

**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**  
“JnanaSangama”, Belgaum -590014, Karnataka.



**LAB REPORT  
On**

**DATA STRUCTURES (23CS3PCDST)**

**Submitted by**

**VANITHA(1BM22CS317)**

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



**B.M.S. COLLEGE OF ENGINEERING  
(Autonomous Institution under VTU)  
BENGALURU-560019  
Dec 2023- March 2024**

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



This is to certify that the Lab work entitled "**DATA STRUCTURES**" carried out by **VANITHA(1BM22CS317)**, who is bona fide student of **B. M. S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2023-24. The Lab report has been approved as it satisfies the academic requirements in respect of Data structures Lab - **(23CS3PCDST )**work prescribed for the said degree.

**Prof. Lakshmi Neelima**  
Assistant Professor  
Department of CSE  
BMSCE, Bengaluru

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

## Index Sheet

Sl. No.	Experiment Title	Page No.
1	Stack Implementation using Array	3
2	Conversion of Infix Expression to Postfix Expression	8
3	Queue Implementation using Array	13
4	Circular Queue Implementation using Array	19
5	Singly Linked List Implementation:Insertion	25
6	Singly Linked List Operations: Deletion	32
7	Single Linked List Operations: Sorting, Reversing, Concatenating	41
8a	Stack implementation using single linked list	50
8b	Queue implementation using single linked list	56
9	Doubly Linked List Implementation	62
10	Binary Search Tree Construction and Traversal	70
11	Graph Traversal using BFS	76
12	Checking Graph Connectivity using DFS	80
13	Leetcode Programms	82

### Course outcomes:

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

### Lab program 1:

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

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

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

```
#include <stdio.h>
#include<stdlib.h>
#define STACK_SIZE 5
```

```

void push(int st[],int *top)
{
    int item;
    if(*top==STACK_SIZE-1)
        printf("Stack overflow\n");
    else
    {
        printf("\nEnter an item :");
        scanf("%d",&item);
        (*top)++;
        st[*top]=item;
    }
}
void pop(int st[],int *top)
{
    if(*top===-1)
        printf("Stack underflow\n");
    else
    {
        printf("\n%d item was deleted",st[(*top)--]);
    }
}
void display(int st[],int *top)
{
    int i;
    if(*top===-1)
        printf("Stack is empty\n");
    for(i=0;i<=*top;i++)
        printf("%d\t",st[i]);
}
void main()
{
    int st[10],top=-1, c,val_del;
    while(1)
    {
        printf("\n1. Push\n2. Pop\n3. Display\n");
        printf("\nEnter your choice :");
        scanf("%d",&c);
        switch(c)
        {
            case 1: push(st,&top);
                      break;
            case 2: pop(st,&top);
                      break;
            case 3: display(st,&top);
        }
    }
}

```

```
        break;
    default: printf("\nInvalid choice!!!");
        exit(0);
    }
}
}
```

### Output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS D:\jyothika\dst> cd "d:\jyothika\dst\" ; if ($?) { gcc 1.c -o 1 } ; if ($?) { .\1 }

1. Push
2. Pop
3. Display

Enter your choice :1

Enter an item :12

1. Push
2. Pop
3. Display

Enter your choice :1

Enter an item :65

1. Push
2. Pop
3. Display

Enter your choice :1

Enter an item :45

1. Push
2. Pop
3. Display

Enter your choice :1
Stack overflow
```

- 1. Push
- 2. Pop
- 3. Display

Enter your choice :2

45 item was deleted

- 1. Push
- 2. Pop
- 3. Display

Enter your choice :2

65 item was deleted

- 1. Push
- 2. Pop
- 3. Display

Enter your choice :3

12

- 1. Push
- 2. Pop
- 3. Display

Enter your choice :2

12 item was deleted

- 1. Push
- 2. Pop
- 3. Display

Enter your choice :2

Stack underflow

- 1. Push
- 2. Pop
- 3. Display

Enter your choice :4

Invalid choice!!!

```
    printf("0\n"); i++ ; }  
    printf("%d ", st[i]);  
    printf("\n");
```

{

}

dp : 1.push

2.pop

3.display

Enter your choice : 1

Enter an item : 12

1.push

2.pop

3.display

Enter your choice : 1

Enter an item : 65

Enter your choice : 2

65 item was deleted

Enter your choice : 3

12

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

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

#define MAX_SIZE 100

// Stack implementation
typedef struct {

    char data[MAX_SIZE];
    int top;
} Stack;

void initialize(Stack* stack) {
    stack->top = -1;
}

void push(Stack* stack, char item) {
    if (stack->top < MAX_SIZE - 1) {
        stack->data[++stack->top] = item;
    } else {
        printf("Error: Stack overflow\n");
        exit(EXIT_FAILURE);
    }
}

char pop(Stack* stack) {
    if (stack->top >= 0) {
```

```

        return stack->data[stack->top--];

    } else {
        printf("Error: Stack underflow\n");
        exit(EXIT_FAILURE);
    }
}

int isOperator(char ch) {
    return (ch == '+' || ch == '-' || ch == '*' || ch == '/');
}

int precedence(char operator) {
    if (operator == '+' || operator == '-')
        return 1;
    if (operator == '*' || operator == '/')
        return 2;
    return 0; // Default case
}

void infixToPostfix(char infix[], char postfix[]) {
    Stack stack;
    initialize(&stack);

    int i = 0, j = 0;

    while (infix[i] != '\0') {
        char currentSymbol = infix[i];

        if (currentSymbol == '(') {

```

```

        push(&stack, currentSymbol);

    } else if (isalnum(currentSymbol)) {

        postfix[j++] = currentSymbol;

    } else if (isOperator(currentSymbol)) {

        while (stack.top >= 0 && precedence(stack.data[stack.top]) >=
precedence(currentSymbol)) {

            postfix[j++] = pop(&stack);

        }

        push(&stack, currentSymbol);

    } else if (currentSymbol == ')') {

        while (stack.top >= 0 && stack.data[stack.top] != '(') {

            postfix[j++] = pop(&stack);

        }

        if (stack.top >= 0 && stack.data[stack.top] == '(') {

            pop(&stack); // Pop '(' from stack

        } else {

            printf("Error: Mismatched parentheses\n");

            exit(EXIT_FAILURE);

        }

    }

    i++;

}

// Pop remaining operators from stack

while (stack.top >= 0) {

    if (stack.data[stack.top] == '(') {

        printf("Error: Mismatched parentheses\n");

        exit(EXIT_FAILURE);

    }

}

```

```

postfix[j++] = pop(&stack);
}

postfix[j] = '\0'; // Null-terminate the postfix expression
}

int main() {
    char infix[MAX_SIZE], postfix[MAX_SIZE];

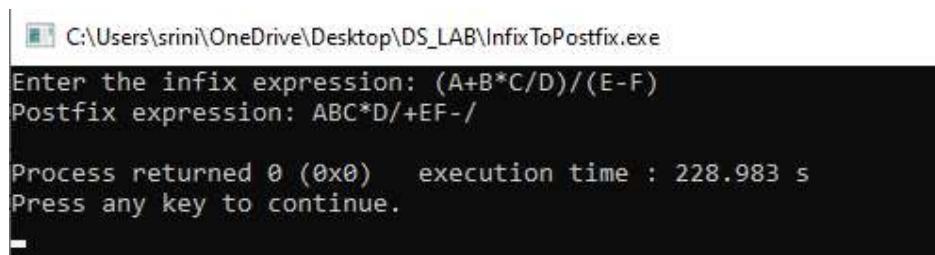
    printf("Enter the infix expression: ");
    scanf("%s", infix);

    infixToPostfix(infix, postfix);

    printf("Postfix expression: %s\n", postfix);

    return 0;
}

```



```

C:\Users\sri\OneDrive\Desktop\DS_LAB\InfixToPostfix.exe
Enter the infix expression: (A+B*C/D)/(E-F)
Postfix expression: ABC*D/+EF-
Process returned 0 (0x0)   execution time : 228.983 s
Press any key to continue.

```

char pop (char st[])

{

    char val = `` ;

    if (top == -1)

        printf ("In stack underflow") ;

    else

    {

        val = st [top] ;

        top -- ;

    }

}

return val ;

01/01/2024

O/P : Enter any infix expression : A + B

The corresponding postfix expression is : AB +

**3.write a program to simulate the working of the queue of integers using an array. Provide the following operations: Insert, delete, display. The program should print appropriate message for overflow and underflow condition.**

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

#define MAX_SIZE 5

typedef struct {
    int data[MAX_SIZE];
    int front, rear;
} Queue;

void initialize(Queue* queue) {
    queue->front = -1;
    queue->rear = -1;
}

int isEmpty(Queue* queue) {
    return (queue->front == -1 && queue->rear == -1);
}

int isFull(Queue* queue) {
    return ((queue->rear + 1) % MAX_SIZE == queue->front);
}

void enqueue(Queue* queue, int value) {
    if (isFull(queue)) {
        printf("Error: Queue Overflow\n");
    }
}
```

```

    exit(EXIT_FAILURE);

}

if (isEmpty(queue)) {
    queue->front = 0; // Initialize front for the first element
}

queue->rear = (queue->rear + 1) % MAX_SIZE; // Circular increment for wrapping around
queue->data[queue->rear] = value;

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

int dequeue(Queue* queue) {
    int value;

    if (isEmpty(queue)) {
        printf("Error: Queue Underflow\n");
        exit(EXIT_FAILURE);
    }

    value = queue->data[queue->front];

    if (queue->front == queue->rear) {
        // Last element in the queue
        initialize(queue);
    } else {
        queue->front = (queue->front + 1) % MAX_SIZE; // Circular increment for wrapping
        around
    }
}

```

```

printf("Dequeued %d\n", value);

return value;
}

void display(Queue* queue) {
    if (isEmpty(queue)) {
        printf("Queue is empty\n");
        return;
    }

    printf("Queue: ");
    int i = queue->front;

    do {
        printf("%d ", queue->data[i]);
        i = (i + 1) % MAX_SIZE; // Circular increment for wrapping around
    } while (i != (queue->rear + 1) % MAX_SIZE);

    printf("\n");
}

int main() {
    Queue myQueue;
    initialize(&myQueue);

    int choice, value;

```

```

do {
    printf("\nMenu:\n");
    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 the value to enqueue: ");
            scanf("%d", &value);
            enqueue(&myQueue, value);
            break;
        case 2:
            dequeue(&myQueue);
            break;
        case 3:
            display(&myQueue);
            break;
        case 4:
            printf("Exiting program.\n");
            break;
        default:
            printf("Invalid choice. Please enter a valid option.\n");
    }
} while (choice != 4);

```

```
    return 0;  
}  
  
C:\Users\sri\OneDrive\Desktop\ArrayQueue.exe
```

```
Menu:  
1. Enqueue  
2. Dequeue  
3. Display  
4. Exit  
Enter your choice: 1  
Enter the value to enqueue: 10  
Enqueued 10  
  
Menu:  
1. Enqueue  
2. Dequeue  
3. Display  
4. Exit  
Enter your choice: 1  
Enter the value to enqueue: 20  
Enqueued 20  
  
Menu:  
1. Enqueue  
2. Dequeue  
3. Display  
4. Exit  
Enter your choice: 3  
Queue: 10 20
```

```
Menu:  
1. Enqueue  
2. Dequeue  
3. Display  
4. Exit  
Enter your choice: 2  
Dequeued 10
```

```
Menu:  
1. Enqueue  
2. Dequeue  
3. Display  
4. Exit  
Enter your choice: 3  
Queue: 20
```

```
Menu:  
1. Enqueue  
2. Dequeue  
3. Display  
4. Exit  
Enter your choice: Queue: 20
```

output :

1. Insert

2. Delete

3. Display

4. Exit

Enter your choice : 1

Enter element to insert : 10

1. Insert

2. Delete

3. Display

4. Exit

Enter your choice : 1

Enter element to insert : 20

1. Insert

2. Delete

3. Display

4. Exit

Enter your choice : 3

Queue is :

10

20

1. Insert

2. Delete

3. Display

4. Exit

Enter your choice : 2

Deleted element is : 10

**4.write a program to simulate the working of a circular queue using an array.  
Provide the following operations: insert, delete & display. The program should print appropriate message for queue empty and queue overflow conditions.**

```
#include <stdio.h>

#define MAX_SIZE 5

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

// Function to check if the queue is empty
int isEmpty() {
    return (front == -1 && rear == -1);
}

// Function to check if the queue is full
int isFull() {
    return (front == (rear + 1) % MAX_SIZE);
}

// Function to insert an element into the queue
void insert(int item) {
    if (isFull()) {
        printf("Queue Overflow: Unable to insert element.\n");
        return;
    } else if (isEmpty()) {
        front = rear = 0;
    } else {
        rear = (rear + 1) % MAX_SIZE;
```

```

    }

    queue[rear] = item;
    printf("Element %d inserted successfully.\n", item);
}

// Function to delete an element from the queue
void delete() {
    if (isEmpty()) {
        printf("Queue Underflow: Unable to delete element.\n");
        return;
    } else if (front == rear) {
        printf("Element %d deleted successfully.\n", queue[front]);
        front = rear = -1;
    } else {
        printf("Element %d deleted successfully.\n", queue[front]);
        front = (front + 1) % MAX_SIZE;
    }
}

// Function to display the elements of the queue
void display() {
    if (isEmpty()) {
        printf("Queue is empty.\n");
        return;
    }

    printf("Queue elements: ");
    int i = front;

```

```

do {
    printf("%d ", queue[i]);
    i = (i + 1) % MAX_SIZE;
} while (i != (rear + 1) % MAX_SIZE);
printf("\n");

}

int main() {
    int choice, item;

    do {
        printf("\nCircular Queue Operations:\n");
        printf("1. Insert\n");
        printf("2. Delete\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the element to be inserted: ");
                scanf("%d", &item);
                insert(item);
                break;
            case 2:
                delete();
                break;
            case 3:

```

```

        display();
        break;
    case 4:
        printf("Exiting program.\n");
        break;
    default:
        printf("Invalid choice. Please enter a valid option.\n");
    }
} while (choice != 4);

return 0;
}

```

```

Circular Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the element to be inserted: 5
Element 5 inserted successfully.

Circular Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the element to be inserted: 10
Element 10 inserted successfully.

Circular Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the element to be inserted: 15
Element 15 inserted successfully.

```

```
Circular Queue Operations:  
1. Insert  
2. Delete  
3. Display  
4. Exit  
Enter your choice: 3  
Queue elements: 5 10 15  
  
Circular Queue Operations:  
1. Insert  
2. Delete  
3. Display  
4. Exit  
Enter your choice: 2  
Element 5 deleted successfully.  
  
Circular Queue Operations:  
1. Insert  
2. Delete  
3. Display  
4. Exit  
Enter your choice: 3  
Queue elements: 10 15
```

Output :

1. Insert
2. delete
3. display
4. exit

Enter choice : 1

Enter element to Insert : 5  
5 is inserted

1. Insert
2. delete
3. display
4. exit

Enter choice : 1

Enter element to Insert : 10  
10 is inserted

8/10/24

**5.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.**

**Display the contents of the linked list.**

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

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

// Function to create a new node
struct Node* createNode(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }
    newNode->data = value;
    newNode->next = NULL;
    return newNode;
}

// Function to insert a node at the first position
struct Node* insertAtFirst(struct Node* head, int value) {
    struct Node* newNode = createNode(value);
```

```

newNode->next = head;
head = newNode;
printf("Node with value %d inserted at the beginning.\n", value);
return head;
}

// Function to insert a node at any position
struct Node* insertAtPosition(struct Node* head, int value, int position) {
    if (position < 1) {
        printf("Invalid position. Position should be greater than 0.\n");
        return head;
    }

    struct Node* newNode = createNode(value);

    if (position == 1) {
        newNode->next = head;
        head = newNode;
        printf("Node with value %d inserted at position %d.\n", value, position);
        return head;
    }

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

    if (temp == NULL) {
        printf("Invalid position. Position exceeds the length of the list.\n");
    }
}

```

```

        return head;
    }

newNode->next = temp->next;
temp->next = newNode;
printf("Node with value %d inserted at position %d.\n", value, position);
return head;
}

// Function to insert a node at the end of the list
struct Node* insertAtEnd(struct Node* head, int 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;
    }

    printf("Node with value %d inserted at the end.\n", value);
    return head;
}

// Function to display the contents of the linked list
void display(struct Node* head) {

```

```

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

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

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

    do {
        printf("\nSingly Linked List Operations:\n");
        printf("1. Insert at First\n");
        printf("2. Insert at Any Position\n");
        printf("3. Insert at End\n");
        printf("4. Display\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {

```

```
case 1:  
    printf("Enter the value to insert at the first position: ");  
    scanf("%d", &value);  
    head = insertAtFirst(head, value);  
    break;  
  
case 2:  
    printf("Enter the value to insert: ");  
    scanf("%d", &value);  
    printf("Enter the position to insert at: ");  
    scanf("%d", &position);  
    head = insertAtPosition(head, value, position);  
    break;  
  
case 3:  
    printf("Enter the value to insert at the end: ");  
    scanf("%d", &value);  
    head = insertAtEnd(head, value);  
    break;  
  
case 4:  
    display(head);  
    break;  
  
case 5:  
    printf("Exiting program.\n");  
    break;  
  
default:  
    printf("Invalid choice. Please enter a valid option.\n");  
}  
} while (choice != 5);  
return 0;  
}
```

```
Singly Linked List Operations:  
1. Insert at First  
2. Insert at Any Position  
3. Insert at End  
4. Display  
5. Exit  
Enter your choice: 1  
Enter the value to insert at the first position: 10  
Node with value 10 inserted at the beginning.
```

```
Singly Linked List Operations:  
1. Insert at First  
2. Insert at Any Position  
3. Insert at End  
4. Display  
5. Exit  
Enter your choice: 1  
Enter the value to insert at the first position: 20  
Node with value 20 inserted at the beginning.
```

```
Singly Linked List Operations:  
1. Insert at First  
2. Insert at Any Position  
3. Insert at End  
4. Display  
5. Exit  
Enter your choice: 3  
Enter the value to insert at the end: 30  
Node with value 30 inserted at the end.
```

```
Singly Linked List Operations:  
1. Insert at First  
2. Insert at Any Position  
3. Insert at End  
4. Display  
5. Exit  
Enter your choice: 2  
Enter the value to insert: 25  
Enter the position to insert at: 3  
Node with value 25 inserted at position 3.
```

```
Singly Linked List Operations:  
1. Insert at First  
2. Insert at Any Position  
3. Insert at End  
4. Display  
5. Exit  
Enter your choice: 4  
Linked list elements: 20 10 25 30
```

Output :

\* \* \* Main menu \* \* \*

1. create linked list
2. display
3. Insert\_beg
4. Insert\_end
5. Insert\_before
6. Insert\_after

Enter your choice : 3

Enter the data : 10

Enter your choice : 4

Enter the data : 40

→ Enter your choice : 5

Enter the value before which data has to be  
inserted : 40

Enter the data : 20

Enter your choice : 6

Enter the data : 30

Enter the value after which data has to be  
inserted : 20

**6.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.**

**Display the contents of the linked list.**

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

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

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

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

printf("Node with value %d inserted at the end.\n", value);
return head;
}

struct Node* deleteFirst(struct Node* head) {
    if (head == NULL) {
        printf("Linked list is empty. Cannot delete.\n");
        return head;
    }

    struct Node* temp = head;
    head = head->next;
    free(temp);
    printf("First element deleted successfully.\n");
    return head;
}

struct Node* deleteElement(struct Node* head, int value) {
    if (head == NULL) {
        printf("Linked list is empty. Cannot delete.\n");
        return head;
}

```

```

}

struct Node* temp = head;
struct Node* prev = NULL;

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

if (temp == NULL) {
    printf("Element %d not found in the linked list. Cannot delete.\n", value);
    return head;
}

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

free(temp);
printf("Element %d deleted successfully.\n", value);
return head;
}

struct Node* deleteLast(struct Node* head) {
    if (head == NULL) {
        printf("Linked list is empty. Cannot delete.\n");
    }
}

```

```

    return head;
}

if (head->next == NULL) {
    free(head);
    head = NULL;
    printf("Last element deleted successfully.\n");
    return head;
}

struct Node* temp = head;
struct Node* prev = NULL;

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

prev->next = NULL;
free(temp);
printf("Last element deleted successfully.\n");
return head;
}

void display(struct Node* head) {
    if (head == NULL) {
        printf("Linked list is empty.\n");
        return;
    }
}

```

```

printf("Linked list elements: ");

struct Node* temp = head;

while (temp != NULL) {

    printf("%d ", temp->data);

    temp = temp->next;

}

printf("\n");

}

int main() {

    struct Node* head = NULL;

    int choice, value;

    do {

        printf("\nSingly Linked List Operations:\n");

        printf("1. Insert at End\n");
        printf("2. Delete First\n");
        printf("3. Delete Specified Element\n");
        printf("4. Delete Last\n");
        printf("5. Display\n");
        printf("6. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch (choice) {

            case 1:

                printf("Enter the value to insert at the end: ");
                scanf("%d", &value);

```

```
head = insertAtEnd(head, value);
break;

case 2:
    head = deleteFirst(head);
    break;

case 3:
    printf("Enter the value to delete: ");
    scanf("%d", &value);
    head = deleteElement(head, value);
    break;

case 4:
    head = deleteLast(head);
    break;

case 5:
    display(head);
    break;

case 6:
    printf("Exiting program.\n");
    break;

default:
    printf("Invalid choice. Please enter a valid option.\n");

}

} while (choice != 6);

return 0;
}
```

```
Singly Linked List Operations:  
1. Insert at End  
2. Delete First  
3. Delete Specified Element  
4. Delete Last  
5. Display  
6. Exit  
Enter your choice: 1  
Enter the value to insert at the end: 10  
Node with value 10 inserted at the end.  
  
Singly Linked List Operations:  
1. Insert at End  
2. Delete First  
3. Delete Specified Element  
4. Delete Last  
5. Display  
6. Exit  
Enter your choice: 1  
Enter the value to insert at the end: 20  
Node with value 20 inserted at the end.  
  
Singly Linked List Operations:  
1. Insert at End  
2. Delete First  
3. Delete Specified Element  
4. Delete Last  
5. Display  
6. Exit  
Enter your choice: 1  
Enter the value to insert at the end: 30  
Node with value 30 inserted at the end.  
  
Singly Linked List Operations:  
1. Insert at End  
2. Delete First  
3. Delete Specified Element  
4. Delete Last  
5. Display  
6. Exit  
Enter your choice: 1  
Enter the value to insert at the end: 40  
Node with value 40 inserted at the end.
```

```
Singly Linked List Operations:  
1. Insert at End  
2. Delete First  
3. Delete Specified Element  
4. Delete Last  
5. Display  
6. Exit  
Enter your choice: 2  
First element deleted successfully.  
  
Singly Linked List Operations:  
1. Insert at End  
2. Delete First  
3. Delete Specified Element  
4. Delete Last  
5. Display  
6. Exit  
Enter your choice: 3  
Enter the value to delete: 30  
Element 30 deleted successfully.  
  
Singly Linked List Operations:  
1. Insert at End  
2. Delete First  
3. Delete Specified Element  
4. Delete Last  
5. Display  
6. Exit  
Enter your choice: 4  
Last element deleted successfully.  
  
Singly Linked List Operations:  
1. Insert at End  
2. Delete First  
3. Delete Specified Element  
4. Delete Last  
5. Display  
6. Exit  
Enter your choice: 5  
Linked list elements: 20
```

```
    psept->next = pto->next;  
    free(pto)  
    return (start);  
};
```

Output :

\*\*\* main menu \*\*\*

- 1. create linked list
- 2. display
- 3. delete\_beg
- 4. delete\_end
- 5. delete\_node

Enter your choice : 1

Enter -1 to end

Enter the data : 10

Enter the data : 20

Enter the data : 30

Enter the data : 40

Enter the data : 1

→ Enter your choice : 2

10 20 30 40

→ Enter your choice : 3

→ Enter your choice : 4

→ Enter your choice : 5

20 30

→ Enter your choice : 5

Enter value to be deleted : 20

26.01.24

## 7. SLL-sort,reverse,concatination.

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

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

struct node *head=NULL;

void create()
{
    struct node *new_node,*ptr;
    int num;
    printf("\n enter -1 to end\n");
    printf("enter data: \n");
    scanf("%d",&num);
    while(num!=-1)
    {
        new_node=(struct node*)malloc(sizeof(struct node));
        new_node->data=num;
        if(head==NULL)
        {
            new_node->next=NULL;
            head=new_node;
        }
        else
        {
            ptr=head;

```

```
while(ptr->next!=NULL)
{
    ptr=ptr->next;
    new_node->next=NULL;
    ptr->next=new_node;
}
printf("enter data:");
scanf("%d",&num);
}
```

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

```
void sort()
{
    struct node *ptr1,*ptr2;
    int temp;
    ptr1=head;
    while(ptr1->next!=NULL)
    {
```

```

ptr2=ptr1->next;
while(ptr2!=NULL)
{
    if(ptr1->data > ptr2->data)
    {
        temp=ptr1->data;
        ptr1->data=ptr2->data;
        ptr2->data=temp;
    }
    ptr2=ptr2->next;
}
ptr1=ptr1->next;
}

void con()
{
    struct node *new_node,*h1,*h2,*ptr;
    int i,n,m;
    printf("enter no. of elements in 1st list:\n");
    scanf("%d",&n);
    h1=NULL;
    for(i=0;i<n;i++)
    {
        printf("enter data:\n");
        new_node=malloc(sizeof(struct node));
        scanf("%d",&new_node->data);
        new_node->next=NULL;
        if(h1==NULL)

```

```

{
    h1=new_node;
}
else
{
    ptr=h1;
    while(ptr->next!=NULL)
    {
        ptr=ptr->next;
    }
    ptr->next=new_node;
}
printf("enter no. of elements in 2nd list:\n");
scanf("%d",&m);
h2=NULL;
for(i=0;i<m;i++)
{
    printf("enter data:\n");
    new_node=malloc(sizeof(struct node));
    scanf("%d",&new_node->data);
    new_node->next=NULL;
    if(h2==NULL)
    {
        h2=new_node;
    }
    else
    {
        ptr=h2;

```

```

        while(ptr->next!=NULL)
    {
        ptr=ptr->next;
    }
    ptr->next=new_node;
}
}

concatenate(h1,h2);
}

void concatenate(struct node *h1,struct node *h2)
{
    struct node *ptr;
    head=h1;
    ptr=head;
    if(h1==NULL && h2==NULL)
    {
        printf("list is empty!");
    }
    else
    {
        while(ptr->next!=NULL)
        {
            ptr=ptr->next;
        }
        ptr->next=h2;
        display();
    }
}

```

```

void reverse()
{
    struct node *current , *ptr , *temp;
    ptr=NULL;
    current=head;
    while(current!=NULL)
    {
        temp=current->next;
        current->next=ptr;
        ptr=current;
        current=temp;
    }
    head=ptr;
    display();
}

void main()
{
    int choice;
    while(1)
    {
        printf("\n Enter \n 1:create \n 2:sort \n 3:concatenate \n 4:reverse \n 5:exit");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:
                create();
                break;
        }
    }
}

```

```
case 2:  
    sort();  
    display();  
    break;  
  
case 3:  
    con();  
    break;  
  
case 4:  
    reverse();  
    break;  
  
case 5:  
    exit(0);  
  
default:  
    printf("invalid input");  
}  
}  
}
```

```
1:create  
2:sort  
3:concatenate  
4:reverse  
5:exit  
Enter your choice : 1  
  
enter -1 to end  
enter data:  
10  
enter data:15  
enter data:5  
enter data:45  
enter data:30  
enter data:20  
enter data:-1  
  
1:create  
2:sort  
3:concatenate  
4:reverse  
5:exit  
Enter your choice : 2  
5  
10  
15  
20  
30  
45
```

```
1:create  
2:sort  
3:concatenate  
4:reverse  
5:exit  
Enter your choice : 3  
enter no. of elements in 1st list:  
3  
enter data:  
5  
enter data:  
10  
enter data:  
15  
enter no. of elements in 2nd list:  
3  
enter data:  
20  
enter data:  
30  
enter data:  
45  
5  
10  
15  
20  
30  
45
```

```
1:create  
2:sort  
3:concatenate  
4:reverse  
5:exit  
Enter your choice : 4  
45  
30  
20  
15  
10  
5
```

→ output for program 2 (concatenation, reverse, sort)

1. Insert into List 1
2. Insert into List 2
3. Sort List 1
4. Reverse List 2
5. Concatenate lists
6. Print lists
7. Exit

→ Enter your choice : 1

Enter data to insert (list 1) : 10

→ Enter your choice : 2

Enter data to insert (list 2) : 20

→ Enter your choice : 3

List 1 sorted

→ Enter your choice : 1

Enter data to insert (list 2) : 6

→ Enter your choice : 3

List 1 sorted

### **8a. Stack implementation using single linked list**

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

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

// Function to create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed.\n");
        exit(EXIT_FAILURE);
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// Function to push an element onto the stack
struct Node* push(struct Node* top, int data) {
    struct Node* newNode = createNode(data);
    newNode->next = top;
    return newNode;
}
```

```

// Function to pop an element from the stack
struct Node* pop(struct Node* top) {
    if (top == NULL) {
        printf("Stack underflow. Cannot pop.\n");
        return NULL;
    }

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

// Function to display the elements of the stack
void displayStack(struct Node* top) {
    printf("Stack: ");
    while (top != NULL) {
        printf("%d ", top->data);
        top = top->next;
    }
    printf("\n");
}

// Function to free the memory allocated for the stack
void freeStack(struct Node* top) {
    while (top != NULL) {
        struct Node* temp = top;
        top = top->next;

```

```
        free(temp);
    }
}

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

    do {
        printf("\nMenu:\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 data to push: ");
                scanf("%d", &data);
                top = push(top, data);
                break;

            case 2:
                top = pop(top);
                break;
        }
    }
}
```

```
case 3:  
    displayStack(top);  
    break;  
  
case 4:  
    printf("Exiting the program.\n");  
    break;  
  
default:  
    printf("Invalid choice! Please enter a valid option.\n");  
}  
  
} while (choice != 4);  
  
// Free the memory allocated for the stack before exiting  
freeStack(top);  
  
return 0;  
}
```

```
Menu:  
1. Push  
2. Pop  
3. Display  
4. Exit  
Enter your choice: 1  
Enter data to push: 5
```

```
Menu:  
1. Push  
2. Pop  
3. Display  
4. Exit  
Enter your choice: 1  
Enter data to push: 10
```

```
Menu:  
1. Push  
2. Pop  
3. Display  
4. Exit  
Enter your choice: 1  
Enter data to push: 15
```

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

```
Menu:  
1. Push  
2. Pop  
3. Display  
4. Exit  
Enter your choice: 3  
Stack: 10 5
```

→ Enter your choice : 6

List1 : 6 10

List2 : 20

→ Enter your choice : 5

Lists concatenated

→ Enter your choice : 6

List1 : 6 10 20

2(a) Output for stack implementation using ls

Menu

1. PUSH

2. POP

3. DISPLAY

4. EXIT

→ Enter your choice : 1

Enter data to push : 60

→ Enter your choice : 1

Enter data to push : 70

## 8b. Queue implementation using single linked list

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

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

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

void enqueue();
void dequeue();
void display();

int main()
{
    int choice;
    while(1)
    {
        printf("\n Enter \n 1:enqueue \n 2:dequeue \n 3:display \n 4:exit \n");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:
                enqueue();
                printf("the queue is:");
                display();
                break;
            case 2:
```

```

        dequeue();

        printf("\n the queue is:");

        display();

        break;

    case 3:

        display();

        break;

    case 4:

        exit(0);

    default:

        printf("Invalid Input!");

    }

}

return 0;
}

```

```

void enqueue()

{
    int val;

    printf("enter data to be inserted:\n");

    scanf("%d",&val);

    struct node *temp;

    temp=(struct node *)malloc(sizeof(struct node));

    temp->data=val;

    temp->next=NULL;

    if(front==NULL && rear==NULL)

    {
        front=rear=temp;
    }
}

```

```

else
{
    rear->next=temp;
    rear=temp;
}
}

void dequeue()
{
    struct node *temp;
    temp=front;
    front=front->next;
    printf("the removed data is: %d ",temp->data);
    free(temp);
}

void display()
{
    struct node *temp;
    temp=front;
    if(front==NULL)
    {
        printf("Queue is empty!");
    }
    else
    {
        while(temp!=NULL)
        {
            printf("%d \n",temp->data);
        }
    }
}

```

```
temp=temp->next;  
}  
}  
}
```

```
Enter  
1:enqueue  
2:dequeue  
3:display  
4:exit  
1  
enter data to be inserted:  
10  
the queue is:10  
  
Enter  
1:enqueue  
2:dequeue  
3:display  
4:exit  
1  
enter data to be inserted:  
20  
the queue is:10  
20
```

```
Enter
1:enqueue
2:dequeue
3:display
4:exit
1
enter data to be inserted:
30
the queue is:10
20
30

Enter
1:enqueue
2:dequeue
3:display
4:exit
2
the removed data is: 10
the queue is:20
30

Enter
1:enqueue
2:dequeue
3:display
4:exit
3
20
30
```

→ Enter your choice : 2  
popped successfully

→ Enter your choice : 3  
60

## 2. by output for queue implementation using u

Main menu

1. Enqueue
2. Dequeue
3. Display
4. Exit

→ Enter your choice : 1  
Enter data to enqueue : 10

→ Enter your choice : 1  
Enter data to enqueue : 20

→ Enter your choice : 3  
10 20

→ Enter your choice : 2

→ Enter your choice : 3

*Ans 30 01 24* 20

## **9. WAP to Implement doubly link list with primitive operations**

- a) Create a doubly linked list.**
- b) Insert a new node to the left of the node.**
- c) Delete the node based on a specific value**

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

struct node
{
    struct node *prev;
    int data;
    struct node *next;
}*head=NULL;

void display();
void create();
void insert_left();
void delete_node();

struct node *new_node,*ptr;
void main()
{
    int choice;
    printf("1:create \n 2:insert_left \n 3:delete_node\n 4:exit");
    while(1)
    {
        printf("\nEnter choice: \t");
        scanf("%d",&choice);
        switch(choice)
```

```

{
    case 1:
        create();
        break;

    case 2:
        insert_left();
        break;

    case 3:
        delete_node();
        break;

    case 4:
        exit(0);

    default:
        printf("invalid input");
}
}
}

```

```

void create()
{
    int val;
    printf("\nenter -1 to end");
    printf("\nenter data:\t");
    scanf("%d",&val);
    while(val!=-1)
    {
        new_node=malloc(sizeof(struct node));
        new_node->data=val;
        if(head==NULL)

```

```

{
    new_node->prev=NULL;
    new_node->next=NULL;
    head=new_node;
}

else
{
    ptr=head;
    while(ptr->next!=NULL)
        ptr=ptr->next;
    ptr->next=new_node;
    new_node->prev=ptr;
    new_node->next = NULL;
}

printf("\n enter data:\t");
scanf("%d",&val);

}
display();
}

void display()
{
if(head==NULL)
    printf("the list is empty");
else
{
    ptr=head;
    printf("\n The list is:\n");
    while(ptr!=NULL)

```

```

    {
        printf("%d \t",ptr->data);
        ptr=ptr->next;
    }
}

void insert_left()
{
    int val,d;
    printf("enter value to be inserted:");
    scanf("%d",&val);
    printf("enter value before which data is to be inserted:");
    scanf("%d",&d);
    new_node=malloc(sizeof(struct node));
    new_node->data=val;
    if(head==NULL)
    {
        new_node->prev=NULL;
        new_node->next=NULL;
        head=new_node;
    }
    else
    {
        ptr=head;
        while(ptr->data!=d)
            ptr=ptr->next;
        if(ptr==head)
        {

```

```

    new_node->prev=NULL;
    new_node->next=head;
    head->prev=new_node;
    head=new_node;
}
else
{
    new_node->next=ptr;
    new_node->prev=ptr->prev;
    ptr->prev->next=new_node;
    ptr->prev=new_node;
}
}
display();
}

```

```

void delete_node()
{
    if(head==NULL)
        printf("\n the list is empty");
    else
    {
        int val;
        printf("enter value to be deleted:\n");
        scanf("%d",&val);
        ptr=head;
        while(ptr->data!=val)
            ptr=ptr->next;
        if(ptr==head)

```

```
{  
    head=head->next;  
    head->prev=NULL;  
    free(ptr);  
}  
else  
{  
    ptr->prev->next=ptr->next;  
    ptr->next->prev=ptr->prev;  
    free(ptr);  
}  
}  
display();  
}
```

```
1:create
2:insert_left
3:delete_node
4:exit
enter choice: 1

enter -1 to end
enter data: 5

enter data: 10

enter data: 15

enter data: 20

enter data: -1

The list is:
5      10      15      20
enter choice: 2
enter value to be inserted:4
enter value before which data is to be inserted:10

The list is:
5      4      10      15      20
enter choice: 3
enter value to be deleted:
15

The list is:
5      4      10      20
enter choice: 4

Process returned 0 (0x0)  execution time : 76.304 s
Press any key to continue.
```

1. create  
2. Insert\_left  
3. delete\_node  
4. exit

Enter your choice : 1

Enter -1 to end

Enter data : 5

Enter data = 10

~~Enter data : 15~~

~~Enter data : 20~~

Enter data : -1

The list is : 5 10 15 20

Enter choice : 2

Enter value to be inserted : 4

Enter value before which data is to be inserted : 10

The list is :

5 4 10 15 20

## **10. Write a program.**

- a. To construct Binary Search tree**
- b. Traverse the tree using inorder , postorder, preorder.**
- c. Display the elements in the tree.**

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

struct node
{
    int data;
    struct node *left,*right;
}*root;

struct node *create()
{
    struct node *temp;
    int val;
    printf("enter data:");
    scanf("%d",&val);
    temp = malloc(sizeof(struct node));
    temp->data = val;
    temp->left=temp->right=NULL;
    return temp;
};

void insert(struct node *root,struct node *temp)
{
    if(temp->data<root->data)
    {
        if(root->left!=NULL)
```

```

    insert(root->left,temp);

else
    root->left=temp;
}

if(temp->data>root->data)
{
    if(root->right!=NULL)
        insert(root->right,temp);

else
    root->right=temp;
}

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

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

```

```

}

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

void main()
{
    int n,choice;
    struct node *temp;
    root=NULL;
    printf("enter no. of elements in tree");
    scanf("%d",&n);
    for(int i=0;i<n;i++)
    {
        temp=create();
        if(root==NULL)
            root=temp;
        else
            insert(root,temp);
    }
    printf("\n1:inorder \n2:preorder \n3:postorder \n4:exit");
    while(1)
    {
        printf("\nEnter choice:");

```

```
scanf("%d",&choice);
switch(choice)
{
    case 1:
        printf("Inorder:");
        inorder(root);
        break;
    case 2:
        printf("preorder:");
        preorder(root);
        break;
    case 3:
        printf("postorder:");
        postorder(root);
    case 4:
        exit(0);
    default:
        printf("INVALID INPUT");
}
}
```

```
enter no. of elements in tree : 5
enter data:10
enter data:20
enter data:30
enter data:40
enter data:50

1:inorder
2:preorder
3:postorder
4:exit
enter choice : 1
Inorder:10      20      30      40      50
enter choice : 2
preorder : 10    20      30      40      50
enter choice : 3
postorder : 50   40      30      20      10
Process returned 0 (0x0)  execution time : 19.720 s
Press any key to continue.
```

Output

insertion successful

8 10 12 16 20 25 30

10 8 20 12 16 30 25

8 16 12 25 30 20 10

~~Stu~~  
20.02.24

## 11. BFS

```
#include<stdio.h>

void bfs (int a[10] [10], int n, int u)

{
    int f, r,q[10], v;
    int s[10]={0};

    printf("the nodes visited from %d:",u);
    f=0;
    r=-1;
    q[++r]=u;
    s[u]=1;
    printf("%d", u);
    while(f<=r)
    {
        u=q[f++];
        for(v=0;v<n;v++)
        {
            if(a[u][v])
            {
                if(s[v]==0)
                {
                    printf("%d", v);
                    s[v]=1;
                    q[++r]=v;
                }
            }
        }
    }
}
```

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

int main()
{
    int n, a[10][10], source, i, j;
    printf("\nEnter no of nodes:");
    scanf("%d",&n);
    printf("\nEnter the adjacency matrix:");
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            scanf("%d",&a[i][j]);
        }
    }
    for(source=0;source<n;source++)
        bfs (a, n, source);
    return 0;
}
```

```
enter no of nodes:7

Enter the adjacency matrix:0 1 1 1 1 0 0
1 0 0 1 0 1 0
1 0 0 0 0 0 1
1 1 0 0 0 1 0
1 0 0 0 0 0 1
0 1 0 1 0 0 0
0 0 1 0 1 0 0
the nodes visited from 0:0123456
the nodes visited from 1:1035246
the nodes visited from 2:2061345
the nodes visited from 3:3015246
the nodes visited from 4:4061235
the nodes visited from 5:5130246
the nodes visited from 6:6240135

Process returned 0 (0x0)    execution time : 84.688 s
Press any key to continue.
```



Output for program 1 (BFS)

Enter no of nodes : 7

Enter the adjacency matrix : 0 1 1 1 1 0 0

1 0 0 1 0 1 0

1 0 0 0 0 0 1

1 1 0 0 0 1 0

1 0 0 0 0 0 1

0 1 0 1 0 0 0

0 0 1 0 1 0 0

The nodes visited from 0 : 0 1 2 3 4 5 6

The nodes visited from 1 : 1 0 3 5 2 4 6

2 : 2 0 6 1 3 4 5

3 : 3 0 1 5 2 4 6

4 : 4 0 6 1 2 3 5

5 : 5 1 3 0 2 4 6

6 : 6 2 4 0 1 3 5

Output for program 2 (DFS)

Enter number of nodes :

4

Enter the adjacency matrix

0 1 0 0

0 0 1 0

0 0 0 1

1 0 0 0

Graph is connected

This  
26.02.24

## 12. DFS

```
#include<stdio.h>
#include<conio.h>
int a[1][10];
void dfs(int n, int cost[10][10], int u, int s[])
{
    int v;
    s[u]=1;
    for(v=0;v<n;v++)
    {
        if((cost[u][v]==1) && (s[v]==0))
            dfs(n,cost,v,s);
    }
}

void main()
{
    int n,i,j,cost[10][10],s[10],con,flag;

    printf("Enter the number of nodes\n");
    scanf("%d", &n);
    printf("Enter the adjacency matrix\n");
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        scanf("%d", &cost[i][j]);
```

```

}

con=0;

for(j=0;j<n;j++)
{
    for(i=0;i<n;i++)
        s[i]=0;

    dfs(n,cost,j,s);

    flag=0;

    for(i=0;i<n;i++)
    {
        if(s[i]==0)
            flag=1;
    }

    if(flag==0)
        con=1;
}

if(con==1)
printf("Graph is connected\n");

else
printf("Graph is not connected\n");

getch();
}

```

```

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

```



Output for program 1 (BFS)

Enter no of nodes : 7

Enter the adjacency matrix : 0 1 1 1 1 0 0

1 0 0 1 0 1 0

1 0 0 0 0 0 1

1 1 0 0 0 1 0

1 0 0 0 0 0 1

0 1 0 1 0 0 0

0 0 1 0 1 0 0

The nodes visited from 0 : 0 1 2 3 4 5 6

The nodes visited from 1 : 1 0 3 5 2 4 6

2 : 2 0 6 1 3 4 5

3 : 3 0 1 5 2 4 6

4 : 4 0 6 1 2 3 5

5 : 5 1 3 0 2 4 6

6 : 6 2 4 0 1 3 5

Output for program 2 (DFS)

Enter number of nodes :

4

Enter the adjacency matrix

0 1 0 0

0 0 1 0

0 0 0 1

1 0 0 0

Graph is connected

This  
26.02.24

## 13. Leetcode Programs

### 1. Score of parentheses

```
1 int scoreOfParentheses(char *s) {  
2     int stack[50];  
3     int top = -1;  
4     int score = 0;  
5  
6     for (int i = 0; s[i] != '\0'; i++) {  
7         if (s[i] == '(') {  
8             stack[++top] = score;  
9             score = 0;  
10        } else {  
11            score = stack[top--] + (score == 0 ? 1 : 2 * score);  
12        }  
13    }  
14  
15    return score;  
16 }
```

Testcase |  Test Result

**Accepted** Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

```
s =  
"()"
```

Output

```
1
```

Expected

```
1
```

Testcase | [Test Result](#)

**Accepted** Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

```
s =  
"((())"
```

Output

```
2
```

Expected

```
2
```

**Accepted** Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

```
s =  
"(()()"
```

Output

```
2
```

Expected

```
2
```

## 2. Delete the middle node of the linked list

```
/*
struct ListNode* deleteMiddle(struct ListNode* head) {
    struct ListNode* ptr=head;
    struct ListNode* preptr=NULL;
    int count=0;
    if(head==NULL){
        return head;
    }
    else if(head->next==NULL){
        free(head);
        return NULL;
    }
    else{
        while(ptr!=NULL){
            count++;
            ptr=ptr->next;
        }
        int n=0;
        ptr=head;
        while(n!=count/2){
            preptr=ptr;
            ptr=ptr->next;
            n++;
        }
        preptr->next=ptr->next;
        free(ptr);
    }
    return head;
}
```

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

```
head =
[1,3,4,7,1,2,6]
```

Output

```
[1,3,4,1,2,6]
```

Expected

```
[1,3,4,1,2,6]
```

**Accepted** Runtime: 0 ms

- Case 1
- Case 2
- Case 3

Input

```
head =  
[1,2,3,4]
```

Output

```
[1,2,4]
```

Expected

```
[1,2,4]
```

**Accepted** Runtime: 0 ms

- Case 1
- Case 2
- Case 3

Input

```
head =  
[2,1]
```

Output

```
[2]
```

Expected

```
[2]
```

### 3. Odd even linked list

```
struct ListNode* oddEvenList(struct ListNode* head) {
    if (head == NULL || head->next == NULL) {
        return head;
    }

    struct ListNode *odd = head;
    struct ListNode *even = head->next;
    struct ListNode *even_head = even;

    while (even != NULL && even->next != NULL) {
        odd->next = even->next;
        odd = odd->next;
        even->next = odd->next;
        even = even->next;
    }

    odd->next = even_head;

    return head;
}
```

Accepted Runtime: 5 ms

• Case 1 • Case 2

Input

```
head =
[1,2,3,4,5]
```

Output

```
[1,3,5,2,4]
```

Expected

```
[1,3,5,2,4]
```

**Accepted** Runtime: 5 ms

- Case 1
- Case 2

Input

```
head =  
[2,1,3,5,6,4,7]
```

Output

```
[2,3,6,7,1,5,4]
```

Expected

```
[2,3,6,7,1,5,4]
```

#### 4.Find bottom left tree

```
int findBottomLeftValue(struct TreeNode* root) {  
    if (root == NULL) {  
        return -1;  
    }  
  
    struct TreeNode** queue = (struct TreeNode**)malloc(pow(10, 4) * sizeof(struct TreeNode*));  
    int front = 0, rear = 0;  
    int leftmostValue = 0;  
  
    queue[rear++] = root;  
  
    while (front < rear) {  
        int levelSize = rear - front;  
  
        for (int i = 0; i < levelSize; i++) {  
            struct TreeNode* currentNode = queue[front++];  
  
            if (i == 0) {  
                leftmostValue = currentNode->val;  
            }  
  
            if (currentNode->left) {  
                queue[rear++] = currentNode->left;  
            }  
  
            if (currentNode->right) {  
                queue[rear++] = currentNode->right;  
            }  
        }  
        free(queue);  
        return leftmostValue;  
    }  
}
```

**Accepted** Runtime: 0 ms

• Case 1 • Case 2

Input

```
root =  
[2,1,3]
```

Output

```
1
```

Expected

```
1
```

**Accepted** Runtime: 0 ms

• Case 1 • Case 2

Input

```
root =  
[1,2,3,4,null,5,6,null,null,7]
```

Output

```
7
```

Expected

```
7
```

## 5. Delete node in BST

```
struct TreeNode* findMin(struct TreeNode* node) {
    while (node->left != NULL) {
        node = node->left;
    }
    return node;
}

struct TreeNode* deleteNode(struct TreeNode* root, int key) {
    if (root == NULL) {
        return root;
    } else if (key < root->val) {
        root->left = deleteNode(root->left, key);
    } else if (key > root->val) {
        root->right = deleteNode(root->right, key);
    } else {
        if (root->left == NULL && root->right == NULL) {
            free(root);
            root = NULL;
        } else if (root->left == NULL) {
            struct TreeNode *ptr = root;
            root = root->right;
            free(ptr);
        } else if (root->right == NULL) {
            struct TreeNode *ptr = root;
            root = root->left;
            free(ptr);
        } else {
            struct TreeNode *temp = findMin(root->right);
            root->val = temp->val;
            root->right = deleteNode(root->right, temp->val);
        }
    }
    return root;
}
```

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

```
root =
[5,3,6,2,4,null,7]
```

key =

3

Output

```
[5,4,6,2,null,null,7]
```

Expected

```
[5,4,6,2,null,null,7]
```

**Accepted** Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

```
root =  
[5,3,6,2,4,null,7]
```

key =

0

Output

```
[5,3,6,2,4,null,7]
```

Expected

```
[5,3,6,2,4,null,7]
```

**Accepted** Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

```
root =  
[]
```

key =

0

Output

```
[]
```

Expected

```
[]
```