;
    }
}
while(swap);
return(first);
}

```

```

void display(NODE first)
{ NODE temp;

    if(first==NULL)
    {

        printf("linked list is empty");
        return;
    }
    temp=first;
    while(temp!=NULL)
    {

        printf("%d\t",temp->value);
        temp=temp->next;
    }
}

```

```

void main() {
    NODE first_1= NULL;
    NODE first_2= NULL;

```

```

int choice, item, pos;
while (1){
    printf("\nMenu:\n");
    printf("1. Insert in linked list 1\n");
    printf("2. Insert in linked list 2\n");
    printf("3. Sort linked list 1\n");
    printf("4. Sort linked list 2\n");
    printf("5. Reverse linked list 1\n");
    printf("6. Reverse linked list 2\n");
    printf("7. Concatenate the two linked lists\n");
    printf("8. Display LL 1\n");
    printf("9. Display LL 2\n");
    printf("10. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("Enter value to insert: ");
            scanf("%d", &item);
            first_1 = insert_end(item, first_1);
            break;
        case 2:
            printf("Enter value to insert: ");
            scanf("%d", &item);
            first_2 = insert_end(item, first_2);
            break;
        case 3:
            printf("Sorted Linked list 1");
            first_1=sort(first_1);
            break;
        case 4:
            printf("Sorted Linked list 2");
            first_2=sort(first_2);
            break;
        case 5:
            printf("Linked list 1 reversed");
            first_1 =reverse(first_1);
            break;
        case 6:

```

```

        printf("Linked list 2 reversed");
        first_2=reverse (first_2);
        break;
    case 7:
        first_1=concatenate (first_1,first_2);
        break;
    case 8:
        display(first_1);
        break;
    case 9:
        display(first_2);
        break;
    case 10:
        exit(0);
    }
}
}

```

Output

```

Menu:
1. Insert in linked list 1
2. Insert in linked list 2
3. Sort linked list 1
4. Sort linked list 2
5. Reverse linked list 1
6. Reverse linked list 2
7. Concatenate the two linked lists
8. Display LL 1
9. Display LL 2
10. Exit
Enter your choice: 1
Enter value to insert: 11

```

```

Menu:
1. Insert in linked list 1
2. Insert in linked list 2
3. Sort linked list 1
4. Sort linked list 2
5. Reverse linked list 1
6. Reverse linked list 2
7. Concatenate the two linked lists
8. Display LL 1
9. Display LL 2
10. Exit
Enter your choice: 1
Enter value to insert: 44

```

Menu:

1. Insert in linked list 1
2. Insert in linked list 2
3. Sort linked list 1
4. Sort linked list 2
5. Reverse linked list 1
6. Reverse linked list 2
7. Concatenate the two linked lists
8. Display LL 1
9. Display LL 2
10. Exit

Enter your choice: 1

Enter value to insert: 33

Menu:

1. Insert in linked list 1
2. Insert in linked list 2
3. Sort linked list 1
4. Sort linked list 2
5. Reverse linked list 1
6. Reverse linked list 2
7. Concatenate the two linked lists
8. Display LL 1
9. Display LL 2
10. Exit

Enter your choice: 1

Enter value to insert: 22

Menu:

1. Insert in linked list 1
2. Insert in linked list 2
3. Sort linked list 1
4. Sort linked list 2
5. Reverse linked list 1
6. Reverse linked list 2
7. Concatenate the two linked lists
8. Display LL 1
9. Display LL 2
10. Exit

Enter your choice: 8

11 44 33 22

Menu:

1. Insert in linked list 1
2. Insert in linked list 2
3. Sort linked list 1
4. Sort linked list 2
5. Reverse linked list 1
6. Reverse linked list 2
7. Concatenate the two linked lists
8. Display LL 1
9. Display LL 2
10. Exit

Enter your choice: 2

Enter value to insert: 66

Menu:

1. Insert in linked list 1
2. Insert in linked list 2
3. Sort linked list 1
4. Sort linked list 2
5. Reverse linked list 1
6. Reverse linked list 2
7. Concatenate the two linked lists
8. Display LL 1
9. Display LL 2
10. Exit

Enter your choice: 2

Enter value to insert: 55

Menu:

1. Insert in linked list 1
2. Insert in linked list 2
3. Sort linked list 1
4. Sort linked list 2
5. Reverse linked list 1
6. Reverse linked list 2
7. Concatenate the two linked lists
8. Display LL 1
9. Display LL 2
10. Exit

Enter your choice: 2

Enter value to insert: 88

Menu:

1. Insert in linked list 1
2. Insert in linked list 2
3. Sort linked list 1
4. Sort linked list 2
5. Reverse linked list 1
6. Reverse linked list 2
7. Concatenate the two linked lists
8. Display LL 1
9. Display LL 2
10. Exit

Enter your choice: 2

Enter value to insert: 77

Menu:

1. Insert in linked list 1
2. Insert in linked list 2
3. Sort linked list 1
4. Sort linked list 2
5. Reverse linked list 1
6. Reverse linked list 2
7. Concatenate the two linked lists
8. Display LL 1
9. Display LL 2
10. Exit

Enter your choice: 9

66 55 88 77

Menu:

1. Insert in linked list 1
2. Insert in linked list 2
3. Sort linked list 1
4. Sort linked list 2
5. Reverse linked list 1
6. Reverse linked list 2
7. Concatenate the two linked lists
8. Display LL 1
9. Display LL 2
10. Exit

Enter your choice: 3

Sorted Linked list 1

Menu:

1. Insert in linked list 1
2. Insert in linked list 2
3. Sort linked list 1
4. Sort linked list 2
5. Reverse linked list 1
6. Reverse linked list 2
7. Concatenate the two linked lists
8. Display LL 1
9. Display LL 2
10. Exit

Enter your choice: 4

Sorted Linked list 2

Menu:

1. Insert in linked list 1
2. Insert in linked list 2
3. Sort linked list 1
4. Sort linked list 2
5. Reverse linked list 1
6. Reverse linked list 2
7. Concatenate the two linked lists
8. Display LL 1
9. Display LL 2
10. Exit

Enter your choice: 8

11 22 33 44

Menu:

1. Insert in linked list 1
2. Insert in linked list 2
3. Sort linked list 1
4. Sort linked list 2
5. Reverse linked list 1
6. Reverse linked list 2
7. Concatenate the two linked lists
8. Display LL 1
9. Display LL 2
10. Exit

Enter your choice: 9

55 66 77 88

Menu:

1. Insert in linked list 1
2. Insert in linked list 2
3. Sort linked list 1
4. Sort linked list 2
5. Reverse linked list 1
6. Reverse linked list 2
7. Concatenate the two linked lists
8. Display LL 1
9. Display LL 2
10. Exit

Enter your choice: 5

Linked list 1 reversed

Menu:

1. Insert in linked list 1
2. Insert in linked list 2
3. Sort linked list 1
4. Sort linked list 2
5. Reverse linked list 1
6. Reverse linked list 2
7. Concatenate the two linked lists
8. Display LL 1
9. Display LL 2
10. Exit

Enter your choice: 6

Linked list 2 reversed

Menu:

1. Insert in linked list 1
2. Insert in linked list 2
3. Sort linked list 1
4. Sort linked list 2
5. Reverse linked list 1
6. Reverse linked list 2
7. Concatenate the two linked lists
8. Display LL 1
9. Display LL 2
10. Exit

Enter your choice: 8

44 33 22 11

Menu:

1. Insert in linked list 1
2. Insert in linked list 2
3. Sort linked list 1
4. Sort linked list 2
5. Reverse linked list 1
6. Reverse linked list 2
7. Concatenate the two linked lists
8. Display LL 1
9. Display LL 2
10. Exit

Enter your choice: 9

88 77 66 55

Menu:

1. Insert in linked list 1
2. Insert in linked list 2
3. Sort linked list 1
4. Sort linked list 2
5. Reverse linked list 1
6. Reverse linked list 2
7. Concatenate the two linked lists
8. Display LL 1
9. Display LL 2
10. Exit

Enter your choice: 7

Menu:

1. Insert in linked list 1
2. Insert in linked list 2
3. Sort linked list 1
4. Sort linked list 2
5. Reverse linked list 1
6. Reverse linked list 2
7. Concatenate the two linked lists
8. Display LL 1
9. Display LL 2
10. Exit

Enter your choice: 8

44 33 22 11 88 77 66 55

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

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

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

typedef struct node* NODE;

NODE newnode(int item) {
    NODE new_node = (NODE)malloc(sizeof(struct node));
    new_node->value = item;
    new_node->next = NULL;
    return new_node;
}

NODE push(NODE top, int item) {
    NODE new_top = newnode(item);
    new_top->next = top;
    return new_top;
}

NODE pop(NODE top) {
    if (top == NULL) {
        printf("Stack is empty, cannot pop\n");
        return top;
    }
    printf("Popped element: %d",top->value);
    NODE temp = top;
    top = top->next;
    free(temp);
    return top;
}

void display_stack(NODE top) {
    if (top == NULL) {
        printf("Stack is empty\n");
    }
}
```

```

        return;
    }
    NODE temp = top;
    printf("Stack:\n");
    while (temp != NULL) {
        printf("%d\n", temp->value);
        temp = temp->next;
    }
    printf("\n");
}

```

```

NODE enqueue(NODE rear, int item) {
    NODE new_node = newnode(item);
    if (rear == NULL) {
        return new_node;
    }
    rear->next = new_node;
    return new_node;
}

```

```

NODE dequeue(NODE *front) {
    if (*front == NULL) {
        printf("Queue is empty, cannot dequeue\n");
        return *front;
    }
    printf("Dequeued element: %d",(*front)->value);
    NODE temp = *front;
    *front = (*front)->next;
    free(temp);
    return *front;
}

```

```

void display_queue(NODE front) {
    if (front == NULL) {
        printf("Queue is empty\n");
        return;
    }
    NODE temp = front;
    printf("Queue: ");
    while (temp != NULL) {

```

```

        printf("%d ", temp->value);
        temp = temp->next;
    }
    printf("\n");
}

void main() {
    NODE stack_top = NULL;
    NODE queue_front = NULL, queue_rear = NULL;

    int choice, item;

    while (1) {
        printf("\nMenu:\n");
        printf("1. Push onto Stack\n");
        printf("2. Pop from Stack\n");
        printf("3. Display Stack\n");
        printf("4. Enqueue into Queue\n");
        printf("5. Dequeue from Queue\n");
        printf("6. Display Queue\n");
        printf("7. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to push onto Stack: ");
                scanf("%d", &item);
                stack_top = push(stack_top, item);
                break;
            case 2:
                stack_top = pop(stack_top);
                break;
            case 3:
                display_stack(stack_top);
                break;
            case 4:
                printf("Enter value to enqueue into Queue: ");
                scanf("%d", &item);
                queue_rear = enqueue(queue_rear, item);

```



```

        if (queue_front == NULL) {
            queue_front = queue_rear;
        }
        break;
    case 5:
        queue_front = dequeue(&queue_front);
        if (queue_front == NULL) {
            queue_rear = NULL;
        }
        break;
    case 6:
        display_queue(queue_front);
        break;
    case 7:
        exit(0);
    }
}
}

```

Output

```

Menu:
1. Push onto Stack
2. Pop from Stack
3. Display Stack
4. Enqueue into Queue
5. Dequeue from Queue
6. Display Queue
7. Exit
Enter your choice: 1
Enter value to push onto Stack: 11

Menu:
1. Push onto Stack
2. Pop from Stack
3. Display Stack
4. Enqueue into Queue
5. Dequeue from Queue
6. Display Queue
7. Exit
Enter your choice: 1
Enter value to push onto Stack: 22

```

```
Menu:
1. Push onto Stack
2. Pop from Stack
3. Display Stack
4. Enqueue into Queue
5. Dequeue from Queue
6. Display Queue
7. Exit
Enter your choice: 1
Enter value to push onto Stack: 33
```

```
Menu:
1. Push onto Stack
2. Pop from Stack
3. Display Stack
4. Enqueue into Queue
5. Dequeue from Queue
6. Display Queue
7. Exit
Enter your choice: 3
Stack:
33
22
11
```

```
Menu:
1. Push onto Stack
2. Pop from Stack
3. Display Stack
4. Enqueue into Queue
5. Dequeue from Queue
6. Display Queue
7. Exit
Enter your choice: 2
Popped element: 33
Menu:
1. Push onto Stack
2. Pop from Stack
3. Display Stack
4. Enqueue into Queue
5. Dequeue from Queue
6. Display Queue
7. Exit
Enter your choice: 2
Popped element: 22
```

Menu:

1. Push onto Stack
2. Pop from Stack
3. Display Stack
4. Enqueue into Queue
5. Dequeue from Queue
6. Display Queue
7. Exit

Enter your choice: 2

Popped element: 11

Menu:

1. Push onto Stack
2. Pop from Stack
3. Display Stack
4. Enqueue into Queue
5. Dequeue from Queue
6. Display Queue
7. Exit

Enter your choice: 2

Stack is empty, cannot pop

Menu:

1. Push onto Stack
2. Pop from Stack
3. Display Stack
4. Enqueue into Queue
5. Dequeue from Queue
6. Display Queue
7. Exit

Enter your choice: 4

Enter value to enqueue into Queue: 44

Menu:

1. Push onto Stack
2. Pop from Stack
3. Display Stack
4. Enqueue into Queue
5. Dequeue from Queue
6. Display Queue
7. Exit

Enter your choice: 4

Enter value to enqueue into Queue: 55

Menu:

1. Push onto Stack
2. Pop from Stack
3. Display Stack
4. Enqueue into Queue
5. Dequeue from Queue
6. Display Queue
7. Exit

Enter your choice: 4

Enter value to enqueue into Queue: 66

Menu:

1. Push onto Stack
2. Pop from Stack
3. Display Stack
4. Enqueue into Queue
5. Dequeue from Queue
6. Display Queue
7. Exit

Enter your choice: 6

Queue: 44 55 66

Menu:

1. Push onto Stack
2. Pop from Stack
3. Display Stack
4. Enqueue into Queue
5. Dequeue from Queue
6. Display Queue
7. Exit

Enter your choice: 5

Dequeued element: 44

Menu:

1. Push onto Stack
2. Pop from Stack
3. Display Stack
4. Enqueue into Queue
5. Dequeue from Queue
6. Display Queue
7. Exit

Enter your choice: 5

Dequeued element: 55

Menu:

1. Push onto Stack
2. Pop from Stack
3. Display Stack
4. Enqueue into Queue
5. Dequeue from Queue
6. Display Queue
7. Exit

Enter your choice: 5

Dequeued element: 66

Menu:

1. Push onto Stack
2. Pop from Stack
3. Display Stack
4. Enqueue into Queue
5. Dequeue from Queue
6. Display Queue
7. Exit

Enter your choice: 5

Queue is empty, cannot dequeue

Menu:

1. Push onto Stack
2. Pop from Stack
3. Display Stack
4. Enqueue into Queue
5. Dequeue from Queue
6. Display Queue
7. Exit

Enter your choice: 7

Q) Create a program to stimulate working of doubly linked list through the following functions:

- a) Create**
- b) Insert to the left of specific element**
- c) Delete specific element**
- d) Display doubly linked list elements**

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
    struct Node* prev;
};

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

void insertLeft(struct Node** head, int newData, int targetData) {
    struct Node* current = *head;
    while (current != NULL && current->data != targetData) {
        current = current->next;
    }

    if (current == NULL) {
        printf("Node with value %d not found.\n", targetData);
        return;
    }

    struct Node* newNode = createNode(newData);
    newNode->next = current;
    newNode->prev = current->prev;

    if (current->prev != NULL) {
        current->prev->next = newNode;
    }
```

```

    } else {
        *head = newNode;
    }
    current->prev = newNode;
}

void deleteNode(struct Node** head, int value) {
    struct Node* current = *head;

    while (current != NULL && current->data != value) {
        current = current->next;
    }

    if (current == NULL) {
        printf("Node with value %d not found.\n", value);
        return;
    }

    if (current->prev != NULL) {
        current->prev->next = current->next;
    } else {
        *head = current->next;
    }

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

    free(current);
}

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

```

```

void createList(struct Node** head) {
    int n, data;
    printf("Enter the number of nodes: ");
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        printf("Enter data for node %d: ", i + 1);
        scanf("%d", &data);
        struct Node* newNode = createNode(data);
        if (*head == NULL) {
            *head = newNode;
        } else {
            struct Node* temp = *head;
            while (temp->next != NULL) {
                temp = temp->next;
            }
            temp->next = newNode;
            newNode->prev = temp;
        }
    }
}

void main() {
    struct Node* head = NULL;
    int choice, data, targetData;

    while (1) {
        printf("\nMenu:\n");
        printf("1. Create a doubly linked list\n");
        printf("2. Insert a new node to the left of a specified node\n");
        printf("3. Delete a node based on a specific value\n");
        printf("4. Display the contents of the list\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                createList(&head);
                break;
            case 2:

```



```

        printf("Enter the value to insert: ");
        scanf("%d", &data);
        printf("Enter the target node value to insert left of: ");
        scanf("%d", &targetData);
        insertLeft(&head, data, targetData);
        break;
    case 3:
        printf("Enter the value of the node to delete: ");
        scanf("%d", &data);
        deleteNode(&head, data);
        break;
    case 4:
        printf("Doubly linked list contents: ");
        displayList(head);
        break;
    case 5:
        exit(0);
    default:
        printf("Invalid choice! Please try again.\n");
    }
}
}

```

Output

```

Menu:
1. Create a doubly linked list
2. Insert a new node to the left of a specified node
3. Delete a node based on a specific value
4. Display the contents of the list
5. Exit
Enter your choice: 1
Enter the number of nodes: 2
Enter data for node 1: 11
Enter data for node 2: 22

```

Menu:

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

Enter your choice: 4

Doubly linked list contents: 11 22

Menu:

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

Enter your choice: 2

Enter the value to insert: 33

Enter the target node value to insert left of: 22

Menu:

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

Enter your choice: 3

Enter the value of the node to delete: 22

Menu:

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

Enter your choice: 4

Doubly linked list contents: 11 33

Menu:

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

Enter your choice: 5

Q) WAP to stimulate working of binary search tree through the following functions:

- a) creation**
- b) in-order traversal**
- c) pre-order traversal**
- d) post-order traversal**
- e) display binary search tree elements**

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

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

node* createNode(int data) {
    node* new1 = (node*)malloc(sizeof(node));
    new1->data = data;
    new1->left = new1->right = NULL;
    return new1;
}

node* insertNode(node* root, int data) {
    if (root == NULL) {
        return createNode(data);
    }
    if (data < root->data) {
        root->left = insertNode(root->left, data);
    } else {
        root->right = insertNode(root->right, data);
    }
    return root;
}

void inorderTraversal(node* root) {
    if (root != NULL) {
        inorderTraversal(root->left);
        printf("%d ", root->data);
    }
}
```

```

        inorderTraversal(root->right);
    }
}

void preorderTraversal(node* root) {
    if (root != NULL) {
        printf("%d ", root->data);
        preorderTraversal(root->left);
        preorderTraversal(root->right);
    }
}

void postorderTraversal(node* root) {
    if (root != NULL) {
        postorderTraversal(root->left);
        postorderTraversal(root->right);
        printf("%d ", root->data);
    }
}

int findHeight(node* root) {
    if (root == NULL) {
        return 0;
    }
    int leftHeight = findHeight(root->left);
    int rightHeight = findHeight(root->right);
    return fmax(leftHeight, rightHeight) + 1;
}

void displayTree(node* root) {
    int height = findHeight(root);
    int space = (int)pow(2, height) - 1;
    node* queue[100]; // Queue for level order traversal
    int front = 0, rear = 0;
    queue[rear++] = root;
    while (front < rear) {
        int levelSize = rear - front; // Number of nodes at current level
        for (int i = 0; i < levelSize; i++) {
            for (int j = 0; j < space; j++) {
                printf(" ");
            }
        }
    }
}

```

```

    }
    node* currentNode = queue[front++];
    if (currentNode != NULL) {
        printf("%d", currentNode->data);
        if (currentNode->left != NULL) {
            queue[rear++] = currentNode->left;
        }
        if (currentNode->right != NULL) {
            queue[rear++] = currentNode->right;
        }
    }
    for (int j = 0; j < space * 2; j++) {
        printf(" ");
    }
}
printf("\n\n");
space /= 2;
}
}

```

```

void main() {
    node* root = NULL;
    int choice, value;
    printf("\nMenu:");
    while (1) {
        printf("\n1. Insert\n2. In-order Traversal\n3. Pre-order Traversal\n4. Post-order Traversal\n5. Display\n6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the value to insert: ");
                scanf("%d", &value);
                root = insertNode(root, value);
                break;
            case 2:
                printf("In-order Traversal: ");
                inorderTraversal(root);
                printf("\n");

```

```

        break;
    case 3:
        printf("Pre-order Traversal: ");
        preorderTraversal(root);
        printf("\n");
        break;
    case 4:
        printf("Post-order Traversal: ");
        postorderTraversal(root);
        printf("\n");
        break;
    case 5:
        printf("Visual Tree Representation:\n");
        displayTree(root);
        break;
    case 6:
        exit(0);
    default:
        printf("Invalid choice. Please try again.\n");
    }
}
}

```

Output

```

Menu:
1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Display
6. Exit
Enter your choice: 1
Enter the value to insert: 15

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Display
6. Exit
Enter your choice: 1
Enter the value to insert: 8

```

```
1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Display
6. Exit
Enter your choice: 1
Enter the value to insert: 23
```

```
1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Display
6. Exit
Enter your choice: 1
Enter the value to insert: 6
```

```
1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Display
6. Exit
Enter your choice: 1
Enter the value to insert: 11
```

```
1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Display
6. Exit
Enter your choice: 1
Enter the value to insert: 22
```

```
1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Display
6. Exit
Enter your choice: 1
Enter the value to insert: 66
```

```

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Display
6. Exit
Enter your choice: 2
In-order Traversal: 6 8 11 15 22 23 66

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Display
6. Exit
Enter your choice: 3
Pre-order Traversal: 15 8 6 11 23 22 66

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Display
6. Exit
Enter your choice: 4
Post-order Traversal: 6 11 8 22 66 23 15

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Display
6. Exit
Enter your choice: 5
Visual Tree Representation:
      15
     /  \
    8    23
   / \  / \
  6 11 22 66

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Display
6. Exit
Enter your choice: 6

```


Q) Write a program to traverse a graph using the BFS method.

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

#define MAX 100

int queue[MAX], front = -1, rear = -1;

void enqueue(int item) {
    if (rear == MAX - 1) {
        printf("Queue Overflow\n");
        return;
    }
    if (front == -1) front = 0;
    queue[++rear] = item;
}

int dequeue() {
    if (front == -1 || front > rear) {
        printf("Queue Underflow\n");
        return -1;
    }
    return queue[front++];
}

void bfs(int graph[MAX][MAX], int visited[MAX], int start, int n) {
    int i;
    enqueue(start);
    visited[start] = 1;

    printf("BFS Traversal: ");
    while (front <= rear) {
        int current = dequeue();
        printf("%d ", current);

        for (i = 0; i < n; i++) {
            if (graph[current][i] == 1 && !visited[i]) {
                enqueue(i);
                visited[i] = 1;
            }
        }
    }
}
```

```

    }
}
}
printf("\n");
}

void main() {
    int n, i, j, start;
    int graph[MAX][MAX], visited[MAX] = {0};

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

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

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

    bfs(graph, visited, start, n);
}

```

Output

```

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

```

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

```
#include <stdio.h>

#define MAX 10

int a[MAX][MAX], vis[MAX], n;

void dfsConnected(int v) {
    vis[v] = 1;

    for (int i = 0; i < n; i++) {
        if (a[v][i] == 1 && !vis[i]) {
            dfsConnected(i);
        }
    }
}

int isConnected() {
    for (int i = 0; i < n; i++) {
        vis[i] = 0;
    }
    dfsConnected(0);
    for (int i = 0; i < n; i++) {
        if (!vis[i]) {
            return 0;
        }
    }
    return 1;
}

void dfs(int v) {
    printf("%d ", v+1);
    vis[v] = 1; // Mark the current node as visited
    for (int i = 0; i < n; i++) {
        // If there is an edge from v to i and i is not visited
        if (a[v][i] == 1 && vis[i] == 0) {
```

```

        dfs(i);
    }
}

```

```

void main() {
    int i, j;

    printf("Enter Number of Vertices: ");
    scanf("%d", &n);

    printf("Enter Adjacency Matrix:\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            scanf("%d", &a[i][j]);
        }
    }

    for (i = 0; i < n; i++) {
        vis[i] = 0; // Initialize visited array
    }

    printf("DFS Traversal: ");
    for (i = 0; i < n; i++) {
        if (vis[i] == 0) {
            dfs(i);
        }
    }

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

```

Output

```
C:\Users\Asha\Downloads>dfs
Enter Number of Vertices: 5
Enter Adjacency Matrix:
0 0 1 1 1
0 0 0 1 1
1 0 0 1 0
1 1 1 0 0
1 1 0 0 0
DFS Traversal: 1 3 4 2 5
The graph is connected.
```

Q) 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 → L as $H(K) = K \text{ mod } m$ (remainder method), and implement hashing technique to map a given key K to the address space L. Resolve the collision (if any) using linear probing.

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

#define MAX_EMPLOYEES 100
#define m 100

typedef struct {
    int key;
    int address;
} EmployeeRecord;

int hashTable[m];

int hashFunction(int key) {
    return key % m;
}

int insert(int key) {
    int index = hashFunction(key);
    while (hashTable[index] != -1) {
        index = (index + 1) % m;
    }
    hashTable[index] = key;
    return index;
}

void displayHashTable() {
    printf("\nHash Table:\n");
    printf("Index  Key\n");
    for (int i = 0; i < m; i++) {
        if (hashTable[i] != -1) {
```

```

        printf("%d    %d\n", i, hashTable[i]);
    }
}

int main() {
    for (int i = 0; i < m; i++)
        hashTable[i] = -1;
    int employeeKeys[MAX_EMPLOYEES];
    int numEmployees;
    printf("Enter number of employees: ");
    scanf("%d", &numEmployees);
    printf("Enter the employee keys (4-digit integers):\n");
    for (int i = 0; i < numEmployees; i++) {
        scanf("%d", &employeeKeys[i]);
    }
    for (int i = 0; i < numEmployees; i++) {
        int address = insert(employeeKeys[i]);
        printf("Employee key %d inserted at address %d\n", employeeKeys[i], address);
    }
    displayHashTable();
}

```

Output

```

Enter number of employees: 6
Enter the employee keys (4-digit integers):
1234
1321
1344
1342
1111
2423
Employee key 1234 inserted at address 34
Employee key 1321 inserted at address 21
Employee key 1344 inserted at address 44
Employee key 1342 inserted at address 42
Employee key 1111 inserted at address 11
Employee key 2423 inserted at address 23

Hash Table:
Index  Key
11      1111
21      1321
23      2423
34      1234
42      1342
44      1344

```