VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

ROSHAN N (1BM23CS275)

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

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

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

This is to certify that the Lab work entitled **“DATA STRUCTURES”** carried out by ROSHAN N **(1BM23CS275)**, who is bonafide student of **B. M. S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2024-25. The Lab report has been approved as it satisfies the academic requirements in respect of Data structures Lab - **(23CS3PCDST)**work prescribed for the said degree.

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

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

Index Sheet

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 MAX 100


typedef struct Stack {

    int items[MAX];

    int top;

} Stack;


void initStack(Stack* s) {

    s->top = -1;

}


int isFull(Stack* s) {

    return s->top == MAX - 1;

}


int isEmpty(Stack* s) {

    return s->top == -1;

}


void push(Stack* s, int value) {

    if (isFull(s)) {
```

```
        printf("Stack Overflow: Cannot push %d, stack is full.\n", value);
        return;
    }
    s->items[++(s->top)] = value;
    printf("%d pushed to stack.\n", value);
}

int pop(Stack* s) {
    if (isEmpty(s)) {
        printf("Stack Underflow: Cannot pop, stack is empty.\n");
        return -1;
    }
    return s->items[(s->top)--];
}

void display(Stack* s) {
    if (isEmpty(s)) {
        printf("Stack is empty.\n");
        return;
    }
    printf("Stack elements: ");
    for (int i = 0; i <= s->top; i++) {
        printf("%d ", s->items[i]);
    }
    printf("\n");
}

int main() {
    Stack s;
```

```
initStack(&s);

int choice, value;

while (1) {
    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 value to push: ");
            scanf("%d", &value);
            push(&s, value);
            break;
        case 2:
            value = pop(&s);
            if (value != -1) {
                printf("%d popped from stack.\n", value);
            }
            break;
        case 3:
            display(&s);
            break;
        case 4:
```

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

Output

```
1.push
2.pop
3.display
4.exit
Enter your choice : 1
Enter element to push : 2
1.push
2.pop
3.display
4.exit
Enter your choice : 1
Enter element to push : 4
1.push
2.pop
3.display
4.exit
Enter your choice : 1
Enter element to push : 6
1.push
2.pop
3.display
4.exit
Enter your choice : 1
Enter element to push : 8
1.push
2.pop
3.display
4.exit
Enter your choice : 1
Enter element to push : 9
```

```
Enter element to push : 9
1.push
2.pop
3.display
4.exit
Enter your choice : 1
Enter element to push : 7
Stack overflowing unable to push 71.push
2.pop
3.display
4.exit
Enter your choice : 2
 9 poped from stack1.push
2.pop
3.display
4.exit
Enter your choice : 2
 9 poped from stack1.push
2.pop
3.display
4.exit
Enter your choice : 3
Stack element :9 8 6 4 2 1.push
2.pop
3.display
4.exit
Enter your choice : 4

Process returned 0 (0x0)   execution time : 31.628 s
Press any key to continue.
```

lab program 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>


#define MAX 100


char stack[MAX];
```

```
int top = -1;

void push(char x) {
   if (top == MAX - 1) {
      printf("Stack Overflow\n");
   } else {
      stack[++top] = x;
   }
}

char pop() {
   if (top == -1) {
      printf("Stack Underflow\n");
      return -1;
   } else {
      return stack[top--];
   }
}

int precedence(char x) {
   if (x == '^') {
      return 3;
   } else if (x == '*' || x == '/') {
      return 2;
   } else if (x == '+' || x == '-') {
      return 1;
   } else {
      return 0;
   }
}

int isOperand(char ch) {
```

```
    if ((ch >= 'A' && ch <= 'Z') || (ch >= 'a' && ch <= 'z') || (ch >= '0' && ch <= '9')) {
        return 1;
    }
    return 0;
}

void infixToPostfix(char* exp) {
    char postfix[MAX];
    int i, j = 0;

    for (i = 0; exp[i] != '\0'; i++) {
        char ch = exp[i];

        if (isOperand(ch)) {
            postfix[j++] = ch;
        }
        else if (ch == '(') {
            push(ch);
        }
        else if (ch == ')') {
            while (top != -1 && stack[top] != '(') {
                postfix[j++] = pop();
            }
            pop();
        }
        else {
            while (top != -1 && precedence(stack[top]) >= precedence(ch)) {
                postfix[j++] = pop();
            }
            push(ch);
        }
    }
```

```
    while (top != -1) {
        postfix[j++] = pop();
    }

    postfix[j] = '\0';
    printf("Postfix Expression: %s\n", postfix);
}

int main() {
    char infix[MAX];
    printf("Enter infix expression: ");
    scanf("%s", infix);
    infixToPostfix(infix);
    return 0;
}
```

OUTPUT

```
Enter infix expression: A+B*(C+D)-E/F
Postfix Expression: ABCD+*+EF/-

Process returned 0 (0x0)   execution time : 6.606 s
Press any key to continue.
```

Lab program 3:

WAP to simulate the working of a queue of integers using an array. Provide the following operations: Insert,Delete, Display The program should print appropriate messages for queue empty and queue overflow conditions

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

#define MAX 5

struct Queue {
```

```
    int arr[MAX];
    int front;
    int rear;
};

void initializeQueue(struct Queue* q) {
    q->front = -1;
    q->rear = -1;
}

int isFull(struct Queue* q) {
    if (q->rear == MAX - 1) {
        return 1;
    }
    return 0;
}

int isEmpty(struct Queue* q) {
    if (q->front == -1) {
        return 1;
    }
    return 0;
}

void insert(struct Queue* q, int value) {
    if (isFull(q)) {
        printf("Queue Overflow! Cannot insert %d\n", value);
    } else {
        if (q->front == -1) {
            q->front = 0;
        }
        q->rear++;
```

```
        q->arr[q->rear] = value;
        printf("Inserted %d into the queue.\n", value);
    }
}

void delete(struct Queue* q) {
    if (isEmpty(q)) {
        printf("Queue Underflow! Cannot delete element\n");
    } else {
        printf("Deleted %d from the queue.\n", q->arr[q->front]);
        if (q->front == q->rear) {
            q->front = q->rear = -1;
        } else {
            q->front++;
        }
    }
}

void display(struct Queue* q) {
    if (isEmpty(q)) {
        printf("Queue is empty. Nothing to display.\n");
    } else {
        printf("Queue elements: ");
        for (int i = q->front; i <= q->rear; i++) {
            printf("%d ", q->arr[i]);
        }
        printf("\n");
    }
}

int main() {
    struct Queue q;
```

```
initializeQueue(&q);

int choice, value;

while (1) {
   printf("\nQueue Operations Menu:\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 value to insert: ");
         scanf("%d", &value);
         insert(&q, value);
         break;

      case 2:
         delete(&q);
         break;

      case 3:
         display(&q);
         break;

      case 4:
         printf("Exiting program.\n");
         exit(0);
```

```
            default:
                printf("Invalid choice! Please try again.\n");
        }
    }

    return 0;

}
```

OUTPUT:

```
Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to insert: 6
Inserted 6 into the queue

Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to insert: 8
Inserted 8 into the queue

Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to insert: 9
Inserted 9 into the queue

Queue Operations:
1. Insert
```

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

Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to insert: 7
Inserted 7 into the queue

Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to insert: 3
Queue Overflow

Queue Operations:
1. Insert
```

```
Enter your choice: 1
Enter the value to insert: 3
Queue Overflow

Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Deleted 6 from the queue

Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
Queue elements: 8 9 4 7

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

Process returned 0 (0x0)   execution time : 26.746 s
Press any key to continue.
```

Lab program 4:

WAP to simulate the working of a circular queue of integers using an array. Provide the following operations:Insert, Delete & Display The program should print appropriate messages for queue empty and queue overflow conditions

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

#define MAX 5
struct Queue {
   int arr[MAX];
   int front;
   int rear;
};

void initializeQueue(struct Queue* q) {
   q->front = -1;
   q->rear = -1;
}

int isFull(struct Queue* q) {
   if ((q->rear + 1) % MAX == q->front) {
      return 1;
   }
   return 0;
}

int isEmpty(struct Queue* q) {
   if (q->front == -1) {
      return 1;
   }
   return 0;
}
```

```
void insert(struct Queue* q, int value) {
    if (isFull(q)) {
        printf("Queue Overflow! Cannot insert %d\n", value);
    } else {
        if (q->front == -1) {
            q->front = 0;
        }
        q->rear = (q->rear + 1) % MAX;
        q->arr[q->rear] = value;
        printf("Inserted %d into the queue.\n", value);
    }
}

void delete(struct Queue* q) {
    if (isEmpty(q)) {
        printf("Queue Underflow! Cannot delete element\n");
    } else {
        printf("Deleted %d from the queue.\n", q->arr[q->front]);
        if (q->front == q->rear) {
            q->front = q->rear = -1;
        } else {
            q->front = (q->front + 1) % MAX;
        }
    }
}

void display(struct Queue* q) {
    if (isEmpty(q)) {
        printf("Queue is empty. Nothing to display.\n");
    } else {
        printf("Queue elements: ");
        int i = q->front;
```

```
        while (i != q->rear) {
            printf("%d ", q->arr[i]);
            i = (i + 1) % MAX;
        }
        printf("%d\n", q->arr[q->rear]);
    }
}

int main() {
    struct Queue q;
    initializeQueue(&q);

    int choice, value;

    while (1) {
        printf("\nCircular Queue Operations Menu:\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 value to insert: ");
                scanf("%d", &value);
                insert(&q, value);
                break;

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

```
            break;

        case 3:
            display(&q);
            break;

        case 4:
            printf("Exiting program.\n");
            exit(0);

        default:
            printf("Invalid choice! Please try again.\n");
        }
    }

    return 0;
}
```

OUTPUT:

```
--- Circular Queue Operations ---
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to insert: 2
Inserted 2 into the queue.

--- Circular Queue Operations ---
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to insert: 3
Inserted 3 into the queue.

--- Circular Queue Operations ---
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to insert: 5
Inserted 5 into the queue.

--- Circular Queue Operations ---
1. Insert
```

```
--- Circular Queue Operations ---
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to insert: 7
Inserted 7 into the queue.

--- Circular Queue Operations ---
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to insert: 8
Inserted 8 into the queue.

--- Circular Queue Operations ---
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to insert: 9
Queue Overflow! Cannot insert 9

--- Circular Queue Operations ---
1. Insert
```

```
--- Circular Queue Operations ---
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Deleted 2 from the queue.

--- Circular Queue Operations ---
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Deleted 3 from the queue.

--- Circular Queue Operations ---
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
Queue elements are: 5 7 8

--- Circular Queue Operations ---
1. Insert
2. Delete
3. Display
4. Exit
```

Lab program 5:

WAP of insertion in linked list:

```
#include <stdio.h>

#include <stdlib.h>


struct Node {

   int data;

   struct Node *next;

};

struct Node *top = NULL;

struct Node* createNode(int value) {

   struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));
```

```
    if (!newNode) {
        printf("Memory allocation failed.\n");
        exit(1);
    }
    newNode->data = value;
    newNode->next = NULL;
    return newNode;
}
void insertAtBeginning(int value) {
    struct Node *newNode = createNode(value);
    newNode->next = top;
    top = newNode;
    printf("Inserted %d at the beginning.\n", value);
}
void insertAtEnd(int value) {
    struct Node *newNode = createNode(value);
    if (top == NULL) {
        top = newNode;
        printf("Inserted %d at the end.\n", value);
        return;
    }
    struct Node *ptr = top;
    while (ptr->next != NULL) {
        ptr = ptr->next;
    }
    ptr->next = newNode;
    printf("Inserted %d at the end.\n", value);
}
void insertAfterPosition(int position, int value) {
    struct Node *newNode = createNode(value);
    struct Node *ptr = top;
    for (int i = 0; i < position; i++) {
```

```
        if (ptr == NULL) {
            printf("Position %d is out of bounds.\n", position);
            free(newNode);
            return;
        }
        ptr = ptr->next;
    }
    newNode->next = ptr->next;
    ptr->next = newNode;
    printf("Inserted %d after position %d.\n", value, position);
}
void displayList() {
    if (top == NULL) {
        printf("The list is empty.\n");
        return;
    }
    struct Node *ptr = top;
    printf("Linked list: ");
    while (ptr != NULL) {
        printf("%d -> ", ptr->data);
        ptr = ptr->next;
    }
    printf("NULL\n");
}


int main() {
    int choice, value, position;
    while (1) {
        printf("\nMenu:\n1. Insert at beginning\n2. Insert at end\n3. Insert after position\n4. Display
list\n5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
```

```
        switch (choice) {
          case 1:
            printf("Enter value to insert at beginning: ");
            scanf("%d", &value);
            insertAtBeginning(value);
            break;
          case 2:
            printf("Enter value to insert at end: ");
            scanf("%d", &value);
            insertAtEnd(value);
            break;
          case 3:
            printf("Enter position after which to insert: ");
            scanf("%d", &position);
            printf("Enter value to insert: ");
            scanf("%d", &value);
            insertAfterPosition(position, value);
            break;
          case 4:
            displayList();
            break;
          case 5:
            exit(0);
          default:
            printf("Invalid choice. Please try again.\n");
        }
    }
    return 0;
}
```

OUTPUT:

```
Menu:
1. Insert at beginning
2. Insert at end
3. Insert after position
4. Display list
5. Exit
Enter your choice: 1
Enter value to insert at beginning: 200
Inserted 200 at the beginning.

Menu:
1. Insert at beginning
2. Insert at end
3. Insert after position
4. Display list
5. Exit
Enter your choice: 1
Enter value to insert at beginning: 150
Inserted 150 at the beginning.

Menu:
1. Insert at beginning
2. Insert at end
3. Insert after position
4. Display list
5. Exit
Enter your choice: 1
Enter value to insert at beginning: 100
Inserted 100 at the beginning.
```

```
Menu:
1. Insert at beginning
2. Insert at end
3. Insert after position
4. Display list
5. Exit
Enter your choice: 4
Linked list: 100 -> 150 -> 200 -> NULL

Menu:
1. Insert at beginning
2. Insert at end
3. Insert after position
4. Display list
5. Exit
Enter your choice: 2
Enter value to insert at end: 250
Inserted 250 at the end.

Menu:
1. Insert at beginning
2. Insert at end
3. Insert after position
4. Display list
5. Exit
Enter your choice: 3
Enter position after which to insert: 3
Enter value to insert: 300
```

```
1. Insert at beginning
2. Insert at end
3. Insert after position
4. Display list
5. Exit
Enter your choice: 3
Enter position after which to insert: 3
Enter value to insert: 300
Inserted 300 after position 3.

Menu:
1. Insert at beginning
2. Insert at end
3. Insert after position
4. Display list
5. Exit
Enter your choice: 4
Linked list: 100 -> 150 -> 200 -> 250 -> 300 -> NULL

Menu:
1. Insert at beginning
2. Insert at end
3. Insert after position
4. Display list
5. Exit
Enter your choice: 5
```

WAP of Deletion in linked list.

```
#include <stdio.h>

#include <stdlib.h>


struct Node {

   int data;

   struct Node *next;

};

struct Node *top = NULL;

struct Node* createNode(int value) {

   struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));

   if (!newNode) {

      printf("Memory allocation failed.\n");

      exit(1);

   }

   newNode->data = value;

   newNode->next = NULL;

   return newNode;

}

void insertAtEnd(int value) {

   struct Node *newNode = createNode(value);

   if (top == NULL) {

      top = newNode;

      return;

   }

   struct Node *ptr = top;

   while (ptr->next != NULL) {

      ptr = ptr->next;

   }

   ptr->next = newNode;

}

void deleteFromBeginning() {
```

```
    if (top == NULL) {
        printf("The list is empty. Nothing to delete.\n");
        return;
    }
    struct Node *temp = top;
    top = top->next;
    printf("Deleted %d from the beginning.\n", temp->data);
    free(temp);
}
void deleteFromEnd() {
    if (top == NULL) {
        printf("The list is empty. Nothing to delete.\n");
        return;
    }
    struct Node *ptr = top;
    if (ptr->next == NULL) {
        printf("Deleted %d from the end.\n", ptr->data);
        free(ptr);
        top = NULL;
        return;
    }
    while (ptr->next->next != NULL) {
        ptr = ptr->next;
    }
    struct Node *temp = ptr->next;
    printf("Deleted %d from the end.\n", temp->data);
    ptr->next = NULL;
    free(temp);
}
void deleteByValue(int value) {
    if (top == NULL) {
        printf("The list is empty. Nothing to delete.\n");
```

```
        return;
    }
    struct Node *ptr = top, *prev = NULL;
    if (ptr != NULL && ptr->data == value) {
        top = ptr->next;
        printf("Deleted %d from the list.\n", ptr->data);
        free(ptr);
        return;
    }
    while (ptr != NULL && ptr->data != value) {
        prev = ptr;
        ptr = ptr->next;
    }
    if (ptr == NULL) {
        printf("Value %d not found in the list.\n", value);
        return;
    }
    prev->next = ptr->next;
    printf("Deleted %d from the list.\n", ptr->data);
    free(ptr);
}
void displayList() {
    if (top == NULL) {
        printf("The list is empty.\n");
        return;
    }
    struct Node *ptr = top;
    printf("Linked list: ");
    while (ptr != NULL) {
        printf("%d -> ", ptr->data);
        ptr = ptr->next;
    }
```

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

int main() {
    int choice, value;
    while (1) {
        printf("\nMenu:\n1. Insert at end\n2. Delete from beginning\n3. Delete from end\n4. Delete by value\n5. Display list\n6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to insert at end: ");
                scanf("%d", &value);
                insertAtEnd(value);
                break;
            case 2:
                deleteFromBeginning();
                break;
            case 3:
                deleteFromEnd();
                break;
            case 4:
                printf("Enter value to delete: ");
                scanf("%d", &value);
                deleteByValue(value);
                break;
            case 5:
                displayList();
                break;
            case 6:
```

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

OUTPUT:

```
Menu:
1. Insert at end
2. Delete from beginning
3. Delete from end
4. Delete by value
5. Display list
6. Exit
Enter your choice: 1
Enter value to insert at end: 100

Menu:
1. Insert at end
2. Delete from beginning
3. Delete from end
4. Delete by value
5. Display list
6. Exit
Enter your choice: 1
Enter value to insert at end: 200

Menu:
1. Insert at end
2. Delete from beginning
3. Delete from end
4. Delete by value
5. Display list
6. Exit
Enter your choice: 1
Enter value to insert at end: 300
```

```
Menu:
1. Insert at end
2. Delete from beginning
3. Delete from end
4. Delete by value
5. Display list
6. Exit
Enter your choice: 5
Linked list: 100 -> 200 -> 300 -> NULL

Menu:
1. Insert at end
2. Delete from beginning
3. Delete from end
4. Delete by value
5. Display list
6. Exit
Enter your choice: 2
Deleted 100 from the beginning.

Menu:
1. Insert at end
2. Delete from beginning
3. Delete from end
4. Delete by value
5. Display list
6. Exit
Enter your choice: 5
```

```
Menu:
1. Insert at end
2. Delete from beginning
3. Delete from end
4. Delete by value
5. Display list
6. Exit
Enter your choice: 1
Enter value to insert at end: 400

Menu:
1. Insert at end
2. Delete from beginning
3. Delete from end
4. Delete by value
5. Display list
6. Exit
Enter your choice: 5
Linked list: 200 -> 300 -> 400 -> NULL

Menu:
1. Insert at end
2. Delete from beginning
3. Delete from end
4. Delete by value
5. Display list
6. Exit
```

```
6. Exit
Enter your choice: 3
Deleted 400 from the end.

Menu:
1. Insert at end
2. Delete from beginning
3. Delete from end
4. Delete by value
5. Display list
6. Exit
Enter your choice: 4
Enter value to delete: 300
Deleted 300 from the list.

Menu:
1. Insert at end
2. Delete from beginning
3. Delete from end
4. Delete by value
5. Display list
6. Exit
Enter your choice: 5
Linked list: 200 -> NULL

Menu:
1. Insert at end
2. Delete from beginning
3. Delete from end
4. Delete by value
```

```
3. Delete from end
4. Delete by value
5. Display list
6. Exit
Enter your choice: 6

Process returned 0 (0x0)   execution time : 77.742 s
Press any key to continue.
```

Lab program 6:

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

#include<stdio.h>

#include<stdlib.h>

```c
#include<stddef.h>

struct Node {
    int data;
    struct Node *next;
};
struct Node *top1 = NULL, *top2 = NULL;

void sortLinkedList(struct Node *top) {
    struct Node *ptr, *pre_ptr;
    int temp;

    if (top == NULL || top->next == NULL) {
        return;
    }
    for (ptr = top; ptr != NULL; ptr = ptr->next) {
        for (pre_ptr = top; pre_ptr->next != NULL; pre_ptr = pre_ptr->next) {
            if (pre_ptr->data > pre_ptr->next->data) {
                temp = pre_ptr->data;
                pre_ptr->data = pre_ptr->next->data;
                pre_ptr->next->data = temp;
            }
        }
    }
}

struct Node* reverseLinkedList(struct Node *top) {
    struct Node *ptr = top, *pre_ptr = NULL, *next_ptr = NULL;

    while (ptr != NULL) {
        next_ptr = ptr->next;
        ptr->next = pre_ptr;
```

```
        pre_ptr = ptr;
        ptr = next_ptr;
    }
    return pre_ptr;
}

struct Node* concatenateLinkedLists(struct Node *top1, struct Node *top2) {
    struct Node *ptr = top1;

    if (top1 == NULL) return top2;
    if (top2 == NULL) return top1;

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

    ptr->next = top2;
    return top1;
}

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

void displayList(struct Node *top) {
    struct Node *ptr = top;
    while (ptr != NULL) {
        printf("%d -> ", ptr->data);
        ptr = ptr->next;
```

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

void main() {
    top1 = createNode(20);
    top1->next = createNode(30);
    top1->next->next = createNode(40);

    top2 = createNode(50);
    top2->next = createNode(60);

    printf("Original List 1: ");
    displayList(top1);

    printf("Original List 2: ");
    displayList(top2);

    sortLinkedList(top1);
    printf("Sorted List 1: ");
    displayList(top1);

    top1 = reverseLinkedList(top1);
    printf("Reversed List 1: ");
    displayList(top1);

    struct Node *mergedList = concatenateLinkedLists(top1, top2);
    printf("Concatenated List: ");
    displayList(mergedList);
}
```

OUTPUT:

```
Original List 1: 100 -> 300 -> 200 -> NULL
Original List 2: 400 -> 500 -> NULL
Sorted List 1: 100 -> 200 -> 300 -> NULL
Reversed List 1: 300 -> 200 -> 100 -> NULL
Concatenated List: 300 -> 200 -> 100 -> 400 -> 500 -> NULL

Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.
```

Lab program 7:

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

```c
#include <stdio.h>

#include <stdlib.h>


struct Node {

   int data;

   struct Node* next;

};


struct Node* top = NULL;

struct Node* front = NULL;

struct Node* rear = NULL;


struct Node* createNode(int value) {

   struct Node* ptr = (struct Node*)malloc(sizeof(struct Node));

   ptr->data = value;

   ptr->next = NULL;

   return ptr;

}


void push(int value) {

   struct Node* ptr = createNode(value);

   ptr->next = top;
```

```
    top = ptr;
    printf("Pushed %d to stack\n", value);
}

void pop() {
    if (top == NULL) {
        printf("Stack Underflow\n");
        return;
    }
    struct Node* ptr = top;
    printf("Popped %d from stack\n", top->data);
    top = top->next;
    free(ptr);
}

void enqueue(int value) {
    struct Node* ptr = createNode(value);
    if (rear == NULL) {
        front = rear = ptr;
    } else {
        rear->next = ptr;
        rear = ptr;
    }
    printf("Enqueued %d to queue\n", value);
}

void dequeue() {
    if (front == NULL) {
        printf("Queue Underflow\n");
        return;
    }
    struct Node* ptr = front;
```

```
    printf("Dequeued %d from queue\n", front->data);
    front = front->next;
    if (front == NULL) {
        rear = NULL;
    }
    free(ptr);
}

void displayStack() {
    struct Node* ptr = top;
    if (ptr == NULL) {
        printf("Stack is Empty\n");
        return;
    }
    printf("Stack: ");
    while (ptr != NULL) {
        printf("%d -> ", ptr->data);
        ptr = ptr->next;
    }
    printf("NULL\n");
}

void displayQueue() {
    struct Node* ptr = front;
    if (ptr == NULL) {
        printf("Queue is Empty\n");
        return;
    }
    printf("Queue: ");
    while (ptr != NULL) {
        printf("%d -> ", ptr->data);
        ptr = ptr->next;
```

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

int main() {
    push(500);
    push(600);
    push(700);
    displayStack();
    pop();
    displayStack();

    enqueue(800);
    enqueue(900);
    enqueue(1000);
    displayQueue();
    dequeue();
    displayQueue();

    return 0;
}
```

OUTPUT:

};

```
Pushed 10 to stack
Pushed 20 to stack
Pushed 30 to stack
Stack: 30 -> 20 -> 10 -> NULL
Popped 30 from stack
Stack: 20 -> 10 -> NULL
Enqueued 40 to queue
Enqueued 50 to queue
Enqueued 60 to queue
Queue: 40 -> 50 -> 60 -> NULL
Dequeued 40 from queue
Queue: 50 -> 60 -> NULL

Process returned 0 (0x0)   execution time : 0.042 s
Press any key to continue.
```

Lab program 8:

1.WAP to Implement doubly link list with primitive operations Create a doubly linked list.Insert a new node to the left of the node.Delete the node based on a specific value Display the contents of the list.

```
#include <stdio.h>

#include <stdlib.h>



struct Node {

    int data;

    struct Node *prev;

    struct Node *next;

};
```

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


void insertAtBeginning(struct Node** head, int value) {
    struct Node* newNode = createNode(value);
    if (*head == NULL) {
        *head = newNode;
    } else {
        newNode->next = *head;
        (*head)->prev = newNode;
        *head = newNode;
    }
}


void insertBefore(struct Node** head, int newValue, int targetValue) {
    struct Node* temp = *head;
    while (temp != NULL && temp->data != targetValue) {
        temp = temp->next;
    }

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

```
    struct Node* newNode = createNode(newValue);
    newNode->prev = temp->prev;
    newNode->next = temp;

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


void deleteNode(struct Node** head, int value) {
    struct Node* temp = *head;
    while (temp != NULL && temp->data != value) {
        temp = temp->next;
    }

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

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

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

```
    }

    free(temp);
    printf("Node with value %d deleted.\n", value);
}


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

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

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

    while (1) {
        printf("\nDoubly Linked List Operations:\n");
        printf("1. Insert at Beginning\n");
        printf("2. Insert Before a Node\n");
        printf("3. Delete a Node\n");
        printf("4. Display List\n");
```

```
    printf("5. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
      case 1:
        printf("Enter value to insert at the beginning: ");
        scanf("%d", &value);
        insertAtBeginning(&head, value);
        break;
      case 2:
        printf("Enter target value before which to insert: ");
        scanf("%d", &targetValue);
        printf("Enter new value to insert: ");
        scanf("%d", &value);
        insertBefore(&head, value, targetValue);
        break;
      case 3:
        printf("Enter value to delete: ");
        scanf("%d", &value);
        deleteNode(&head, value);
        break;
      case 4:
        displayList(head);
        break;
      case 5:
        printf("Exiting...\n");
        return 0;
      default:
        printf("Invalid choice! Please try again.\n");
    }
  }
```

```
    return 0;
}
```

OUTPUT:

```
Initial list:
List contents: 10 20 30

List after inserting 15 to the left of 20:
List contents: 10 15 20 30
Node with value 20 deleted.

List after deleting node with value 20:
List contents: 10 15 30
Node with value 100 not found.

List after attempting to delete node with value 100:
List contents: 10 15 30

Process returned 0 (0x0)   execution time : 0.045 s
Press any key to continue.
```

Lab program 9:

Write a program

- **To construct a binary Search tree.**
- **To traverse the tree using all the methods i.e., in-order, preorder and post order**
- **To display the elements in the tree.**

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


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


struct Node* createNode(int data) {
```

```
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->left = newNode->right = NULL;
    return newNode;
}

struct Node* insert(struct Node* root, int data) {
    if (root == NULL) {
        return createNode(data);
    }

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

    return root;
}

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

void preOrderTraversal(struct Node* root) {
    if (root != NULL) {
```

```
        printf("%d ", root->data);
        preOrderTraversal(root->left);
        preOrderTraversal(root->right);
    }
}

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

void displayTree(struct Node* root) {
    printf("In-order Traversal: ");
    inOrderTraversal(root);
    printf("\n");

    printf("Pre-order Traversal: ");
    preOrderTraversal(root);
    printf("\n");

    printf("Post-order Traversal: ");
    postOrderTraversal(root);
    printf("\n");
}

int main() {
    struct Node* root = NULL;
```

```c
    int n, data;

    printf("Enter the number of elements to insert in the binary search tree: ");
    scanf("%d", &n);

    printf("Enter the elements to insert into the tree:\n");
    for (int i = 0; i < n; i++) {
        printf("Enter element %d: ", i + 1);
        scanf("%d", &data);
        root = insert(root, data);
    }

    displayTree(root);

    return 0;
}
```

OUTPUT:

```
Enter the number of elements to insert in the binary search tree:
 6
Enter the elements to insert into the tree:
Enter element 1: 8
Enter element 2: 20
Enter element 3: 25
Enter element 4: 14
Enter element 5: 9
Enter element 6: 3
In-order Traversal: 3 8 9 14 20 25
Pre-order Traversal: 8 3 20 14 9 25
Post-order Traversal: 3 9 14 25 20 8

Process returned 0 (0x0)   execution time : 24.585 s
Press any key to continue.
```

Lab program 10:

Write a program to traverse a graph using BFS method.

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

```c
#include <stdbool.h>

#define MAX_NODES 100

struct Queue {
    int items[MAX_NODES];
    int front, rear;
};

void initQueue(struct Queue* q) {
    q->front = -1;
    q->rear = -1;
}

bool isQueueEmpty(struct Queue* q) {
    return (q->front == -1);
}

void enqueue(struct Queue* q, int value) {
    if (q->rear == MAX_NODES - 1)
        printf("Queue Overflow\n");
    else {
        if (q->front == -1)
            q->front = 0;
        q->rear++;
        q->items[q->rear] = value;
    }
}

int dequeue(struct Queue* q) {
```

```
    if (isQueueEmpty(q)) {
        printf("Queue Underflow\n");
        return -1;
    } else {
        int item = q->items[q->front];
        if (q->front == q->rear) {
            q->front = q->rear = -1;
        } else {
            q->front++;
        }
        return item;
    }
}

void BFS(int adjMatrix[MAX_NODES][MAX_NODES], int numVertices, int startVertex) {
    struct Queue q;
    initQueue(&q);

    bool visited[MAX_NODES] = {false};
    visited[startVertex] = true;

    enqueue(&q, startVertex);

    printf("BFS Traversal starting from vertex %d: ", startVertex);

    while (!isQueueEmpty(&q)) {
        int currentVertex = dequeue(&q);
        printf("%d ", currentVertex);

        for (int i = 0; i < numVertices; i++) {
```

```
            if (adjMatrix[currentVertex][i] == 1 && !visited[i]) {
                visited[i] = true;
                enqueue(&q, i);
            }
        }
    }
    printf("\n");
}

int main() {
    int adjMatrix[MAX_NODES][MAX_NODES];
    int numVertices, startVertex;

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

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

    printf("Enter the starting vertex for BFS: ");
    scanf("%d", &startVertex);

    BFS(adjMatrix, numVertices, startVertex);

    return 0;
}
```

OUTPUT:

```
Enter the number of vertices in the graph: 5
Enter the adjacency matrix (enter 1 if there's an edge, otherwise 0):
0 1 0 1 1
1 0 0 1 0
0 1 0 1 1
1 0 1 0 1
0 1 1 1 0
Enter the starting vertex (0 to 4): 3
BFS Traversal: 3 0 2 4 1

Process returned 0 (0x0)   execution time : 61.713 s
Press any key to continue.
```

Lab program 11:

write a program to traverse a graph using DFS method:

```
#include <stdio.h>

#include <stdbool.h>


#define MAX 100


int adjMatrix[MAX][MAX];

bool visited[MAX];

int stack[MAX];

int top = -1;


void push(int vertex) {

   if (top == MAX - 1) {

      printf("Stack Overflow\n");

      return;

   }

   stack[++top] = vertex;

}


int pop() {

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

```
        printf("Stack Underflow\n");
        return -1;
    }
    return stack[top--];
}

void dfsUsingStack(int startVertex, int numVertices) {
    push(startVertex);
    visited[startVertex] = true;

    while (top != -1) {
        int currentVertex = pop();

        for (int i = 0; i < numVertices; i++) {
            if (adjMatrix[currentVertex][i] == 1 && !visited[i]) {
                push(i);
                visited[i] = true;
            }
        }
    }
}

bool isConnected(int numVertices) {
    for (int i = 0; i < numVertices; i++) {
        visited[i] = false;
    }

    dfsUsingStack(0, numVertices);

    for (int i = 0; i < numVertices; i++) {
        if (!visited[i]) {
            return false;
```

```
        }
    }
    return true;
}

int main() {
    int numVertices, numEdges;
    printf("Enter the number of vertices: ");
    scanf("%d", &numVertices);

    printf("Enter the number of edges: ");
    scanf("%d", &numEdges);

    for (int i = 0; i < numVertices; i++) {
        for (int j = 0; j < numVertices; j++) {
            adjMatrix[i][j] = 0;
        }
    }

    printf("Enter the edges (start_vertex end_vertex):\n");
    for (int i = 0; i < numEdges; i++) {
        int u, v;
        scanf("%d %d", &u, &v);
        adjMatrix[u][v] = 1;
        adjMatrix[v][u] = 1;
    }

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

```
    return 0;
}
```

OUTPUT:

```
Enter the number of vertices: 6
Enter the number of edges: 8
Enter the edges (start_vertex end_vertex):
0 1
1 2
2 3
3 4
4 5
5 6
6 7
7 8
The graph is connected.

Process returned 0 (0x0)   execution time : 39.121 s
Press any key to continue.
```

Lab program 12:

write Hashing program:

```
#include <stdio.h>
#define MAX 100

struct Employee {
    int k;
    char n[50];
};

struct Employee ht[MAX];
int ts;

void init() {
    for (int i = 0; i < MAX; i++) ht[i].k = -1;
```

```
}

int hash(int k) {
    return k % ts;
}

void insert(int k, char n[]) {
    int idx = hash(k);
    while (ht[idx].k != -1) {
        idx = (idx + 1) % ts;
    }
    ht[idx].k = k;
    for (int i = 0; n[i] != '\0' && i < 49; i++) {
        ht[idx].n[i] = n[i];
    }
    ht[idx].n[49] = '\0';
}

void display() {
    for (int i = 0; i < ts; i++) {
        if (ht[i].k != -1)
            printf("Idx %d: Key = %d, Name = %s\n", i, ht[i].k, ht[i].n);
        else
            printf("Idx %d: Empty\n", i);
    }
}

int main() {
    int n;
    printf("Enter table size (max size %d): ", MAX);
    scanf("%d", &ts);
```

```
    if (ts > MAX) ts = MAX;

    init();

    printf("Enter number of employees: ");
    scanf("%d", &n);
    getchar();

    for (int i = 0; i < n; i++) {
        int k;
        char name[50];
        printf("Enter key and name for employee %d: ", i + 1);
        scanf("%d", &k);
        getchar();
        gets(name);
        insert(k, name);
    }

    display();

    return 0;
}
```

OUTPUT:

```
Enter table size (max size 100): 7
Enter number of employees: 4
Enter key and name for employee 1: 100 raja
Enter key and name for employee 2: 200 rohan
Enter key and name for employee 3: 300 rohit
Enter key and name for employee 4: 400 kush
Idx 0: Empty
Idx 1: Key = 400, Name = kush
Idx 2: Key = 100, Name = raja
Idx 3: Empty
Idx 4: Key = 200, Name = rohan
Idx 5: Empty
Idx 6: Key = 300, Name = rohit

Process returned 0 (0x0)   execution time : 43.011 s
Press any key to continue.
```

lab program 13:

Given an array nums of size n, return the majority elements. The majority element is the element that appears more than [n/2] times. you may assume that the majority element always exists in the array.

```
#include <stdio.h>
int majorityElement(int* nums, int numsSize) {
int candidate = nums[0];
int count = 1;
for (int i = 1; i < numsSize; i++) {
if (count == 0) {
candidate = nums[i];
count = 1;
} else if (nums[i] == candidate) {
count++;
} else {
count--;
}
}
return candidate;
```

}

OUTPUT:

```
Input: nums = [2,2,1,1,1,2,2]
Output: 2
```

lab program 14:

Given an integer array nums, move all zeroes to the end of it while maintaining

the relative order of the non-zero elements.

```
#include <stdio.h>
void moveZeroes(int *nums, int numsSize){
int L=0, r=numsSize-1;
for (int i = 0; i < r; i++) {
if (nums[i] == 0) {
for(int j = 1; j<r; j++){
nums[j] = nums[j+1];
}
nums[r] = 0;
r--;
}
}
```

OUTPUT:

```
OUTPUT:-
For the input array {0, 1, 0, 3, 12} the output will be:
1 3 12 0 0
```

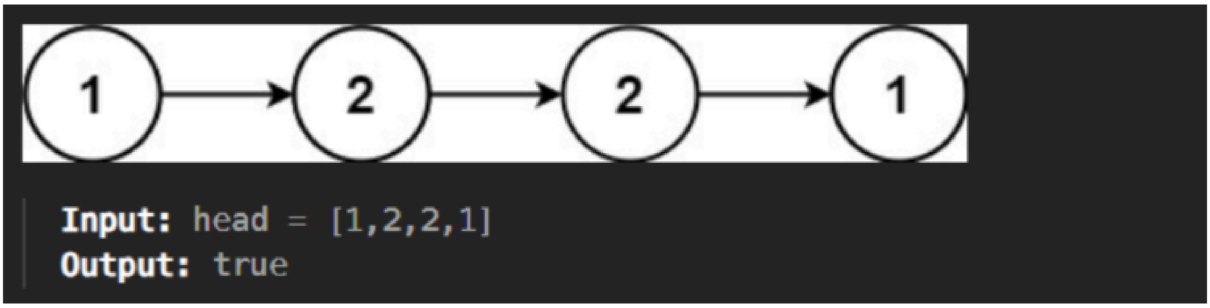
lab program 15:

Given the head of a singly linked list, return true if it is a palindrome OR false otherwise.

```
struct ListNode {
int val;
struct ListNode* next;
};
bool isPalindrome(struct ListNode* head) {
if (!head || !head->next) return true;
struct ListNode* slow = head;
struct ListNode* fast = head;
while (fast && fast->next) {
slow = slow->next;
fast = fast->next->next;
}
struct ListNode* prev = NULL;
while (slow) {
struct ListNode* nextTemp = slow->next;
slow->next = prev;
prev = slow;
slow = nextTemp;
}
struct ListNode* firstHalf = head;
struct ListNode* secondHalf = prev;
while (secondHalf) {
if (firstHalf->val != secondHalf->val) return false;
firstHalf = firstHalf->next;
secondHalf = secondHalf->next;
```

```
}
return true;
}
```

OUTPUT :



lab program 16 :

Given the root of a binary tree and an integer targetSum, return true if the tree

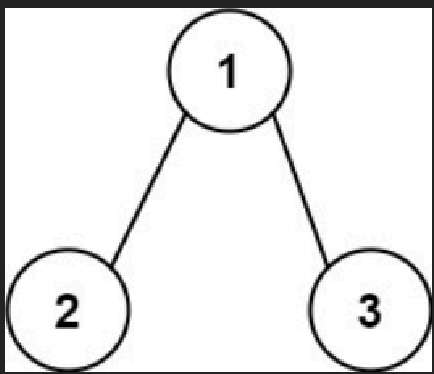
has a root-to-leaf path such that adding up all the values along the path equals.

targetSum, A leaf is a node with no children.

```
struct TreeNode {
int val;
struct TreeNode* left;
struct TreeNode* right;
};
bool hasPathSum(struct TreeNode* root, int targetSum) {
if (root == NULL) {
return false;
}
if (root->left == NULL && root->right == NULL) {
return root->val == targetSum;
}
targetSum -= root->val;
return hasPathSum(root->left, targetSum) || hasPathSum(root->right, targetSum);
}
struct TreeNode* createNode(int val) {
```

```
struct TreeNode* newNode = (struct TreeNode*)malloc(sizeof(struct TreeNode));

newNode->val = val;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}
```

OUTPUT:



Input: root = [1,2,3], targetSum = 5
Output: false
Explanation: There are two root-to-leaf paths in the tree:
(1 --> 2): The sum is 3.
(1 --> 3): The sum is 4.
There is no root-to-leaf path with sum = 5.