

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT On

DATA STRUCTURES (23CS3PCDST)

Submitted by

ABHINAV C (1BM23CS008)

**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 ABHINAV C (**1BM23CS008**), 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 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. Rajeshwari B S
Associate Professor
Department of CSE
BMSCE, Bengaluru

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

Index Sheet

Sl. No.	Experiment Title	Page No.
1	stack operations	5
2	infix to postfix	10
3	linear and circular queue	16
4	insertion in linked list and leetcode	22
5	deletion in linked list	31
6	sort, reverse, concatenate linked lists and stack and queue operations	35
7	doubly linked list	52
8	binary search list	70
9	BFS and DFS	82
10	Hashing	101

Course outcomes:

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

Lab program 1:

Write a program to simulate the working of stack using an array with the following:

- a) Push
- b) Pop
- c) Display

The program should print appropriate messages for stack overflow, stack underflow.

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

```
#define MAX 5
```

```
int stack[MAX];
```

```
int top = -1;
```

```
void push(int value) {
```

```
    if (top == MAX - 1) {
```

```
        printf("Stack Overflow! Cannot push %d\n", value);
```

```
    } else {
```

```
        top++;
```

```
        stack[top] = value;
```

```
        printf("%d pushed into the stack\n", value);
```

```
    }
```

```
}
```

```
void pop() {
```

```
    if (top == -1) {
```

```
        printf("Stack Underflow! Cannot pop from an empty stack\n");
```

```
    } else {
```

```
        printf("%d popped from the stack\n", stack[top]);
```

```
        top--;  
    }  
}
```

```
void display() {  
    if (top == -1) {  
        printf("Stack is empty\n");  
    } else {  
        printf("Stack elements are:\n");  
        for (int i = top; i >= 0; i--) {  
            printf("%d\n", stack[i]);  
        }  
    }  
}
```

```
int main() {  
    int choice, value;  
  
    do {  
  
        printf("\nStack Operations Menu:\n");  
        printf("1. Push\n");  
        printf("2. Pop\n");  
        printf("3. Display\n");  
        printf("4. Exit\n");  
        printf("Enter your choice: ");  
        scanf("%d", &choice);
```

```
switch(choice) {  
    case 1:  
        printf("Enter the value to push: ");  
        scanf("%d", &value);  
        push(value);  
        break;  
  
    case 2:  
        pop();  
        break;  
  
    case 3:  
        display();  
        break;  
  
    case 4:  
        printf("Exiting the program.\n");  
        break;  
  
    default:  
        printf("Invalid choice! Please try again.\n");  
}  
} while (choice != 4);  
  
return 0;  
}
```

```
Stack Operations Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the value to push: 10
10 pushed into the stack
```

```
Stack Operations Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the value to push: 20
20 pushed into the stack
```

```
Stack Operations Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter the value to push: 30
30 pushed into the stack
```

```
Stack Operations Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2
30 popped from the stack
```

```
Stack Operations Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2
20 popped from the stack
```


Stack Operations Menu:

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

Enter your choice: 2

Stack Underflow! Cannot pop from an empty stack

Lab program 2:

Write a program to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + (plus), - (minus), * (multiply) and / (divide)

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

int index = 0, pos = 0, top = -1, length;
char symbol, temp, infix[30], postfix[30], stack[30];

void infixToPostfix();
void push(char symbol);
char pop();
int precedence(char symb);

int main() {
    printf("Enter infix expression:\n");
    scanf("%s", infix);

    infixToPostfix();
    printf("\nInfix expression:\n%s", infix);
    printf("\nPostfix expression:\n%s", postfix);
    return 0;
}

void infixToPostfix() {
    length = strlen(infix);
    push('#');
```

```

while (index < length) {
    symbol = infix[index];
    switch (symbol) {
        case '(':
            push(symbol);
            break;

        case ')':
            temp = pop();
            while (temp != '(') {
                postfix[pos++] = temp;
                temp = pop();
            }
            break;

        case '+':
        case '-':
        case '*':
        case '/':
        case '^':
            while (precedence(stack[top]) >= precedence(symbol)) {
                temp = pop();
                postfix[pos++] = temp;
            }
            push(symbol);
            break;
        default:
            postfix[pos++] = symbol;
    }
    index++;
}

```

```
}
```

```
while (top > 0) {
```

```
    temp = pop();
```

```
    postfix[pos++] = temp;
```

```
}
```

```
postfix[pos] = '\0';
```

```
}
```

```
void push(char symbol) {
```

```
    top = top + 1;
```

```
    stack[top] = symbol;
```

```
}
```

```
char pop() {
```

```
    char symb = stack[top];
```

```
    top--;
```

```
    return symb;
```

```
}
```

```
int precedence(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;
}
```

```
Enter infix expression:
a^bc-d+e/f/(g+h)

Infix expression:
a^bc-d+e/f/(g+h)
Postfix expression:
abc^d-ef/gh+/+
Process returned 0 (0x0)   execution time : 73.563 s
Press any key to continue.
```

Leet code :

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

```
int firstUniqChar(char* s) {  
    int freq[26]={0};  
    for(int i=0; s[i]!='\0'; i++){  
        freq[s[i]-'a']++;  
    }  
    for(int i=0; s[i]!='\0'; i++){  
        if(freq[s[i]-'a']==1){  
            return i;  
        }  
    }  
    return -1;  
}
```

Input

```
s =  
"leetcode"
```

Output

```
0
```

Expected

```
0
```

Accepted Runtime: 0 ms

- Case 1
- Case 2
- Case 3

Input

s =
"loveleetcode"

Output

2

Expected

2

Accepted Runtime: 0 ms

- Case 1
- Case 2
- Case 3

Input

s =
"aabb"

Output

-1

Expected

-1

Lab program 3:

linear queue insertion and deletion

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

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

```
#define SIZE 5
```

```
int queue[SIZE];
```

```
int front = -1;
```

```
int rear = -1;
```

```
void enter (int value) {
```

```
    if ((front == 0 && rear == SIZE - 1) || (rear == (front - 1) % (SIZE - 1))) {
```

```
        printf("Queue is Full\n");
```

```
        return;
```

```
    }
```

```
    else if (front == -1) {
```

```
        front = rear = 0;
```

```
        queue[rear] = value;
```

```
    }
```

```
    else if (rear == SIZE - 1 && front != 0)
```

```
    {
```

```
        rear = 0;
```

```
        queue[rear] = value;
```

```
    }
```

```
    else {
```

```
        rear++;
```

```
        queue[rear] = value;
```

```
    }
```



```
    printf("Inserted %d\n", value);  
}
```

```
void del() {  
    if (front == -1) {  
        printf("Queue is Empty\n");  
        return;  
    }  
}
```

```
    printf("Deleted %d\n", queue[front]);  
    queue[front] = -1;  
    if (front == rear) {  
        front = rear = -1;  
    }  
    else if (front == SIZE - 1) {  
        front = 0;  
    }  
    else {  
        front++;  
    }  
}
```

```
void display() {  
    if (front == -1) {  
        printf("Queue is Empty\n");  
        return;  
    }  
    printf("Queue elements are: ");  
    if (rear >= front) {  
        for (int i = front; i <= rear; i++)
```

```

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

int main() {
    int choice, value;
    while (1) {
        printf("\n1. Insert\n2. Delete\n3. Display\n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter the value to insert: ");
                scanf("%d", &value);
                enter(value);
                break;
            case 2:
                del();
                break;
            case 3:
                display();
                break;
            case 4:

```

```
        exit(0);  
    default:  
        printf("Invalid choice\n");  
    }  
}  
return 0;  
}
```

```
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to insert: 1
Inserted 1
```

```
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to insert: 2
Inserted 2
```

```
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to insert: 3
Inserted 3
```

```
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to insert: 4
Inserted 4
```

```
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to insert: 5
Inserted 5
```

```
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to insert: 6
Queue is Full
```

```
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
Queue elements are: 1 2 3 4 5
```

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

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

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

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

```
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Deleted 5
```

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

LAB PROGRAM 4:

Circular Queue implementation

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

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

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

```
#define MAX 4
```

```
void Insert();
```

```
int Delete();
```

```
void Display();
```

```
int cq[20];
```

```
int front=-1, rear=-1, item, ch, i;
```

```
void main()
```

```
{
```

```
    while(1)
```

```
    {
```

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

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

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

```
        switch(ch)
```

```
        {
```

```
            case 1: Insert();
```

```
                break;
```

```
            case 2: item=Delete();
```

```

        if (item!=-1)
        {
            printf("The Dequeued Element is: %d",item);
        }
        break;
    case 3: Display();
        break;
    case 4: exit(0);
}
}
}

```

```

void Insert()
{
    if (front == (rear+1) % MAX)
    {
        printf("Circular Queue is Full. \n");
        return;
    }
    if (rear==-1 && front==-1)
    {
        rear=0;
        front=0;
    }
    else
        rear=(rear+1)%MAX;
    printf("Enter the Element to be Inserted: ");
}

```

```
scanf("%d",&item);  
cq[rear]=item;  
return;  
}
```

```
int Delete()  
{  
    if(front==-1 && rear==-1)  
    {  
        printf("Circular Queue is Empty. \n");  
        return (-1);  
    }  
    item=cq[front];  
    if(front==rear)  
    {  
        front=-1;  
        rear=-1;  
    }  
    else  
        front=(front+1)%MAX;  
    return item;  
}
```

```
void Display()  
{  
    if(front==-1 && rear==-1)  
    {
```



```

    printf("Circular Queue is Empty. \n");
    return;
}

printf("Circular Queue Contents: \n");
if (front<=rear)
{
    for (int i=front;i<=rear;i++)
    {
        printf("%d\n",cq[i]);
    }
}

else
{
    for(int i=front;i<=MAX-1;i++)
    {
        printf("%d\n",cq[i]);
    }
    for (int i=0;i<=rear;i++)
    {
        printf("%d\n",cq[i]);
    }
}
return;
}

```

```
1. Insert
2. Delete
3. Display
4. Exit
Enter Your Choice: 1
Enter the Element to be Inserted: 10
```

```
1. Insert
2. Delete
3. Display
4. Exit
Enter Your Choice: 1
Enter the Element to be Inserted: 20
```

```
1. Insert
2. Delete
3. Display
4. Exit
Enter Your Choice: 1
Enter the Element to be Inserted: 30
```

```
1. Insert
2. Delete
3. Display
4. Exit
Enter Your Choice: 1
Enter the Element to be Inserted: 40
```

```
1. Insert
2. Delete
3. Display
4. Exit
Enter Your Choice: 1
Circular Queue is Full.
```

```
1. Insert
2. Delete
3. Display
4. Exit
Enter Your Choice: 3
Circular Queue Contents:
10
20
30
40
```

```
1. Insert
2. Delete
3. Display
4. Exit
Enter Your Choice: 2
The Dequeued Element is: 10
1. Insert
2. Delete
3. Display
4. Exit
Enter Your Choice: 2
The Dequeued Element is: 20
1. Insert
2. Delete
3. Display
4. Exit
Enter Your Choice: 2
The Dequeued Element is: 30
1. Insert
2. Delete
3. Display
4. Exit
Enter Your Choice: 2
The Dequeued Element is: 40
1. Insert
2. Delete
3. Display
4. Exit
Enter Your Choice: 2
Circular Queue is Empty.

1. Insert
2. Delete
3. Display
4. Exit
Enter Your Choice: 4

Process returned 0 (0x0)   execution time : 43.123 s
Press any key to continue.
|
```

LAB 4 LEET CODE:

For a stream of integers, implement a data structure that checks if the last k integers parsed in the stream are equal to value.

Implement the DataStream class:

DataStream(int value, int k) Initializes the object with an empty integer stream and the two integers value and k.

boolean consec(int num) Adds num to the stream of integers. Returns true if the last k integers are equal to value, and false otherwise. If there are less than k integers, the condition does not hold true, so returns false.

```
class DataStream {
public:
    int val;
    int k;
    int cnt=0;

    DataStream(int value, int K) {
        val=value;
        k=K;
    }

    bool consec(int num) {
        if(num==val)
        {
            cnt++;
            if(cnt>=k)
                return true;
            return false;
        }
    }
};
```

```
    }  
    cnt=0;  
    return false;  
}  
};
```

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

s =
"leetcode"

Output

0

Expected

0

♥ Contribute a testcase

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

s =
"loveleetcode"

Output

2

Expected

2

♥ Contribute a testcase

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

```
s =  
"aabb"
```

Output

```
-1
```

Expected

```
-1
```

♥ [Contribute a testcase](#)

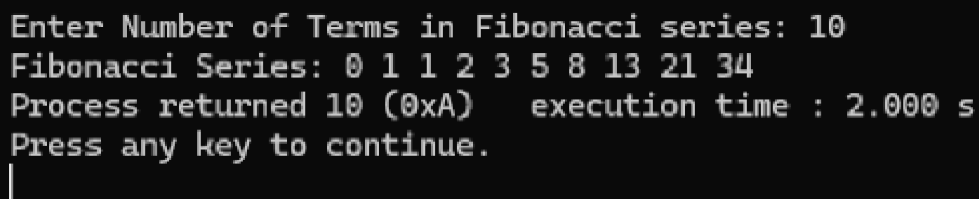
LAB PROGRAM 5:

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

int fibonacci(int n) {
    if (n <= 1) {
        return n;
    }
    return fibonacci(n - 1) + fibonacci(n - 2);
}

void main() {
    int n, i;
    printf("Enter Number of Terms in Fibonacci series: ");
    scanf("%d", &n);

    printf("Fibonacci Series: ");
    for (i = 0; i < n; i++) {
        printf("%d ", fibonacci(i));
    }
}
```

A screenshot of a terminal window with a black background and white text. It shows the output of the Fibonacci program: 'Enter Number of Terms in Fibonacci series: 10', 'Fibonacci Series: 0 1 1 2 3 5 8 13 21 34', 'Process returned 10 (0xA) execution time : 2.000 s', and 'Press any key to continue.' followed by a vertical cursor line.

```
Enter Number of Terms in Fibonacci series: 10
Fibonacci Series: 0 1 1 2 3 5 8 13 21 34
Process returned 10 (0xA) execution time : 2.000 s
Press any key to continue.
|
```

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

```
int factorial(int n)
{
    if (n<=1)
    {
        return 1;
    }
    return n * factorial(n - 1);
}
```

```
void main()
{
    int num;

    printf("Enter Number to Calculate Factorial: ");
    scanf("%d", &num);

    if (num < 0)
    {
        printf("Factorial Not Possible\n");
    }
    else
    {
        printf("Factorial of %d is %d\n", num, factorial(num));
    }
}
```



```
Enter Number to Calculate Factorial: 6
Factorial of 6 is 720
```

```
//Tower of Hanoi
#include <stdio.h>

void TOH(int n, char s, char t, char d)
{
    if (n == 1)
    {
        printf("Move Disk %d from %c to %c\n", n, s, d);
        return;
    }
    TOH(n - 1, s, d, t);
    printf("Move disk %d from %c to %c\n", n, s, d);
    TOH(n - 1, t, s, d);
}

void main()
{
    int n = 3;
    TOH(n, 'S', 'T', 'D');
}
```

```
Move Disk 1 from S to D
Move disk 2 from S to T
Move Disk 1 from D to T
Move disk 3 from S to D
Move Disk 1 from T to S
Move disk 2 from T to D
Move Disk 1 from S to D
```

LAB PROGRAM 6:

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);
```

```
}
```

```
}
```

```
}
```

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

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

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

```

```
    ptr = ptr->link;
}
printf("\n");
}
```

```
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 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 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;
}
```


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

Enter Your Choice: 1

Enter Value: 10

Do You Want to Add an Element (Y/N)? y

Enter Value: 20

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:

10 20

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

Enter Your Choice: 3

Enter Value: 30

Enter Value: 30

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

Enter Your Choice: 4

Enter Value: 40

Enter Position: 2

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

Enter Your Choice: 5

Enter Value: 50

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:

30 40 10 20 50

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

Enter Your Choice: 6

2) 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;  
node *new1, *curr, *ptr;
```

```
void create();  
void display();  
void DeleteStart();  
void DeletePosition();  
void DeleteEnd();
```

```
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: DeleteStart();
            break;
        case 4: DeletePosition();
            break;
        case 5: DeleteEnd();
            break;
        case 6: exit(0);
    }
}
}

```

```

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

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

    while (ptr != NULL) {

```

```
    printf("%d ", ptr->data);  
    ptr = ptr->link;  
}  
printf("\n");  
}
```

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

```
  
    node *temp = start;  
    start = start->link;  
    free(temp);  
    printf("\nFirst Element Deleted.\n");  
}
```

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

```
  
    printf("\nEnter Position: ");  
    scanf("%d", &pos);
```

```
node *temp = start;
```

```
node *prev = NULL;
```

```
if (pos == 1) {
```

```
    start = temp->link;
```

```
    free(temp);
```

```
    printf("\nElement at Position %d Deleted.\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\n", pos);
```

```
}
```

```
void DeleteEnd() {  
    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.\n");  
        return;  
    }
```

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

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



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

Enter Value: 10
Do You Want to Add an Element (Y/N)? y

Enter Value: 20
Do You Want to Add an Element (Y/N)? y

Enter Value: 30
Do You Want to Add an Element (Y/N)? y

Enter Value: 40
Do You Want to Add an Element (Y/N)? y

Enter Value: 50
Do You Want to Add an Element (Y/N)? y

Enter Value: 60
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:
10 20 30 40 50 60
```

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.

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:

20 30 40 50 60

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

Enter Your Choice: 4

Enter Position: 3

Element at Position 3 Deleted

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:
20 30 50 60

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.

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:
20 30 50

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

Enter Your Choice: 6

Process returned 0 (0x0) execution time : 51.985 s
Press any key to continue.

LAB RECORD 7:

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

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

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

```
struct Node {
```

```
    int data;
```

```
    struct Node *link;
```

```
};
```

```
typedef struct Node node;
```

```
node *start = NULL, *temp, *new1, *curr;
```

```
int ch;
```

```
char c;
```

```
void createList();
```

```
void sort();
```

```
void reverse();
```

```
void display();
```

```
void concatenate();
```

```
void createList() {
```

```
    do {
```

```
        new1 = (node*)malloc(sizeof(node));
```

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

```
        scanf("%d", &new1->data);
```

```

new1->link = NULL;

if (start == NULL) {
    start = new1;
    curr = new1;
} else {
    curr->link = new1;
    curr = new1;
}

printf("Do you want to add another element (Y/N): ");
scanf(" %c", &c);
} while (c == 'y' || c == 'Y');
}

```

```

void sort() {
    if (start == NULL) {
        printf("The Linked List is Empty.\n");
        return;
    }
}

```

```

node *i, *j;
int tempData;
for (i = start; i != NULL; i = i->link) {
    for (j = i->link; j != NULL; j = j->link) {
        if (i->data > j->data) {
            tempData = i->data;
            i->data = j->data;

```

```
        j->data = tempData;
    }
}
}
printf("Linked List is Sorted.\n");
}
```

```
void reverse() {
    node *a = start, *b = NULL;
    while (a != NULL) {
        temp = a->link;
        a->link = b;
        b = a;
        a = temp;
    }
    start = b;
    printf("Linked List is Reversed.\n");
}
```

```
void display() {
    if (start == NULL) {
        printf("Linked list is Empty\n");
        return;
    }

    temp = start;
    printf("Elements in Linked List:\n");
```

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

```
void concatenate() {  
    node *start2 = NULL, *curr2 = NULL;  
  
    printf("Enter the second linked list:\n");  
    createList();  
  
    do {  
        new1 = (node*)malloc(sizeof(node));  
        printf("Enter value for second list: ");  
        scanf("%d", &new1->data);  
        new1->link = NULL;  
  
        if (start2 == NULL) {  
            start2 = new1;  
            curr2 = new1;  
        } else {  
            curr2->link = new1;  
            curr2 = new1;  
        }  
        printf("Do you want to add another element (Y/N): ");
```

```

        scanf(" %c", &c);
    } while (c == 'y' || c == 'Y');

    if (start == NULL) {
        start = start2;
    } else {
        temp = start;
        while (temp->link != NULL) {
            temp = temp->link;
        }
        temp->link = start2;
    }
    start2 = NULL;
    printf("Lists concatenated successfully.\n");
}

int main() {
    while (1) {
        printf("\n1. Create 1st Linked List\n2. Sort Linked List\n3. Reverse Linked List\n4.
Concatenate Linked Lists\n5. Display Linked List\n6. Exit\n");

        printf("Enter Your Choice: ");
        scanf("%d", &ch);
        switch (ch) {
            case 1:
                createList();
                break;
            case 2:
                sort();

```



```
        break;
    case 3:
        reverse();
        break;
    case 4:
        concatenate();
        break;
    case 5:
        display();
        break;
    case 6:
        exit(0);
        break;
    default:
        printf("Invalid choice. Please try again.\n");
        break;
}
}
}
```

```
1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 1
```

Enter Value to Push: 10

```
1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 1
```

Enter Value to Push: 20

```
1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 1
```

Enter Value to Push: 30

```
1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 1
```

Enter Value to Push: 40

```
1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 4
```

```
Enter Value to Insert: 10
```

```
1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 4
```

```
Enter Value to Insert: 20
```

```
1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 4
```

```
Enter Value to Insert: 30
```

```
1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 4
```

```
Enter Value to Insert: 40
```

1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit

Enter Your Choice: 6

Elements in the Queue: 10 20 30 40

1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit

Enter Your Choice: 5

Deleted Element: 10

1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit

Enter Your Choice: 5

Deleted Element: 20

1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit

Enter Your Choice: 5

Deleted Element: 30

```

1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 6

Elements in the Queue: 40

1. Push (Stack)
2. Pop (Stack)
3. Display (Stack)
4. Insert (Queue)
5. Delete (Queue)
6. Display (Queue)
7. Exit
Enter Your Choice: 7

```

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

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

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

```

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

```

```
typedef struct Node node;
```

```
//Stack
```

```
node *top=NULL;
```

```
void push();
```

```
void pop();
```

```
void displayStack();
```

```
void push(){
```

```
    node *new1=(node*)malloc(sizeof(node));
```

```
    if(new1==NULL){
```

```
        printf("\nStack Overflow.\n");
```

```
        return;
```

```
    }
```

```
    printf("\nEnter Value to Push: ");
```

```
    scanf("%d", &new1->data);
```

```
    new1->link=top;
```

```
    top=new1;
```

```
}
```

```
void pop(){
```

```
    if(top==NULL){
```

```
        printf("\nStack Underflow.\n");
```

```
        return;
```

```
    }
```

```
    node *temp=top;
```

```
    printf("\nPopped Element: %d\n", temp->data);
```

```
    top=top->link;
```

```
    free(temp);
```

```
}
```

```
void displayStack(){  
    if(top==NULL){  
        printf("\nThe Stack is Empty.\n");  
        return;  
    }
```

```
  
    printf("\nElements in the Stack: ");  
    node *temp=top;  
    while(temp!=NULL){  
        printf("%d ", temp->data);  
        temp=temp->link;  
    }  
    printf("\n");  
}
```

//Queue

```
node *front=NULL, *rear=NULL;
```

```
void insert();
```

```
void del();
```

```
void displayQueue();
```

```
void insert(){  
    node *new1=(node*)malloc(sizeof(node));  
    if(new1==NULL){
```

```
    printf("\nQueue Full.\n");  
    return;  
}
```

```
printf("\nEnter Value to Insert: ");  
scanf("%d", &new1->data);  
new1->link=NULL;
```

```
if(rear==NULL){  
    front=rear=new1;  
    return;  
}  
rear->link=new1;  
rear=new1;  
}
```

```
void del(){  
    if(front==NULL){  
        printf("\nQueue Empty.\n");  
        return;  
    }
```

```
    node *temp=front;  
    printf("\nDeleted Element: %d\n", temp->data);  
    front=front->link;  
  
    if(front==NULL){
```



```
        rear=NULL;
    }
    free(temp);
}
```

```
void displayQueue(){
    if(front==NULL){
        printf("\nThe Queue is Empty.\n");
        return;
    }
```

```
    printf("\nElements in the Queue: ");
    node *temp=front;
    while(temp!=NULL){
        printf("%d ", temp->data);
        temp=temp->link;
    }
    printf("\n");
}
```

```
// Main
```

```
void main(){
    int ch;

    while(1){
        printf("\n1. Push (Stack) \n2. Pop (Stack) \n3. Display (Stack)");
```

```
printf("\n4. Insert (Queue) \n5. Delete (Queue) \n6. Display (Queue) \n7. Exit");
```

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

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

```
switch(ch){
```

```
    case 1:
```

```
        push();
```

```
        break;
```

```
    case 2:
```

```
        pop();
```

```
        break;
```

```
    case 3:
```

```
        displayStack();
```

```
        break;
```

```
    case 4:
```

```
        insert();
```

```
        break;
```

```
    case 5:
```

```
        del();
```

```
        break;
```

```
    case 6:
```

```
        displayQueue();
```

```
        break;
```

```
    case 7:
```

```
        exit(0);
```

```
    default:
```

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

```
}  
}
```

```
1. Create 1st Linked List  
2. Sort Linked List  
3. Reverse Linked List  
4. Concatenate Linked Lists  
5. Display Linked List  
6. Exit  
Enter Your Choice: 1  
Enter Value: 10  
Do you want to add another element (Y/N): y  
Enter Value: 80  
Do you want to add another element (Y/N): y  
Enter Value: 60  
Do you want to add another element (Y/N): y  
Enter Value: 20  
Do you want to add another element (Y/N): y  
Enter Value: 70  
Do you want to add another element (Y/N): y  
Enter Value: 30  
Do you want to add another element (Y/N): n
```

```
1. Create 1st Linked List  
2. Sort Linked List  
3. Reverse Linked List  
4. Concatenate Linked Lists  
5. Display Linked List  
6. Exit  
Enter Your Choice: 5  
Elements in Linked List:  
10      80      60      20      70      30
```

```
1. Create 1st Linked List  
2. Sort Linked List  
3. Reverse Linked List  
4. Concatenate Linked Lists  
5. Display Linked List  
6. Exit  
Enter Your Choice: 3  
Linked List is Reversed.
```

```
1. Create 1st Linked List  
2. Sort Linked List  
3. Reverse Linked List  
4. Concatenate Linked Lists  
5. Display Linked List  
6. Exit  
Enter Your Choice: 5  
Elements in Linked List:  
30      70      20      60      80      10
```

```
1. Create 1st Linked List
2. Sort Linked List
3. Reverse Linked List
4. Concatenate Linked Lists
5. Display Linked List
6. Exit
Enter Your Choice: 2
Linked List is Sorted.

1. Create 1st Linked List
2. Sort Linked List
3. Reverse Linked List
4. Concatenate Linked Lists
5. Display Linked List
6. Exit
Enter Your Choice: 5
Elements in Linked List:
10      20      30      60      70      80

1. Create 1st Linked List
2. Sort Linked List
3. Reverse Linked List
4. Concatenate Linked Lists
5. Display Linked List
6. Exit
Enter Your Choice: 4
Enter the second linked list:
Enter Value: 10
Do you want to add another element (Y/N): y
Enter Value: 70
Do you want to add another element (Y/N): y
Enter Value: 80
Do you want to add another element (Y/N): y
Enter Value: 60
Do you want to add another element (Y/N): y
Enter Value: 30
Do you want to add another element (Y/N): n
Enter value for second list: 10
Do you want to add another element (Y/N): y
Enter value for second list: 50
Do you want to add another element (Y/N): y
Enter value for second list: 60
Do you want to add another element (Y/N): y
Enter value for second list: 40
Do you want to add another element (Y/N): n
Lists concatenated successfully.
```

```

1. Create 1st Linked List
2. Sort Linked List
3. Reverse Linked List
4. Concatenate Linked Lists
5. Display Linked List
6. Exit
Enter Your Choice: 5
Elements in Linked List:
10      10      70      80      60      30      10      50      60      40

1. Create 1st Linked List
2. Sort Linked List
3. Reverse Linked List
4. Concatenate Linked Lists
5. Display Linked List
6. Exit
Enter Your Choice: 2
Linked List is Sorted.

1. Create 1st Linked List
2. Sort Linked List
3. Reverse Linked List
4. Concatenate Linked Lists
5. Display Linked List
6. Exit
Enter Your Choice: 5
Elements in Linked List:
10      10      10      30      40      50      60      60      70      80

1. Create 1st Linked List
2. Sort Linked List
3. Reverse Linked List
4. Concatenate Linked Lists
5. Display Linked List
6. Exit
Enter Your Choice: 3
Linked List is Reversed.

1. Create 1st Linked List
2. Sort Linked List
3. Reverse Linked List
4. Concatenate Linked Lists
5. Display Linked List
6. Exit
Enter Your Choice: 5
Elements in Linked List:
80      70      60      60      50      40      30      10      10      10

```

LAB PROGRAM 8:

```
#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 DeleteStart();
```

```
void DeletePosition();
```

```
void DeleteEnd();
```

```
void main() {
```

```
    int ch;
```

```
    while (1) {
```

```
        printf("\n1. Create \n2. Display \n3. Insert at Beginning \n4. Insert at Position \n5. Insert at  
End \n6. Delete from Beginning \n7. Delete at Position \n8. Delete at End \n9. 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: DeleteStart();
```

```
        break;
```

```
    case 7: DeletePosition();
```

```
        break;
```

```
    case 8: DeleteEnd();
```

```
        break;
```

```
    case 9: exit(0);
```

```
}
```

```
}
```

```
}
```

```
void create() {
```

```
    char ch;
```

```
    do {
```

```

new1 = (node*)malloc(sizeof(node));
printf("\nEnter Value: ");
scanf("%d", &new1->data);
if (start == NULL) {
    start = new1;
    new1->link = start;
}
else {
    curr = start;
    while (curr->link != start) {
        curr = curr->link;
    }
    curr->link = new1;
    new1->link = start;
}
printf("Do You Want to Add an Element (Y/N)? ");
scanf(" %c", &ch);
} while (ch == 'y' || ch == 'Y');
}

```

```

void display() {
    if (start == NULL) {
        printf("\nLinked List is Empty.");
        return;
    }
    ptr = start;
    printf("\nElements in Circular Linked List: \n");

```



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

```
void InsertStart() {  
    new1 = (node*)malloc(sizeof(node));  
    printf("\nEnter Value: ");  
    scanf("%d", &new1->data);  
  
    if (start == NULL) {  
        start = new1;  
        new1->link = start;  
    }  
    else {  
        new1->link = start;  
        start = new1;  
        ptr = start;  
        while (ptr->link != start) {  
            ptr = ptr->link;  
        }  
        ptr->link = start;  
    }  
}
```

```

void InsertEnd() {
    new1 = (node*)malloc(sizeof(node));
    printf("\nEnter Value: ");
    scanf("%d", &new1->data);

    if (start == NULL) {
        start = new1;
        new1->link = start;
    }
    else {
        curr = start;
        while (curr->link != start) {
            curr = curr->link;
        }
        curr->link = new1;
        new1->link = start;
    }
}

```

```

void InsertPosition() {
    int i = 1, pos;
    new1 = (node*)malloc(sizeof(node));
    printf("\nEnter Value: ");
    scanf("%d", &new1->data);

    if (start == NULL) {
        start = new1;

```

```
    new1->link = start;
    return;
}
```

```
printf("\nEnter Position: ");
scanf("%d", &pos);
if (pos == 1) {
    new1->link = start;
    start = new1;
    ptr = start;
    while (ptr->link != start) {
        ptr = ptr->link;
    }
    ptr->link = start;
    return;
}
```

```
ptr = start;
while (ptr->link != start && i < pos - 1) {
    ptr = ptr->link;
    i++;
}
```

```
if (i == pos - 1) {
    new1->link = ptr->link;
    ptr->link = new1;
} else {
```

```
        printf("\nPosition Not Found.\n");
    }
}
```

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

```
    node *temp = start;
    if (start->link == start) {
        start = NULL;
    } else {
        ptr = start;
        while (ptr->link != start) {
            ptr = ptr->link;
        }
        start = start->link;
        ptr->link = start;
    }
    free(temp);
    printf("\nFirst Element Deleted.\n");
}
```

```
void DeletePosition() {
    int i = 1, pos;
```

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

```
printf("\nEnter Position: ");  
scanf("%d", &pos);
```

```
if (pos == 1) {  
    DeleteStart();  
    return;  
}
```

```
ptr = start;  
node *prev = NULL;  
while (ptr != start && i < pos) {  
    prev = ptr;  
    ptr = ptr->link;  
    i++;  
}
```

```
if (ptr == start) {  
    printf("\nPosition Not Found.\n");  
    return;  
}
```

```
prev->link = ptr->link;
```

```
    free(ptr);  
    printf("\nElement at Position %d Deleted\n", pos);  
}
```

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

```
  
    node *temp = start;  
    if (start->link == start) {  
        start = NULL;  
    }  
    else {  
        ptr = start;  
        while (ptr->link != start) {  
            ptr = ptr->link;  
        }  
        ptr->link = start;  
    }  
    free(temp);  
    printf("\nLast Element Deleted.\n");  
}
```

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

Enter Value: 10
Do You Want to Add an Element (Y/N)? y

Enter Value: 20
Do You Want to Add an Element (Y/N)? y

Enter Value: 30
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. Delete from Beginning
7. Delete at Position
8. Delete at End
9. Exit
Enter Your Choice: 4

Enter Value: 40

Enter Position: 2

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

Elements in Circular Linked List:
10 40 20 30
```

1. Create
2. Display
3. Insert at Beginning
4. Insert at Position
5. Insert at End
6. Delete from Beginning
7. Delete at Position
8. Delete at End
9. Exit

Enter Your Choice: 5

Enter Value: 50

1. Create
2. Display
3. Insert at Beginning
4. Insert at Position
5. Insert at End
6. Delete from Beginning
7. Delete at Position
8. Delete at End
9. Exit

Enter Your Choice: 2

Elements in Circular Linked List:

10 40 20 30 50

1. Create
2. Display
3. Insert at Beginning
4. Insert at Position
5. Insert at End
6. Delete from Beginning
7. Delete at Position
8. Delete at End
9. Exit

Enter Your Choice: 6

First Element Deleted.

1. Create
2. Display
3. Insert at Beginning
4. Insert at Position
5. Insert at End
6. Delete from Beginning
7. Delete at Position
8. Delete at End
9. Exit

First Element Deleted.

1. Create
2. Display
3. Insert at Beginning
4. Insert at Position
5. Insert at End
6. Delete from Beginning
7. Delete at Position
8. Delete at End
9. Exit

Enter Your Choice: 8

Last Element Deleted.

LAB PROGRAM 9:

BINARY TREE AND DOUBLY LINKED LIST

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

```
#include <stdlib.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);  
    }  
}
```

```
void displayTree(node* root, int space) {
```

```

if (root == NULL) {
    return;
}

space += 10;

displayTree(root->right, space);

printf("\n");
for (int i = 10; i < space; i++) {
    printf(" ");
}
printf("%d\n", root->data);

displayTree(root->left, space);
}

int main() {
    node* root = NULL;
    int choice, value;

    printf("Binary Search Tree Operations:\n");
    while (1) {
        printf("\n1. Insert\n2. In-order Traversal\n3. Pre-order Traversal\n4. Post-order Traversal\n5. Display Tree\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("Tree Representation:\n");  
        displayTree(root, 0);  
        printf("\n");  
        break;  
    case 6:
```

```

        exit(0);

    default:

        printf("Invalid choice. Please try again.\n");

    }

}

return 0;

}

```

Binary Search Tree Operations:

```

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 1
Enter the value to insert: 50

```

```

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 1
Enter the value to insert: 40

```

```

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 1
Enter the value to insert: 75

```

```

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 1
Enter the value to insert: 10

```

```

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 1
Enter the value to insert: 25

```

```

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 1
Enter the value to insert: 80

```

```

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 1
Enter the value to insert: 20

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 2
In-order Traversal: 10 20 25 40 50 75 80

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 3
Pre-order Traversal: 50 40 10 25 20 75 80

1. Insert
2. In-order Traversal
3. Pre-order Traversal
4. Post-order Traversal
5. Exit
Enter your choice: 4
Post-order Traversal: 20 25 10 40 80 75 50

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

```

```

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

          80
        75
      50
    40
      25
        20
          10

```

2)DLL

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

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

```
struct Node {  
    int data;  
    struct Node *left;  
    struct Node *right;  
};
```

```
typedef struct Node node;
```

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

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

```
void create();
```

```
void display();
```

```
void InsertLeft();
```

```
void DeleteSpecificElement();
```

```
void main() {
```

```
    int ch;
```

```
    while (1) {
```

```
        printf("\n1. Create \n2. Display \n3. Insert Left \n4. Delete Specific Element \n5. Exit");
```

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

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



```
switch (ch) {  
    case 1: create();  
        break;  
    case 2: display();  
        break;  
    case 3: InsertLeft();  
        break;  
    case 4: DeleteSpecificElement();  
        break;  
    case 5: exit(0);  
}  
}  
}
```

```
void create() {  
    char ch;  
  
    do {  
        new1 = (node*)malloc(sizeof(node));  
        printf("\nEnter Value: ");  
        scanf("%d", &new1->data);  
        new1->left = NULL;  
        new1->right = NULL;  
  
        if (start == NULL) {  
            start = new1;  
            curr = new1;  
        }  
    } while (ch != 'q');
```

```

    } else {
        curr->right = new1;
        new1->left = curr;
        curr = new1;
    }

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

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

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

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

```

```

void InsertLeft() {
    int val;

    printf("\nEnter Value: ");
    scanf("%d", &val);

    new1 = (node*)malloc(sizeof(node));
    new1->data = val;
    new1->left = NULL;
    new1->right = NULL;

    printf("\nEnter the Value to Insert Left of: ");
    scanf("%d", &val);

    ptr = start;
    while (ptr != NULL && ptr->data != val) {
        ptr = ptr->right;
    }

    if (ptr != NULL) {
        new1->right = ptr;
        new1->left = ptr->left;
        if (ptr->left != NULL) {
            ptr->left->right = new1;
        }
        ptr->left = new1;
        if (ptr == start) {
            start = new1;
        }
    }
}

```

```

    }
} else {
    printf("\nValue not found.\n");
}
}

```

```

void DeleteSpecificElement() {
    int value;

    printf("\nEnter Value to Delete: ");
    scanf("%d", &value);

    ptr = start;
    while (ptr != NULL && ptr->data != value) {
        ptr = ptr->right;
    }

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

    if (ptr->left != NULL) {
        ptr->left->right = ptr->right;
    }

    if (ptr->right != NULL) {
        ptr->right->left = ptr->left;
    }
}

```

```

    if (ptr == start) {
        start = ptr->right;
    }

    free(ptr);

    printf("\nElement with value %d deleted.\n", value);
}

```

```

1. Create
2. Display
3. Insert Left
4. Delete Specific Element
5. Exit
Enter Your Choice: 1

Enter Value: 10
Do You Want to Add an Element (Y/N)? y

Enter Value: 20
Do You Want to Add an Element (Y/N)? y

Enter Value: 30
Do You Want to Add an Element (Y/N)? n

1. Create
2. Display
3. Insert Left
4. Delete Specific Element
5. Exit
Enter Your Choice: 3

Enter Value: 40

Enter the Value to Insert Left of: 20

1. Create
2. Display
3. Insert Left
4. Delete Specific Element
5. Exit
Enter Your Choice: 2

Elements in Linked List:
10 40 20 30

```

```
1. Create
2. Display
3. Insert Left
4. Delete Specific Element
5. Exit
Enter Your Choice: 4

Enter Value to Delete: 20

Element with value 20 deleted.

1. Create
2. Display
3. Insert Left
4. Delete Specific Element
5. Exit
Enter Your Choice: 2

Elements in Linked List:
10 40 30

1. Create
2. Display
3. Insert Left
4. Delete Specific Element
5. Exit
Enter Your Choice: 5

Process returned 0 (0x0)   execution time : 397.571 s
Press any key to continue.
|
```

LAB PROGRAM 9:

bfs and dfs

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

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

```
#define MAX 10
```

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

```
void enqueue(int item) {
```

```
    if (rear == MAX - 1) {
```

```
        printf("Queue is Full\n");
```

```
        return;
```

```
    }
```

```
    if (front == -1){
```

```
        front = 0;
```

```
    }
```

```
    queue[++rear] = item;
```

```
}
```

```
int dequeue() {
```

```
    if (front == -1 || front > rear) {
```

```
        printf("Queue is Empty\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] == 0){
                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);

}

```

```

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: 1
BFS Traversal: 1 3 4 0 2

Process returned 10 (0xA)   execution time : 30.652 s
Press any key to continue.
|

```

3) DFS

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

```
#define MAX 10
```

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

```
void dfs(int v);
```

```
int isConnected();
```

```
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]);
```

```
        }
```

```
    }
```

```
    printf("\nDFS Traversal: ");
```

```
    if (isConnected()) {
```

```
        printf("\nThe graph is connected.\n");
```

```
} else {  
    printf("\nThe graph is disconnected.\n");  
}
```

```
for (i = 0; i < n; i++) {  
    vis[i] = 0;  
}
```

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

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

```
void dfs(int v) {  
    printf("%d ", v);  
    vis[v] = 1;  
  
    for (int i = 0; i < n; i++) {  
        if (a[v][i] == 1 && vis[i] == 0) {  
            dfs(i);  
        }  
    }  
}
```

```
}
```

```
int isConnected() {
```

```
    int i;
```

```
    for (i = 0; i < n; i++) {
```

```
        vis[i] = 0;
```

```
    }
```

```
    dfs(0);
```

```
    for (i = 0; i < n; i++) {
```

```
        if (vis[i] == 0) {
```

```
            return 0;
```

```
        }
```

```
    }
```

```
    return 1;
```

```
}
```

```
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: 0 2 3 1 4
```

```
The graph is connected.
```

