

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT on

Data Structures (23CS3PCDST)

Submitted by

Ritu Sinha (1BM23CS271)

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

Sep-2024 to Jan-2025

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



CERTIFICATE

This is to certify that the Lab work entitled “Data Structures (23CS3PCDST)” carried out by **Ritu Sinha(1BM23CS271)**, who is a bonafide student of **B. M. S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2024. The Lab report has been approved as it satisfies the academic requirements in respect of a Data Structures (23CS3PCDST) work prescribed for the said degree.

Lab faculty Incharge Name Saritha Department of CSE, BMSCE	M. Lakshmi NEELIMA Department of CSE, BMSCE
--	--

Index

Sl. No.	Date	Experiment Title	Page No.
1	26-9-2024	Leetcode Program 1	4
2	9-10-2024	Lab Program 1 (push,pop)	5-7
3	16-10-2024	Lab Program 2(infix to postfix)	8-10
4	23-10-2024	Lab Program 3 (A) (Linear Queue)	11-13
5	23-10-2024	Lab Program 3 (B) (Circular Queue)	14-16
6	16-10-2024	Leetcode Program 2	17
7	13-11-2024	Lab Program 4 (insertion deletion)	18-23
8	20-11-2024	Lab Program 6 (reversing,concat,sort)	24-30
9	27-11-2024	Lab Program 5 (doubly linked list)	31-34
10	20-11-2024	Lab Program 7(A) (Stack implementation using linked list)	35-38
11	20-11-2024	Lab Program 7(B) (Queue implementation using linked list)	39-41
12	4-12-2024	Lab Program 8(A) (BST)	42-43
13	4-12-2024	Lab Program 8(B) (BFS)	44-45
14	4-12-2024	Leetcode Program 3	46
15	13-11-2024	Hackerrank Program	47
16	4-12-2024	Hashing program	48-49
17	4-12-2024	DFS	50-52

Github Link: <https://github.com/Ritusinhabms/1BM23CS271-RITU>

LEETCODE Program:1 Move Zeroes

Code:

```
void moveZeroes(int* nums, int n)
{
    int nonzeros=0;
    for(int i=0 ; i<n ; i++)
    {
        if(nums[i] !=0)
        {
            nums[nonzeros] = nums[i];
            nonzeros++;
        }
    }
    for(int i=nonzeros; i<n; i++)
    {
        nums[i] = 0;
    }
}
```

Output:

Testcase > Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

nums =
[0, 1, 0, 3, 12]

Output

[1, 3, 12, 0, 0]

Expected

[1, 3, 12, 0, 0]

Lab Program 1:(push,pop,display)

```
#include<stdio.h>
#include<stdlib.h>
#define max 5
int stack[max];
int top=-1;
void push(int value)
{
    if(top==max-1)
    {
        printf("stack overflow!\n");
        printf("cannot push %d into the stack\n",value);
    }
    else
    {
        top++;
        stack[top]=value;
        printf("%d pushed into the stack\n",value);
    }
    printf("element pushed\n");
}
void pop()
{
    if(top== -1)
    {
        printf("stack underflow!\n");
        printf("cannot pop %d into the stack\n",value);
    }
    else
    {
        printf("%d popped into the stack\n",value);
        top--;
    }
}
void display()
{
    if(top== -1)
    {
        printf("stack is empty\n");
    }
    else
    {

```

```

    printf("stack elements are:\n");
    for(int j=top;j>=0;j--)
    {
        printf("%d\t",stack[j]);
    }
}
int main()
{
    int CHOICE,value;
    while(1)
    {
        printf("choose an operation from below:\n");
        printf("1.PUSH\n");
        printf("1.POP\n");
        printf("1.DISPLAY\n");
        printf("1.EXIT\n");
        printf("Enter the choice:\n");
        scanf("%d",&CHOICE);

        switch(CHOICE)
        {
            case 1:
                printf("enter the number:\n");
                scanf("%d",&value);
                push(value);
                break;
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
            case 4:
                exit(0);
            default:
                printf("TRY AGAIN!");
        }
    }
}

```

Output:

```
1. Push
2. Pop
3. Display
4. Exit
1
Enter your element: 2
Element 2 is pushed
1. Push
2. Pop
3. Display
4. Exit
1
Enter your element: 3
Element 3 is pushed
1. Push
2. Pop
3. Display
4. Exit
1
Enter your element: 4
Element 4 is pushed
1. Push
2. Pop
3. Display
4. Exit
1
Enter your element: 5
Element 5 is pushed
1. Push
2. Pop
3. Display
4. Exit
1
Enter your element: 6
Element 6 is pushed
1. Push
2. Pop
3. Display
4. Exit
1
Enter your element: 3
Stack overflow
1. Push
2. Pop
3. Display
4. Exit
2
Popped from stack: 6
1. Push
2. Pop
3. Display
4. Exit
3
5 4 3 2
1. Push
2. Pop
3. Display
4. Exit
ss
```

Lab Program 2:(Infix,Postfix)

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

int index1 = 0, pos = 0, top = -1, length;
char symbol, temp, infix[20], postfix[20], stack[20];

void infixtopostfix();
void push(char symbol);
char pop();
int pred(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 (index1 < length) {
        symbol = infix[index1];
        switch (symbol) {
            case ')':
                temp = pop();
                while (temp != '(') {
                    postfix[pos++] = temp;
                    temp = pop();
                }
                break;
            case '(':
                push(symbol);
                break;
            case '+':
            case '-':
            case '*':
            case '/':
            case '^':
                while (pred(stack[top]) >= pred(symbol)) {
```



```

        temp = pop();
        postfix[pos++] = temp;
    }
    push(symbol);
    break;
default:
    postfix[pos++] = symbol;
}
index1++;
}
while (top > -1) {
    temp = pop();
    postfix[pos++] = temp;
}
}

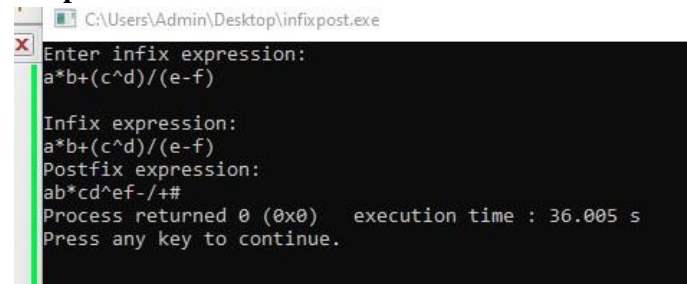
void push(char symbol) {
    top++;
    stack[top] = symbol;
}

char pop() {
    if (top == -1) {
        printf("Stack underflow error!\n");
        return '\0';
    }
    return stack[top--];
}

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

```

Output:



```
C:\Users\Admin\Desktop\infixpost.exe
Enter infix expression:
a*b+(c^d)/(e-f)

Infix expression:
a*b+(c^d)/(e-f)
Postfix expression:
ab*cd^ef-/+#
Process returned 0 (0x0) execution time : 36.005 s
Press any key to continue.
```

Lab Program 3(A):(Linear Queue)

```
#include <stdio.h>
#include <stdlib.h>
#define size 3
int Q[size];
int rear=-1;
int front=-1;
void delete1();
void insert1();
void display1();
int main()
{
    int choice;
    while(1)
    {
        printf("1.insertion\n");
        printf("2.deletion\n");
        printf("3.Display\n");
        printf("4.Exit\n");
        printf("enter your choice\n");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:insert1();
            break;
            case 2:delete1();
            break;
            case 3:display1();
            break;
            case 4:exit(1);
            default:
                printf("Invalid input\n");
        }
    }
    return 0;
}

void insert1()
{
    int item;
    if(rear==(size-1))
    {
        printf("Queue Overflow\n");
    }
}
```

```

else
{
    if(front ==-1)
        front=0;
    printf("Enter the element to be inserted\n");
    scanf("%d",&item);
    rear=rear+1;
    Q[rear]=item;
}
}
void delete1()
{
    if(front== -1||front>rear)
    {
        printf("Queue underflow\n");
        return;
    }
    else
    {
        printf("Deleted element is:%d\n",Q[front]);
        front=front+1;
    }
}
void display1()
{
    int i;
    if(front== -1)
    {
        printf("Queue is Empty\n");
    }
    else
    {
        printf("Queue elements:\n");
        for(i=front;i<=rear;i++)
        {
            printf("%d\n",Q[i]);
        }
    }
}

```

Output:

```
1.insertion
2.deletion
3.Display
4.Exit
enter your choice
1
Enter the element to be inserted
1
1.insertion
2.deletion
3.Display
4.Exit
enter your choice
1
Enter the element to be inserted
2
1.insertion
2.deletion
3.Display
4.Exit
enter your choice
1
Enter the element to be inserted
3
1.insertion
2.deletion
3.Display
4.Exit
enter your choice
2
Deleted element is:1
1.insertion
2.deletion
3.Display
4.Exit
enter your choice
2
Deleted element is:1
1.insertion
2.deletion
3.Display
4.Exit
enter your choice
1
Queue Overflow
1.insertion
2.deletion
3.Display
4.Exit
enter your choice
1
Queue Overflow
1.insertion
2.deletion
3.Display
4.Exit
enter your choice
3
Queue elements:
2
3
1.insertion
2.deletion
3.Display
4.Exit
enter your choice
2
Deleted element is:2
1.insertion
2.deletion
3.Display
4.Exit
enter your choice
3
Queue elements:
3
1.insertion
2.deletion
3.Display
4.Exit
enter your choice
4
1.insertion
2.deletion
3.Display
4.Exit
enter your choice
4
Process returned 1 (0x1)   execution time : 64.185 s
Press any key to continue.
```

Lab Program 3(B):(Circular Queue)

```
#include <stdio.h>
#include <stdlib.h>
#define size 3
int Q[size];
int rear=-1;
int front=-1;
void delete1();
void insert1();
void display1();
int main()
{
    int choice;
    while(1)
    {
        printf("1.insertion\n");
        printf("2.deletion\n");
        printf("3.Display\n");
        printf("4.Exit\n");
        printf("enter your choice\n");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:insert1();
            break;
            case 2:delete1();
            break;
            case 3:display1();
            break;
            case 4:exit(1);
            default:
                printf("Invalid input\n");
        }
    }
    return 0;
}

void insert1()
{
    int item;
    if(rear==(size-1))
    {
        printf("Queue Overflow\n");
    }
}
```

```

else
{
    if(front ==-1)
        front=0;
    printf("Enter the element to be inserted\n");
    scanf("%d",&item);
    rear=rear+1;
    Q[rear]=item;
}
}
void delete1()
{
    if(front== -1||front>rear)
    {
        printf("Queue underflow\n");
        return;
    }
    else
    {
        printf("Deleted element is:%d\n",Q[front]);
        front=front+1;
    }
}
void display1()
{
    int i;
    if(front== -1)
    {
        printf("Queue is Empty\n");
    }
    else
    {
        printf("Queue elements:\n");
        for(i=front;i<=rear;i++)
        {
            printf("%d\n",Q[i]);
        }
    }
}

```

Output:

```
1.insertion
2.deletion
3.Display
4.Exit
enter your choice
1
Enter the element to be inserted
24
1.insertion
2.deletion
3.Display
4.Exit
enter your choice
1
Enter the element to be inserted
12
1.insertion
2.deletion
3.Display
4.Exit
enter your choice
1
Enter the element to be inserted
13
1.insertion
2.deletion
3.Display
4.Exit
enter your choice
1
Queue Overflow
1.insertion
2.deletion
3.Display
4.Exit
enter your choice
2
Deleted element is:24
1.insertion
2.deletion
3.Display
4.Exit
enter your choice
3
Queue elements:
12
13
1.insertion
2.deletion
3.Display
4.Exit
enter your choice
4

Process returned 1 (0x1)   execution time : 54.416 s
Press any key to continue.
```


LEETCODE Program 2:

```
int majorityElement(int* nums, int numsSize) {  
    int i;  
    int maj=nums[0];  
    int count=0;  
    for(i=0;i<numsSize; i++)  
    {  
        if(count==0)  
        {  
            maj=nums[i];  
        }  
        if(nums[i]==maj)  
        {  
            count++;  
        }  
        else  
        {  
            count--;  
        }  
    }  
    return maj;  
}
```

Output:

Case 1

Case 2

+

nums =

[2,2,1,1,1,2,2]

Lab Program 4:(insertion deletion)

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

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

struct Node* createNode(int data) {
    struct Node *newNode = (struct Node*) malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

void insertAtBeginning(struct Node **head, int data) {
    struct Node *newNode = createNode(data);
    newNode->next = *head;
    *head = newNode;
}

void insertAtEnd(struct Node **head, int 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;
    }
}

void insertAtPosition(struct Node **head, int data, int position) {
    if (position <= 0) {
        printf("Invalid position!\n");
        return;
    }
}
```

```

}

struct Node *newNode = createNode(data);
if (position == 1) {
    newNode->next = *head;
    *head = newNode;
    return;
}

struct Node *temp = *head;
for (int i = 1; i < position - 1 && temp != NULL; i++) {
    temp = temp->next;
}

if (temp == NULL) {
    printf("Position exceeds the list length\n");
} else {
    newNode->next = temp->next;
    temp->next = newNode;
}
}

void deleteAtBeginning(struct Node **head) {
    if (*head == NULL) {
        printf("List is empty\n");
        return;
    }

    struct Node *temp = *head;
    *head = (*head)->next;
    free(temp);
}

void deleteAtPosition(struct Node **head, int position) {
    if (*head == NULL) {
        printf("List is empty\n");
        return;
    }

    if (position == 1) {
        struct Node *temp = *head;
        *head = (*head)->next;
        free(temp);
        return;
    }

```

```

    }

    struct Node *temp = *head;
    for (int i = 1; i < position - 1 && temp != NULL; i++) {
        temp = temp->next;
    }

    if (temp == NULL || temp->next == NULL) {
        printf("Position exceeds the list length\n");
    } else {
        struct Node *toDelete = temp->next;
        temp->next = temp->next->next;
        free(toDelete);
    }
}

void deleteAtEnd(struct Node **head) {
    if (*head == NULL) {
        printf("List is empty\n");
        return;
    }

    if ((*head)->next == NULL) {
        free(*head);
        *head = NULL;
        return;
    }

    struct Node *temp = *head;
    while (temp->next != NULL && temp->next->next != NULL) {
        temp = temp->next;
    }

    struct Node *toDelete = temp->next;
    temp->next = NULL;
    free(toDelete);
}

void printList(struct Node *head) {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
}

```

```

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

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

    while (1) {
        printf("\nLinked List Operations:\n");
        printf("1. Insert at Beginning\n");
        printf("2. Insert at End\n");
        printf("3. Insert at Position\n");
        printf("4. Delete at Beginning\n");
        printf("5. Delete at Position\n");
        printf("6. Delete at End\n");
        printf("7. Print List\n");
        printf("8. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter data to insert at beginning: ");
                scanf("%d", &data);
                insertAtBeginning(&head, data);
                break;

            case 2:
                printf("Enter data to insert at end: ");
                scanf("%d", &data);
                insertAtEnd(&head, data);
                break;

            case 3:
                printf("Enter data to insert: ");
                scanf("%d", &data);
                printf("Enter position to insert at: ");
                scanf("%d", &position);
                insertAtPosition(&head, data, position);

```

```

        break;

    case 4:
        deleteAtBeginning(&head);
        break;

    case 5:
        printf("Enter position to delete from: ");
        scanf("%d", &position);
        deleteAtPosition(&head, position);
        break;

    case 6:
        deleteAtEnd(&head);
        break;

    case 7:
        printList(head);
        break;

    case 8:
        exit(0);

    default:
        printf("Invalid choice\n");
    }
}

return 0;
}

```

Output:

```
Linked List Operations:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete at Position
6. Delete at End
7. Print List
8. Exit
Enter your choice: 7
22 -> 66 -> NULL
```

```
Linked List Operations:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete at Position
6. Delete at End
7. Print List
8. Exit
Enter your choice: 8
```

```
Linked List Operations:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete at Position
6. Delete at End
7. Print List
8. Exit
Enter your choice: 2
Enter data to insert at end: 66
```

```
Linked List Operations:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete at Position
6. Delete at End
7. Print List
8. Exit
Enter your choice: 4
```

```
Linked List Operations:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete at Position
6. Delete at End
7. Print List
8. Exit
Enter your choice: 5
Enter position to delete from: 0
```

```
Linked List Operations:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete at Position
6. Delete at End
7. Print List
8. Exit
Enter your choice: 1
Enter data to insert at beginning: 11
```

```
Linked List Operations:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete at Position
6. Delete at End
7. Print List
8. Exit
Enter your choice: 1
Enter data to insert at beginning: 22
```

```
Linked List Operations:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete at Beginning
5. Delete at Position
6. Delete at End
7. Print List
8. Exit
Enter your choice: 1
Enter data to insert at beginning: 44
```

Lab Program 5:(reverse,concatination,sorting)

(REVERSE)

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
    int data;
    struct node *next;
};
struct node *head=NULL;
void reverse()
{
    struct node *prevnode,*currentnode,*nextnode;
    prevnode=0;
    currentnode=nextnode=head;
    while(nextnode!=NULL)
    {
        nextnode=nextnode->next;
        currentnode->next=prevnode;
        prevnode=currentnode;
        currentnode=nextnode;
    }
    head=prevnode;
}

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

void push(int x)
{
    struct node *newnode;
    newnode=(struct node*)malloc(sizeof(struct node));
    newnode->data=x;
```



```

    newnode->next=head;
    head=newnode;
}
int main()
{
    push(20);
    push(30);
    push(40);
    printf("original list:");
    displaylist();
    reverse();
    printf("reversed list:");
    displaylist();
}

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

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

void concat(struct Node** head, int data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    struct Node* temp = *head;

    new_node->data = data;
    new_node->next = NULL;

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

void display(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d -> ", temp->data);
    }
}

```

```

        temp = temp->next;
    }
    printf("NULL\n");
}

void concatenate(struct Node** list1, struct Node** list2) {
    if (*list1 == NULL) {
        *list1 = *list2;
        return;
    }

    struct Node* last = *list1;
    while (last->next != NULL) {
        last = last->next;
    }

    last->next = *list2;
}

```

(CONCATENATION)

```

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

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

void concat(struct Node** head, int data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    struct Node* temp = *head;

    new_node->data = data;
    new_node->next = NULL;

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

```

```

        temp->next = new_node;
    }
}

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

void concatenate(struct Node** list1, struct Node** list2) {
    if (*list1 == NULL) {
        *list1 = *list2;
        return;
    }

    struct Node* last = *list1;
    while (last->next != NULL) {
        last = last->next;
    }

    last->next = *list2;
}

int main() {
    struct Node* list1 = NULL;
    struct Node* list2 = NULL;

    concat(&list1, 10);
    concat(&list1, 20);
    concat(&list1, 30);

    concat(&list2, 40);
    concat(&list2, 50);
    concat(&list2, 60);

    printf("Original list1:\n");
    display(list1);

    printf("Original list2:\n");
    display(list2);
}

```

```

concatenate(&list1, &list2);

printf("Concatenated list1 and list2:\n");
display(list1);

return 0;
}

```

(SORT)

```

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

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

void insert(struct Node** head, int data) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    struct Node* temp = *head;
    new_node->data = data;
    new_node->next = NULL;

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

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

```

```

void sort(struct Node* head) {
    struct Node* i;
    struct Node* j;
    int temp;
    for (i = head; i != NULL; i = i->next) {
        for (j = i->next; j != NULL; j = j->next) {
            if (i->data > j->data) {
                temp = i->data;
                i->data = j->data;
                j->data = temp;
            }
        }
    }
}

```

```

int main() {
    struct Node* list = NULL;

    insert(&list, 30);
    insert(&list, 10);
    insert(&list, 50);
    insert(&list, 20);
    insert(&list, 40);

    printf("Original list:\n");
    display(list);

    sort(list);

    printf("Sorted list:\n");
    display(list);

    return 0;
}

```

Output:

```

original list:40 ->30 ->20 ->NULL
reversed list:20 ->30 ->40 ->NULL

```

(Reverse)

```
Original list1:  
10 -> 20 -> 30 -> NULL  
Original list2:  
40 -> 50 -> 60 -> NULL  
Concatenated list1 and list2:  
10 -> 20 -> 30 -> 40 -> 50 -> 60 -> NULL
```

(Concat)

```
Original list:  
30 -> 10 -> 50 -> 20 -> 40 -> NULL  
Sorted list:  
10 -> 20 -> 30 -> 40 -> 50 -> NULL
```

(Sorting)

Lab Program 6:(Doubly linked list)

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

struct Node {
    int data;
    struct Node* next;
    struct Node* prev;
};

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

void insertLeft(struct Node** head, int newData, int existingValue) {
    struct Node* newNode = createNode(newData);
    struct Node* temp = *head;

    while (temp != NULL && temp->data != existingValue) {
        temp = temp->next;
    }

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

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

        temp->prev = newNode;
    } else {
        printf("Node with value %d not found.\n", existingValue);
        free(newNode);
    }
}
```

```

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

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

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

        free(temp);
        printf("Node with value %d deleted.\n", value);
    } else {
        printf("Node with value %d not found.\n", value);
    }
}

void displayList(struct Node* head) {
    struct Node* temp = head;

    if (temp == NULL) {
        printf("The list is empty.\n");
        return;
    }

    printf("Doubly Linked List: ");
    while (temp != NULL) {
        printf("%d <-> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

int main() {

```



```

struct Node* head = NULL;
int n, value, newValue, existingValue;

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

for (int i = 0; i < n; i++) {
    printf("Enter value for node %d: ", i + 1);
    scanf("%d", &value);

    struct Node* newNode = createNode(value);
    if (head == NULL) {
        head = newNode;
    } else {
        struct Node* temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->prev = temp;
    }
}

displayList(head);

printf("Enter the value of the node to insert a new node before of: ");
scanf("%d", &existingValue);
printf("Enter the value of the new node to insert: ");
scanf("%d", &newValue);
insertLeft(&head, newValue, existingValue);
displayList(head);

printf("Enter the value of the node to delete: ");
scanf("%d", &value);
deleteNode(&head, value);
displayList(head);

return 0;
}

```

Output:

```
Enter the number of nodes: 4
Enter value for node 1: 30
Enter value for node 2: 12
Enter value for node 3: 21
Enter value for node 4: 13
Doubly Linked List: 30 <--> 12 <--> 21 <--> 13 <--> NULL
Enter the value of the node to insert a new node before of: 12
Enter the value of the new node to insert: 52
Doubly Linked List: 30 <--> 52 <--> 12 <--> 21 <--> 13 <--> NULL
Enter the value of the node to delete: 12
Node with value 12 deleted.
Doubly Linked List: 30 <--> 52 <--> 21 <--> 13 <--> NULL
```

Lab Program7(A):(stack implementation using linked list)

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int data;
    struct node *next;
};
struct node* head=NULL;
struct node* ptr=NULL;
int top1=-1;
struct node* createnode(int value)
{
    struct node* newnode=(struct node*)malloc(sizeof(struct node));
    newnode->data=value;
    newnode->next=NULL;
}
void printstack()
{
    ptr=head;
    while(ptr->next!=NULL)
    {
        printf("%d->",ptr->data);
        ptr=ptr->next;
    }
    printf("%d\nEND OF STACK\n",ptr->data);
}
void push(int value)
{
    if (top1==4)
    {
        printf("Stack is full, element not pushed\n");
    }
    else
    {
        struct node* newnode = createnode(value);
        if (head==NULL)
        {
            head=newnode;
            ptr=newnode;
        }
        else
        {
```

```

        ptr->next=newnode;
        ptr=newnode;
    }
    printf("Element pushed is %d\n",ptr->data);
    top1++;
}
}
void pop()
{
    if (head == NULL)
    {
        printf("Stack is empty, nothing to delete\n");
    }
    else if (head->next == NULL)
    {
        printf("Element deleted is %d\n", head->data);
        free(head);
        head = NULL;
        ptr = NULL;
        top1--;
    }
    else
    {
        struct node* deleteptr;
        struct node* preptr = head;
        while (preptr->next->next != NULL)
        {
            preptr = preptr->next;
        }
        deleteptr = preptr->next;
        preptr->next = NULL;
        ptr = preptr;
        printf("Element deleted is %d\n", deleteptr->data);
        free(deleteptr);
        top1--;
    }
}
void peek()
{
    if (ptr==NULL)
    {
        printf("Stack is empty dawg\n");
    }
    else

```

```

    {
        printf("%d\n",ptr->data);
    }
}
void main()
{
    int n,pushedValue;
    for(int i=0;i<=10000;i++)
    {
        printf("Enter \n1 to push\n2 to pop\n3 to display\n4 to peek\n5 to exit");
        scanf("%d",&n);
        if (n==1)
        {
            printf("Enter value to be pushed");
            scanf("%d",&pushedValue);
            push(pushedValue);
        }
        else if (n==2)
        {
            pop();
        }
        else if(n==3)
        {
            printstack();
        }
        else if (n==4)
        {
            peek();
        }
        else if(n==5)
        {
            exit(0);
        }
        else
        {
            printf("Invalid input");
        }
    }
}

```

Output:

```
Enter
1 to push
2 to pop
3 to display
4 to peek
5 to exit
1
Enter value to be pushed24
Element pushed is 24
Enter
1 to push
2 to pop
3 to display
4 to peek
5 to exit
1
Enter value to be pushed12
Element pushed is 12
Enter
1 to push
2 to pop
3 to display
4 to peek
5 to exit
1
Enter value to be pushed13
Element pushed is 13
Enter
1 to push
2 to pop
3 to display
4 to peek
5 to exit1
Enter value to be pushed30
Element pushed is 30
Enter
1 to push
2 to pop
3 to display
4 to peek
5 to exit
2
Element deleted is 30
Enter
1 to push
2 to pop
3 to display
4 to peek
5 to exit
3
24->12->13
END OF STACK
Enter
1 to push
2 to pop
3 to display
4 to peek
5 to exit|
```

Lab Program 7(B):(Queue implementation using linked list)

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

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

struct Queue {
    struct Node* front;
    struct Node* rear;
};

void enqueue(struct Queue* q, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    if (q->rear == NULL) {
        q->front = q->rear = newNode;
        return;
    }
    q->rear->next = newNode;
    q->rear = newNode;
}

void dequeue(struct Queue* q) {
    if (q->front == NULL) {
        printf("Queue is empty, cannot dequeue.\n");
        return;
    }
    struct Node* temp = q->front;
    q->front = q->front->next;
    if (q->front == NULL) {
        q->rear = NULL;
    }
    free(temp);
}

void display(struct Queue* q) {
    if (q->front == NULL) {
        printf("Queue is empty.\n");
        return;
    }
}
```

```

    }
    struct Node* temp = q->front;
    printf("Queue elements: ");
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    struct Queue q;
    q.front = q.rear = NULL;
    int choice, value;

    while (1) {
        printf("1. Enqueue\n");
        printf("2. Dequeue\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to enqueue: ");
                scanf("%d", &value);
                enqueue(&q, value);
                break;
            case 2:
                dequeue(&q);
                break;
            case 3:
                display(&q);
                break;
            case 4:
                exit(0);
            default:
                printf("Invalid choice!\n");
        }
    }
    return 0;
}

```


Output:

```
Enter your choice: 1
Enter value to enqueue: 10
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 20
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 30
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue elements: 10 20 30
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 10
Invalid choice!
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue elements: 20 30
1. Enqueue
2. Dequeue
3. Display
4. Exit
```

Lab Program 8(A):(BST)

```
#include <stdio.h>
#include <stdlib.h>
struct node {
    int data;
    struct node *left;
    struct node *right;
};

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

struct node* insert(struct node* root, int data) {
    if (root == NULL)
        return newNode(data);

    if (data < root->data)
        root->left = insert(root->left, data);
    else if (data > root->data)
        root->right = insert(root->right, data);

    return root;
}

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

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

```

    }
}

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

void display(struct node* root) {
    printf("inorder traversal: ");
    inorder(root);

    printf("preorder traversal: ");
    preorder(root);

    printf("postorder traversal: ");
    postorder(root);
}

int main() {
    struct node* root = NULL;
    int n, data;

    printf("Enter the number of nodes to insert in the BST: ");
    scanf("%d", &n);

    for (int i = 0; i < n; i++) {
        printf("Enter value for node %d: ", i + 1);
        scanf("%d", &data);
        root = insert(root, data);
    }

    display(root);

    return 0;
}

```

Output:

```

Enter the number of nodes to insert in the BST: 6
Enter value for node 1: 10
Enter value for node 2: 4
Enter value for node 3: 5
Enter value for node 4: 12
Enter value for node 5: 25
Enter value for node 6: 1
inorder traversal: 1 4 5 10 12 25 preorder traversal: 10 4 1 5 12 25 postorder traversal: 1 5 4 25 12 10

```

Lab Program 8(B):(BFS)

```
#include <stdio.h>
void bfs(int a[10][10], int n, int u, int arr[10]) {
    int f, r, q[10], v;
    printf("The nodes visited from %d: ", u);
    f = 0;
    r = -1;
    q[++r] = u;
    arr[u] = 1;
    printf("%d ", u);
    while (f <= r) {
        u = q[f++];
        for (v = 0; v < n; v++) {
            if (a[u][v] == 1) {
                if (arr[v] == 0) {
                    printf("%d ", v);
                    arr[v] = 1;
                    q[++r] = v;
                }
            }
        }
    }
    printf("\n");
}
```

```
int main() {
    int n, a[10][10], source, i, j;
    int arr[10] = {0};
    printf("Enter number of nodes: ");
    scanf("%d", &n);
    printf("The adjacency matrix:\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            scanf("%d", &a[i][j]);
        }
    }
    for (source = 0; source < n; source++) {
        for (i = 0; i < n; i++) {
            arr[i] = 0;
        }
        bfs(a, n, source, arr);
    }
    return 0;
}
```

}

Output:

```
Enter number of nodes: 4
The adjacency matrix:
0 1 1 0
1 0 1 1
1 1 0 1
0 1 1 0
The nodes visited from 0: 0 1 2 3
The nodes visited from 1: 1 0 2 3
The nodes visited from 2: 2 0 1 3
The nodes visited from 3: 3 1 2 0
```

LEETCODE Program 3:

```
class Solution {
public:
    bool hasPathSum(TreeNode* root, int targetSum) {
        int rem=targetSum;
        if(root==NULL)
            return false;
        if(root->left == NULL && root->right==NULL){
            if(root->val == targetSum)
                return true;
            else
                return false;
        }
        rem -= root->val;
        return hasPathSum(root->left,rem) || hasPathSum(root->right, rem);
    }
};
```

Output:

• Case 1		• Case 2		• Case 3	
Input		Input		Input	
root = [5,4,8,11,null,13,4,7,2,null,null,null,1]		root = [1,2,3]		root = []	
targetSum = 22		targetSum = 5		targetSum = 0	
Output		Output		Output	
true		false		false	
Expected		Expected		Expected	
true		false		false	

HackerRank Program:

```
int two_stacks(int maxsum, int a_count, int *a, int b_count, int *b) {
    int sum = 0;
    int Score = 0;
    int i = 0;
    int j = 0;

    while (i < a_count && j < b_count) {
        if (a[i] <= b[j] && (sum + a[i] <= maxsum)) {
            sum += a[i];
            Score++;
            i++;
            if (sum == maxsum) break;
        } else if (a[i] > b[j] && (sum + b[j] <= maxsum)) {
            sum += b[j];
            Score++;
            j++;
            if (sum == maxsum) break;
        } else {
            break;
        }
    }
    return Score;
}
```

Lab Program 9:(Hashing program)

```
#include <stdio.h>
#include <stdlib.h>
#define TABLE_SIZE 10

int h[TABLE_SIZE] = {0};

void insert() {
    int key, index, i, hkey;
    printf("\nEnter a value to insert into hash table: ");
    scanf("%d", &key);
    hkey = key % TABLE_SIZE;
    for (i = 0; i < TABLE_SIZE; i++) {
        index = (hkey + i) % TABLE_SIZE;
        if (h[index] == 0) {
            h[index] = key;
            break;
        }
    }
    printf("Number of probes for %d is %d\n", key, i + 1);
    if (i == TABLE_SIZE) {
        printf("\nElement cannot be inserted\n");
    }
}

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

void display() {
    int i;
```



```

printf("\nElements in the hash table are:\n");
for (i = 0; i < TABLE_SIZE; i++) {
    printf("At index %d \t value = %d\n", i, h[i]);
}
}

int main() {
    int opt;
    while (1) {
        printf("\nPress 1. Insert\t 2. Display \t 3. Search \t 4. Exit\n");
        scanf("%d", &opt);
        switch (opt) {
            case 1: insert(); break;
            case 2: display(); break;
            case 3: search(); break;
            case 4: exit(0);
            default: printf("Invalid option. Please try again.\n");
        }
    }
    return 0;
}

```

Output:

```

Press 1. Insert  2. Display  3. Search  4. Exit
1
Enter a value to insert into hash table: 24
Number of probes for 24 is 1

Press 1. Insert  2. Display  3. Search  4. Exit
1
Enter a value to insert into hash table: 12
Number of probes for 12 is 1

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

```

```

Elements in the hash table are:
At index 0  value = 0
At index 1  value = 0
At index 2  value = 12
At index 3  value = 0
At index 4  value = 24
At index 5  value = 0
At index 6  value = 0
At index 7  value = 0
At index 8  value = 0
At index 9  value = 0

Press 1. Insert  2. Display  3. Search  4. Exit
3
Enter search element:
12
Value is found at index 2

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

```

Program 10 : (DFS)

```
#include <stdio.h>
int a[10][10], vis[10], parent[10], n;
char nodes[10];
int dfs(int v)
{
    vis[v] = 1;
    printf("%c ", nodes[v]);
    for (int j = 0; j < n; j++)
    {
        if (a[v][j] == 1 && vis[j] == 0)
        {
            parent[j] = v;
            if (dfs(j))
                return 1;
        }
        else if (a[v][j] == 1 && vis[j] == 1 && parent[v] != j)
        {
            printf("\nCycle detected!\n");
            return 1;
        }
    }
    return 0;
}
int isConnected()
{
    for (int i = 0; i < n; i++)
    {
        if (vis[i] == 0)

    return 0;
    }
    return 1;
}
int main()
{
    printf("Enter number of vertices: ");
    scanf("%d", &n);
    printf("Enter the characters for the nodes (e.g., A, B, C, etc.): \n");
    for (int i = 0; i < n; i++)
    {
        printf("Node %d: ", i + 1);
        scanf(" %c", &nodes[i]);
```

```

}
printf("Enter adjacency matrix (0 or 1) :\n");
for (int i = 0; i < n; i++)
{
for (int j = 0; j < n; j++)
{
scanf("%d", &a[i][j]);
}
vis[i] = 0;
parent[i] = -1;
}
printf("DFS Traversal: ");
for (int i = 0; i < n; i++)
{
if (vis[i] == 0)
{

if (dfs(i))
{
break;
}
}

if (isConnected())
{
printf("\nThe graph is connected.\n");
}
else
{
printf("\nThe graph is not connected.\n");
}
return 0;
}

```

Output:

```
Enter number of vertices: 5
Enter the characters for the nodes (e.g., A, B, C, etc.):
Node 1: A
Node 2: B
Node 3: C
Node 4: D
Node 5: E
Enter adjacency matrix (0 or 1) :
0 1 1 1 0
0 0 0 1 0
0 1 0 0 0
0 0 0 0 1
0 0 1 0 0
DFS Traversal: A B D E C
Cycle detected!

The graph is connected.
```