VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

SHANLKAR SHIVAPPA PUJAR (1BM23CS309)

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 **SHANKAR SHIVAPPA PUJAR(1BM23CS309)**, 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.

Geetha N Assistant Professor Department of CSE BMSCE, Bengaluru	**Dr. Kavitha Sooda** Professor and Head Department of CSE BMSCE, Bengaluru

Index Sheet

GITHUB LINK: https://github.com/shankar045/DSA_LAB.git

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

PROGRAM:

```
#include <stdio.h>

#include <stdlib.h>


#define SIZE 4


int stack[SIZE];

int top = -1;



void push(int element) {

    if (top == SIZE - 1) {

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

    } else {

        top++;

        stack[top] = element;

        printf("%d pushed to stack\n", element);

    }

}



int pop() {

    if (top == -1) {

        printf("Stack Underflow! No element to pop\n");

        return -1;

    } else {
```

```
        int poppedElement = stack[top];
        top--;
        return poppedElement;
    }
}

int peek() {
    if (top == -1) {
        printf("Stack is empty\n");
        return -1;
    } else {
        return stack[top];
    }
}

int isEmpty() {
    return top == -1;
}

int isFull() {
    return top == SIZE - 1;
}

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

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

int main() {

    push(10);
    push(20);
    push(30);
    push(40);
    push(50);

    printf("\nTop element is: %d\n", peek());

    printf("Is stack full? %s\n", isFull() ? "true" : "false");
    printf("Is stack empty? %s\n", isEmpty() ? "true" : "false");

    printf("\nPopped element: %d\n", pop());
    printf("Popped element: %d\n", pop());

    display();

    return 0;
```

OUTPUT:

```
10 pushed to stack
20 pushed to stack
30 pushed to stack
40 pushed to stack
Stack Overflow! Cannot push 50

Top element is: 40
Is stack full? true
Is stack empty? false

Popped element: 40
Popped element: 30
Stack elements are:
20
10

Process returned 0 (0x0)   execution time : 0.008 s
Press ENTER to continue.
```

1) Stack Operation (or) program to implement stack using arr

```
#include <stdio.h>
# define MAX 5
int Stack [MAX];
int top = -1;
void push (int value){
    if (top == MAX-1){
    printf ("Stack overflow !" Cant push %d\n", value);
  }
  Else{
    top ++;
    Stack [top] = value;
     printf ("%d pushed to the Stack\n", value);
     }
   }
void pop(){
    if (top == -1){
    printf (" Stack underflow! no element to pop\n");
   }
 Else {
  print ("%d popped from the Stack \n", Stack[top]);
  ~~top --;~~
  }
}
void display (){
  if (top == -1){
```

```
printf("Stack is Empty\n"); }
 else{
  printf("Stack Elements are:");
 for (int i=0; i<=top; i++){
    printf("%d", stack[i]);
 }
  printf("\n");
 }
}
int main() {
  int choice value;
  while (1){
     printf("\n", push\n, pop\n, Display\n, Exist\n");
     printf(" Enter your choice:");
     scanf("%d", &choice);
     switch (choice){
      case 1:
        printf("Enter value to push:");
        scanf("%d", &value);
        push (value);
         break;
      case 2:
        pop();
        break;
      case 3:
         display();
         break;
      case 4:
         return 0;
      default:
         printf("Invalid choice\n");
       }
     }
}
```

OUTPUT:

10 pushed to stack
20 pushed to stack
30 pushed to stack
40 pushed to stack
Top stack overflow.
top element is 40
Stack full: false
poped element is 40
poped element is 30
Stack elements are:
20
10

Lab program 2:

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

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

int precedence(char c) {
    if (c == '^')
        return 3;
    else if (c == '*' || c == '/')
        return 2;
    else if (c == '+' || c == '-')
        return 1;
    return -1;
}
char associativity(char c) {
    if (c == '^')
        return 'R';
    return 'L';
```

```
}

void infixToPostfix(char* exp) {
    int len = strlen(exp);

    char* result = (char*)malloc((len + 1) * sizeof(char));
    char* stack = (char*)malloc(len * sizeof(char));
    if (!result || !stack) {
        printf("Memory allocation failed\n");
        return;
    }

    int resultIndex = 0, stackIndex = -1;

    for (int i = 0; i < len; i++) {
        char c = exp[i];

        if (isalnum(c)) {
            result[resultIndex++] = c;
        }

        else if (c == '(') {
            stack[++stackIndex] = c;
        }
        else if (c == ')') {
            while (stackIndex >= 0 && stack[stackIndex] != '(') {
                result[resultIndex++] = stack[stackIndex--];
```

```
            }
            stackIndex--; // Remove '(' from stack
        }

        else {
            while (stackIndex >= 0 &&
                   (precedence(c) < precedence(stack[stackIndex]) ||
                   (precedence(c) == precedence(stack[stackIndex]) &&
                    associativity(c) == 'L'))) {
                result[resultIndex++] = stack[stackIndex--];
            }
            stack[++stackIndex] = c;
        }
    }

    while (stackIndex >= 0) {
        result[resultIndex++] = stack[stackIndex--];
    }

    result[resultIndex] = '\0';
    printf("Postfix Expression: %s\n", result);

    free(result);
    free(stack);
}

int main() {
```

```
    char exp[] = "a+b*(c^d-e)^(f+g*h)-i";
    printf("Infix Expression: %s\n", exp);
    infixToPostfix(exp);
    return 0;
}
```

OUTPUT:

```
Infix Expression: a+b*(c^d-e)^(f+g*h)-i
Postfix Expression: abcd^e-fgh*+^*+i-

Process returned 0 (0x0)   execution time : 0.006 s
Press ENTER to continue.
```

2> WAP to convert a given valid parenthesized infix arithme-tic expression to postfix expression. The expression consist of single character operands and binary operators + (plus) - (minus) * (multiply) and / (divide)

Program:

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

int prec(char c){
    if(c=='^')
        return 3;
    Else if(c=='/' || c=='*')
        return 2;
    Else if(c=='+' || c=='-')
        return 1;
    Else
        return -1;}

char associativity(char c){
    if(c=='^')
    return 'R';
    return 'L';
}

void infixToPostfix(const char* s){
    int len= strlen(s);
    char* result =(char*) malloc(len+1);
    char* stack= (char*)malloc(len);
    int resultIndex =0;
    int stackIndex = -1;
    if(!result||!stack){
        printf("Memory allocation failed!\n");
        return;
    }
    for(int i=0; i<len; i++) {
    char c= s[i];
    if((c>='a' && c<='z')||(c>='A' && c<='Z')||(c>='0' && c<=
            '9')){
            result[resultIndex++]=c;
    }
    Else if (c=='('){
        stack[++stackIndex]=c;
    }
```

~~Else if (c=='('){~~

```
    Else if(c==')'){
        while (stackIndex>=0 && stack[stackIndex]!='('){
        result[resultIndex++]= stack[stackIndex--];
        }
        stackIndex--;
        }
    Else{
        while(stackIndex>=0 && (prec(c)<prec(stack[stack
        Index]) || (prec(c)== prec(stack[stackIndex])
```

```
if associativity (c) = 'L'))){
        result[resultIndex ++] = stack[stackIndex --];
    }
    stack[++stackIndex] = c;
  }
}
while (stackIndex >= 0) {
the stack
    result [resultIndex ++] = stack [stackIndex --];
}
result[resultIndex] = '\0';
  printf("%s\n", result);
  free (result);
  free (stack); }
  int main () {
    char exp[] = "a + b*(c^d - e)^(f+g*h)-i";
  infixToPostfix (exp);
   return 0;
  }
```

user input

Output:

abcd^e-fgh*+^*+i-

Nameeta M.
7/10/2024

```
}
```

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

PROGRAM:

```
#include <stdio.h>

#include <stdlib.h>


#define QUEUE_SIZE 10



int queue[QUEUE_SIZE];

int front = -1, rear = -1;



void insert(int item) {

    if (rear == QUEUE_SIZE - 1) {

        printf("Queue Overflow! Cannot insert %d.\n", item);

        return;

    }

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

```
        front = 0;
    }
    rear++;
    queue[rear] = item;
    printf("Inserted: %d\n", item);
}


int delete() {
    if (front == -1 || front > rear) {
        printf("Queue Underflow! Queue is empty.\n");
        return -1;
    }
    int deletedItem = queue[front];
    printf("Deleted: %d\n", deletedItem);
    front++;


    if (front > rear) {
        front = rear = -1;
    }
    return deletedItem;
}
```

```
void display() {
    if (front == -1 || front > rear) {
        printf("Queue is empty.\n");
        return;
    }
    printf("Queue contains: ");
    for (int i = front; i <= rear; i++) {
        printf("%d ", queue[i]);
    }
    printf("\n");
}


int main() {
    int choice, item;

    while (1) {
        printf("\nQueue 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 insert: ");
                scanf("%d", &item);
                insert(item);
                break;
            case 2:
                delete();
                break;
            case 3:
                display();
                break;
            case 4:
                printf("Exiting program.\n");
                exit(0);
                break;
            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 element to insert: 10
Inserted: 10

Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Deleted: 10

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

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

3) WAP to stimulate the working of a queue of integers using an array provide the following operators insert deleat display.

```
#include <stdio.h>

#define queSize 10

int item front=10, rear=-1, q[10];

void insert(){
if(rear==que-size-1){
printf("stack over-flow! \n");
return;
}

rear+=1;

q[rear]=item;
}
int deleate(){
if(front>rear){
printf("Queue is empty \n");
return -1;
}
return q[fron+1];
void display(){
int i;
if(front>rear){
printf("Queue is empty \n");
```

}
return;
printf ("queue contains: |n");
for (i= front; i<= rear; i++) {
printf ("%d|n", q[i]);
}
}

output:

options:
1) insert
2) Deleate
3) print queue
4) Exist

Select an option: 1
Enter the value to insert: 10

options:
1) Insert
2) Deleate
3) print queue
4) Exist

select an option: 3
Current queue contains: 10
options:
1) Insert
2) Delete
3) print queue
4) Exist

Select an option: 4
program terminating

Namratha M
22/10/2024

Lab program 4:

Write a program to simulate the working of a queue of integers using an

array. Provide the following operations

a) Insert b) Delete c) Display

The program should print appropriate messages for queue empty and queue

overflow conditions

PROGRAMM:

```
#include <stdio.h>

#define SIZE 5


int queue[SIZE];

int front = -1, rear = -1;



int is_full() {

    return ((rear + 1) % SIZE == front);

}



int is_empty() {
```

```
    return (front == -1);
}


void insert(int value) {
    if (is_full()) {
        printf("Queue Overflow\n");
        return;
    }
    if (front == -1) {
        front = rear = 0;
    } else {
        rear = (rear + 1) % SIZE;
    }
    queue[rear] = value;
    printf("Inserted: %d\n", value);
}


void delete() {
    if (is_empty()) {
        printf("Queue Underflow\n");
        return;
    }
```

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


void display() {
    if (is_empty()) {
        printf("Queue is Empty\n");
        return;
    }
    printf("Queue: ");
    int i = front;
    while (1) {
        printf("%d ", queue[i]);
        if (i == rear) break;
        i = (i + 1) % SIZE;
    }
    printf("\n");
}
```

```c
int main() {
    int choice, value;
    while (1) {
        printf("\nCircular Queue Operations:\n");
        printf("1. Insert\n2. Delete\n3. Display\n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the value to insert: ");
                scanf("%d", &value);
                insert(value);
                break;
            case 2:
                delete();
                break;
            case 3:
                display();
                break;
            case 4:
                printf("Exiting...\n");
                return 0;
```

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

OUTPUT 4:

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

Circular Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Deleted: 5

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

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

Process returned 0 (0x0)   execution time : 49.717 s
Press ENTER to continue.
```

Q4) 21/10/24 WAP to Stimulate working of Circular - queue of integers using an array provide the following operation Insert delete & Display
The program should print appropriate messages for a queue Empty and queue overflow condition

Program:

```
#include <stdio.h>
#define size 10
int queue [size];
int front rear = -1;    } pass by reference
int is_full () {
  if (front == 0 && rear_size-1)||
  (rear == (front-1)){
   return1;
int is Empty () {
  if (front == -1){
  return 1; }
void Insert(int value){
     if (is full){
   printf ("queue overflow\n");
     return 1;
}
```

```
if (front == -1) {
  front = rear = 0;
else if (rear = size-1 && front != 0) {
  rear = 0;
  }
Else
  {
   rear = (rear+1) % size;
  }
Queue [rear] = value;
void delete () {
   if (is Empty ()) {
   printf (" Queue is overflow\n"); }
  printf (" Delete %d \n", queue [front]);
 if (front == rear) {
    front = rear = -1}
   Else if front = size-1) {
     front = 0}
    Else {
  front = (front+1) % size;
 }
}
```

```
void display () {
 if (is Empty ()) {
   printf ("Queue is Empty "); }
   Else {
   printf ("Queue: ");
     for  (int i = front; i != rear; i = (i+1) % size) {
          printf (" %d ", Queue [i]);

       }
```

execute
Namrata M.
24/10/2024

<u>output</u>:

Circular Queue Operations:
1) Insert
2) Delete
3) display
4) Exist

Enter your choice : 1
 Enter a value to insert : 5
  Inserted 5

~~Circular~~ ~~Queue~~ Operations
~~1) Insert~~ ~~delete~~
2) ~~delete~~
3) ~~display~~

```
void display(){
if(is_empty()){
printf("queue is Empty");}

Else{
printf("queue:");
for(int i = front; i != rear; i=(i+1)%size){

printf("%d", queue[i]);
}
```

Circular queue operations:
1. Insert
2. Delete
3. Display
4. Exist

enter your choice:3
queue element: 5

Circular queue operations:
1. Insert
2. Delete
3. Display
4. Exist

Deleted 5

Circular queue operations:
1. Insert
2. Delete
3. Display
4. Exist

Enter your choice:4
exiting...

Lab program 5 A:

Write a program to implement Singly Linked List with following

operations

a) Create a linked list.

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

c) Display the contents of the linked list

PROGRAMM:

```
#include <stdio.h>

#include <stdlib.h>


struct Node {

    int data;

    struct Node* next;

};


struct Node* createNode(int data) {

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

```
    if (newNode == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

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

void insertEnd(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    struct Node* temp = *head;
```

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

void displayList(struct Node* head) {
    if (head == NULL) {
        printf("The list is empty.\n");
        return;
    }
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

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

```
int choice, value;
while (1) {
    printf("\n1. Insert at beginning\n");
    printf("2. Insert at end\n");
    printf("3. Display the linked list\n");
    printf("4. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("Enter value to insert at beginning: ");
            scanf("%d", &value);
            insertBeg(&head, value);
            break;
        case 2:
            printf("Enter value to insert at end: ");
            scanf("%d", &value);
            insertEnd(&head, value);
            break;
        case 3:
            printf("Linked List: ");
```

```
            displayList(head);
            break;
        case 4:
            printf("Exiting...\n");
            exit(0);
        default:
            printf("Invalid choice, please try again.\n");
      }
   }

   return 0;
}
```

OUTPUT:

```
2. Insert at end
3. Display the linked list
4. Exit
Enter your choice: 1
Enter value to insert at beginning: 10

1. Insert at beginning
2. Insert at end
3. Display the linked list
4. Exit
Enter your choice: 2
Enter value to insert at end: 20

1. Insert at beginning
2. Insert at end
3. Display the linked list
4. Exit
Enter your choice: 3
Linked List: 10 -> 20 -> NULL

1. Insert at beginning
2. Insert at end
3. Display the linked list
4. Exit
Enter your choice: 4
Exiting...

Process returned 0 (0x0)   execution time : 57.032 s
Press any key to continue.
```

06 week 20/11/24

05) WAP to Implement Singly linked list with following Operation

a) Create a linked list

b) Insertion of node at first position and end of list

Display the content of the linked list

Program:

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

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

Struct Node* Create Node (int data) {
    Struct Node* new node = (Struct Node*) Malloc (sizeof (struct node));
    newNode -> data = data;
    new Node -> data = NULL;
    return New node;
}
void insertAtBeginning (Struct Node** head, int data) {
    Struct Node* New Node =
        Create Node (data);
    new node -> next = * head;
    * head = New Node;
}
void insertAtEnd (Struct Node** head int data) {
    Struct Node * new Node =
        Create Node (data);
if (* head == NULL) {
  * head = New Node;
  return;
}
Struct Node* temp = * head;
  while (temp -> Next != NULL) {
    temp = temp -> next; }

  temp -> next = New Node;
void display (Struct Node* head) {
    Struct Node * temp = head;

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

execute
Namratha M.
28/10/2024

Output:

1. Insert at beggining
2. Insert at end
3. Display
4. Exit

Enter your Choice: 1

Enter data to insert at beginning: 10

Menu
1 Insert at beginning
2. Insert at end
3. Display list
4. Exist
Enter your choice: 1
Enter data ~~insert end~~ : 20
Enter data insert beginning

Menu:
1. Insert at begining
2. Insert at end
3. Display list
4. Exist

Enter your choice: 2
~~Enter data to insert end~~
~~linked list: 20->10->~~
Enter data to insert end: 40

Menu:

Enter your choice: 3
linked list: 20->10->40->NULL

Lab program 5 B:

Write a program to Implement Singly Linked List with following operations

a) Create a linked list.

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

c) Display the contents of the linked list.

PROGRAM:

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

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

struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
```

```
        printf("Memory allocation failed.\n");
        exit(1);
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

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

void insertEnd(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    struct Node* temp = *head;
    while (temp->next != NULL) {
```

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

void deleteFirst(struct Node** head) {
    if (*head == NULL) {
        printf("The list is empty, nothing to delete.\n");
        return;
    }
    struct Node* temp = *head;
    *head = (*head)->next;
    printf("Deleted: %d\n", temp->data);
    free(temp);
}

void deleteLast(struct Node** head) {
    if (*head == NULL) {
        printf("The list is empty, nothing to delete.\n");
        return;
    }
    if ((*head)->next == NULL) {
```

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

void deleteSpecific(struct Node** head, int key) {
    if (*head == NULL) {
        printf("The list is empty, nothing to delete.\n");
        return;
    }
    if ((*head)->data == key) {
        struct Node* temp = *head;
        *head = (*head)->next;
```

```c
        printf("Deleted: %d\n", temp->data);
        free(temp);
        return;
    }
    struct Node* temp = *head;
    while (temp->next != NULL && temp->next->data != key) {
        temp = temp->next;
    }
    if (temp->next == NULL) {
        printf("Element %d not found in the list.\n", key);
        return;
    }
    struct Node* toDelete = temp->next;
    temp->next = temp->next->next;
    printf("Deleted: %d\n", toDelete->data);
    free(toDelete);
}

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

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

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

    int choice, value;
    while (1) {
        printf("\n1. Insert at beginning\n");
        printf("2. Insert at end\n");
        printf("3. Delete first element\n");
        printf("4. Delete last element\n");
        printf("5. Delete specific element\n");
        printf("6. Display the linked list\n");
        printf("7. Exit\n");
        printf("Enter your choice: ");
```

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

switch (choice) {
    case 1:
        printf("Enter value to insert at beginning: ");
        scanf("%d", &value);
        insertBeg(&head, value);
        break;
    case 2:
        printf("Enter value to insert at end: ");
        scanf("%d", &value);
        insertEnd(&head, value);
        break;
    case 3:
        deleteFirst(&head);
        break;
    case 4:
        deleteLast(&head);
        break;
    case 5:
        printf("Enter value to delete: ");
        scanf("%d", &value);
```

```
            deleteSpecific(&head, value);
            break;
        case 6:
            printf("Linked List: ");
            displayList(head);
            break;
        case 7:
            printf("Exiting...\n");
            exit(0);
        default:
            printf("Invalid choice, please try again.\n");
        }
    }

    return 0;
}
```

OUTPUT:

```
1. Insert at beginning
2. Insert at end
3. Delete first element
4. Delete last element
5. Delete specific element
6. Display the linked list
7. Exit
Enter your choice: 1
Enter value to insert at beginning: 10

1. Insert at beginning
2. Insert at end
3. Delete first element
4. Delete last element
5. Delete specific element
6. Display the linked list
7. Exit
Enter your choice: 1
Enter value to insert at beginning: 20

1. Insert at beginning
2. Insert at end
3. Delete first element
4. Delete last element
5. Delete specific element
6. Display the linked list
7. Exit
Enter your choice: 2
Enter value to insert at end: 30

1. Insert at beginning
2. Insert at end
3. Delete first element
4. Delete last element
5. Delete specific element
6. Display the linked list
7. Exit
Enter your choice: 6
Linked List: 20 -> 10 -> 30 -> NULL
```

```
1. Insert at beginning
2. Insert at end
3. Delete first element
4. Delete last element
5. Delete specific element
6. Display the linked list
7. Exit
Enter your choice: 3
Deleted: 20

1. Insert at beginning
2. Insert at end
3. Delete first element
4. Delete last element
5. Delete specific element
6. Display the linked list
7. Exit
Enter your choice: 6
Linked List: 10 -> 30 -> NULL

1. Insert at beginning
2. Insert at end
3. Delete first element
4. Delete last element
5. Delete specific element
6. Display the linked list
7. Exit
Enter your choice: 4
Deleted: 30

1. Insert at beginning
2. Insert at end
3. Delete first element
4. Delete last element
5. Delete specific element
6. Display the linked list
7. Exit
Enter your choice: 6
Linked List: 10 -> NULL
```

```
1. Insert at beginning
2. Insert at end
3. Delete first element
4. Delete last element
5. Delete specific element
6. Display the linked list
7. Exit
Enter your choice: 5
Enter value to delete: 10
Deleted: 10

1. Insert at beginning
2. Insert at end
3. Delete first element
4. Delete last element
5. Delete specific element
6. Display the linked list
7. Exit
Enter your choice: 6
Linked List: The list is empty.

1. Insert at beginning
2. Insert at end
3. Delete first element
4. Delete last element
5. Delete specific element
6. Display the linked list
7. Exit
Enter your choice: 7
Exiting...

Process returned 0 (0x0)   execution time : 153.277 s
Press any key to continue.
```

Write a program to implement Singly linked list with the following operations
a) Create a linked list
b) Deletion of find element specified and last element on the list
c) Display the contents of the linked list

```
#include<stdio.h>
#include <conio.h>
struct Node{
        int data;
        struct Node* next;
    };
struct Node* createNode(int data){
    struct Node* newNode = (struct Node*) malloc (size of structNode);
        newNode→ data = data;
        NewNode → next = NULL;
         return newNode;
void getd( struct Node** head_ref, int data) {
    struct Node* newNode = createNode(data);
        if(* head_ref = = NULL){
        *head_ref = newNode;
    }
    else {
    struct Node* last = * head_ref;
        while (last → next != NULL){
            last = last → next;
        }
```

```
    last -> next = newNode;
  }
  printf("Added %d to the list\n", data);
}
void deletefirst(struct Node** head_ref){
  if(*head_ref == NULL){
    printf("list is empty Nothing to delete\n");
    return;
  }
  struct Node* temp = *head_ref;
  *head_ref = (*head_ref) -> next;
  printf("Deleted first element: %d\n", temp->data);
  free(temp);
}
void delete element(struct Node** head_ref, int
    key){
  struct Node* temp = *head_ref, *prev = NULL;
  if (temp != NULL && temp->data == key){
  printf("Deleted specified element %d\n", key);
  return;
  }
  while (temp != NULL && temp -> data != key){
      prev = temp;
      temp = temp -> next;
  }
  if (temp == NULL) {
    printf("element %d not find\n");
    return;
  }
  prev -> next = temp -> next;
  printf("Deleted element: %d\n", key);
  free(temp);
}
void deletelast (struct Node** head_ref){
  if (*head_ref == NULL){
    printf("list is empty\n");
    return;
  }
  struct Node* temp = *head_ref;
  if (temp -> next == NULL){
  printf("Deleted last element %d\n", temp->data
  free (temp);
    *head_ref = NULL;
    return;
  }
  struct Node* prev = NULL;
  while (temp->next != NULL){
      prev = temp;
      temp = temp -> next;
```

```
}
printf("Deleted   last element : %d\n", temp->data);
  prev->next= NULL;
free  (temp);
 }

 void display list (struct node* node){
    if (node== NULL){
    printf ("list is Empty \n");
     return;
    }
   printf (" linked list ");
     while (node != NULL){
       print ("%d -> ", node->data);
       node = node -> next;
      }
     printf ("NULL\n");
  }
```

Naveeda. M.
11/11/2024

OUTPUT: 1. Insert at beggining
2. Insert at end
3. Delete first element
4. Delete last element
5. Delete specific element
6. Display the linked list
7. Exit
enter your choice : 1
Enter value to insert to begging : 10

1. Insert at beginning
2. Insert at end
3. Delete first element.
4. Delete last element
5. Display the linked list
6. Display the linked list.
7. Exist

Enter your choice: 2
Enter value to insert at end: 30

1. Insert at beginning
2. Insert at end
4. Delete first element.
5. Delete specific element.
6. Display the linked list.
7. Exist

Enter your choice: 6
linked list: 20->10->30->NULL

1. Insert at beginning.
2. Insert at end
3. Delete first element.
4. Delete last element
5. Delete specific element.
6. Display the linked list
7. Exist

Enter your choice: 6
Linked list: 10->30->NULL

1. Insert at beginning
2. Insert at end.
3. Delete element.
4. Delete last element.
5. Delete specific element.
7. Exist

Enter your choice: 4
Deleted: 30

Enter your choice: 6
linked list: 10->NULL

Enter your choice: 5
Enter value to deleted: 10
Deleted: 10

Enter your choice: 6
linked list: The list is empty.

Enter your choice: 7
Existing...

}

LAB PROGRAM 6 A:

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

```
#include <stdio.h>

#include <stdlib.h>


struct Node {

    int data;

    struct Node* next;

};


struct Node* createNode(int data) {

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

    if (newNode == NULL) {

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

        return NULL;

    }

    newNode->data = data;

    newNode->next = NULL;

    return newNode;

}
```

```
void append(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    struct Node* temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
}

void printList(struct Node* head) {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
```

```
}

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

void reverseList(struct Node** head) {
    struct Node* prev = NULL;
    struct Node* current = *head;
    struct Node* next = NULL;
    while (current != NULL) {
```

```
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    *head = prev;
}

void concatenateList(struct Node** head1, struct Node* head2) {
    if (*head1 == NULL) {
        *head1 = head2;
        return;
    }
    struct Node* temp = *head1;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = head2;
}

int main() {
    struct Node* head1 = NULL;
    struct Node* head2 = NULL;
```

```
append(&head1, 10);
append(&head1, 20);
append(&head1, 30);

append(&head2, 5);
append(&head2, 15);
append(&head2, 25);

printf("List 1: ");
printList(head1);
printf("List 2: ");
printList(head2);

sortList(head1);
sortList(head2);

printf("Sorted List 1: ");
printList(head1);
printf("Sorted List 2: ");
printList(head2);

concatenateList(&head1, head2);

printf("Concatenated List: ");
```

```
    printList(head1);


    reverseList(&head1);


    printf("Reversed Concatenated List: ");
    printList(head1);


    return 0;
}
```

OUTPUT:

```
List 1: 10->20->30->null

List 2: 5->15->25->null

Sorted List 1: 10->20->30->null

Sorted List 2: 5->15->25->null

Concatenated List: 10->20->30->5->15->25->null

Reversed Concatenated List: 25->15->5->30->20->10->null


Process returned 0 (0x0)   execution time : 0.000 s
Press any key to continue.
```

week 8

6a) WAP to implement Single linked list with following operations:

Sort the linked list

Reverse the linked list

Concatination of two linked list

```
#include <stdio.h>
# include <stdio.h>
typedef struct Node {
  int data;
 struct Node* next;
} Node;
Node* createNode (int data) {
     Node* new Node =
  (Node*) malloc(Size of (Node));
  newNode->data = data;
  newNode->next = NULL;
   return newNode;
 }
void insert (Node** head, int data) {
  Node* newNode = createNode (data);
  if (*head == NULL) {
  * head = newNode;
 }
  else{
  Node* temp = *head;
  while (temp -> next != NULL) {
  printf("%d -> ", head -> data);
  head = head->next;
```

```
}
printf("NULL\n");
}
void sort(Node** head){
  Node* i, *j;
  int temp;
for(i = *head; i != NULL; i = i->next){
  for(j = i->next; j != NULL; j = j->next)
  {
    if (i->data > j->data){
      temp = i->data;
      i->data = j->data;
      j->data = temp;
    }
  }
}}
void reverse(Node** head){
    Node* prev = NULL, *current = *head,
  * next = NULL;
    while (current != NULL){
    next = current->next;
    current->next = prev;
    prev = current;
    current = next;
    }
    *head = prev;
}
void concatinate(Node** head1,
  Node** head2){
    if (*head1 == NULL){
      *head = *head2;
    }else{
      Node* temp = *head1;
      while(temp->next != NULL)
        temp = temp->next;
        temp->next = *head2;

    }
}
```

Output:

list1: 10 -> 10 -> 30 -> NULL

list2: 5 -> 15 -> 25 -> NULL

Starting list1:

10 -> 20 -> 30 -> NULL

Reversing list1:

30 -> 20 -> 10 -> NULL

Concating list 1 and list 2:

30 -> 20 -> 10 -> 5 -> 15 -> 25 -> NULL

LAB PROGRAM 6 B:

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

PROGRAM:

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

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

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

void push(struct Node** top, int data) {
    struct Node* newNode = createNode(data);
    if (!newNode) return;
    newNode->next = *top;
```

```
    *top = newNode;
}

int pop(struct Node** top) {
    if (*top == NULL) return -1;
    struct Node* temp = *top;
    int poppedData = temp->data;
    *top = (*top)->next;
    free(temp);
    return poppedData;
}

int peek(struct Node* top) {
    if (top == NULL) return -1;
    return top->data;
}

int isStackEmpty(struct Node* top) {
    return top == NULL;
}

void enqueue(struct Node** front, struct Node** rear, int data) {
    struct Node* newNode = createNode(data);
    if (!newNode) return;
```

```
    if (*rear == NULL) {
        *front = *rear = newNode;
        return;
    }
    (*rear)->next = newNode;
    *rear = newNode;
}

int dequeue(struct Node** front, struct Node** rear) {
    if (*front == NULL) return -1;
    struct Node* temp = *front;
    int dequeuedData = temp->data;
    *front = (*front)->next;
    if (*front == NULL) *rear = NULL;
    free(temp);
    return dequeuedData;
}

int peekQueue(struct Node* front) {
    if (front == NULL) return -1;
    return front->data;
}

int isQueueEmpty(struct Node* front) {
```

```
    return front == NULL;
}

void printList(struct Node* head) {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

int main() {
    struct Node* stackTop = NULL;
    struct Node* queueFront = NULL;
    struct Node* queueRear = NULL;

    push(&stackTop, 10);
    push(&stackTop, 20);
    push(&stackTop, 30);
```

```
    printf("Popped from stack: %d\n", pop(&stackTop));
    printf("Top element of stack: %d\n", peek(stackTop));

    enqueue(&queueFront, &queueRear, 100);
    enqueue(&queueFront, &queueRear, 200);
    enqueue(&queueFront, &queueRear, 300);
    printf("Dequeued from queue: %d\n", dequeue(&queueFront, &queueRear));
    printf("Front element of queue: %d\n", peekQueue(queueFront));

    printf("Stack elements: ");
    printList(stackTop);
    printf("Queue elements: ");
    printList(queueFront);

    return 0;
}
```

OUTPUT:

```
Menu:
1. Push (Stack)
2. Pop (Stack)
3. Display Stack
4. Enqueue (Queue)
5. Dequeue (Queue)
6. Display Queue
7. Exit
Enter your choice: 1
Enter value to push onto stack: 10
10 pushed onto the stack.

Menu:
1. Push (Stack)
2. Pop (Stack)
3. Display Stack
4. Enqueue (Queue)
5. Dequeue (Queue)
6. Display Queue
7. Exit
Enter your choice: 1
Enter value to push onto stack: 20
20 pushed onto the stack.

Menu:
1. Push (Stack)
2. Pop (Stack)
3. Display Stack
4. Enqueue (Queue)
5. Dequeue (Queue)
6. Display Queue
7. Exit
Enter your choice: 3
Stack: 20 -> 10 -> NULL

Menu:
1. Push (Stack)
2. Pop (Stack)
3. Display Stack
4. Enqueue (Queue)
```

```
Menu:
1. Push (Stack)
2. Pop (Stack)
3. Display Stack
4. Enqueue (Queue)
5. Dequeue (Queue)
6. Display Queue
7. Exit
Enter your choice: 2
20 popped from the stack.

Menu:
1. Push (Stack)
2. Pop (Stack)
3. Display Stack
4. Enqueue (Queue)
5. Dequeue (Queue)
6. Display Queue
7. Exit
Enter your choice: 3
Stack: 10 -> NULL

Menu:
1. Push (Stack)
2. Pop (Stack)
3. Display Stack
4. Enqueue (Queue)
5. Dequeue (Queue)
6. Display Queue
7. Exit
Enter your choice: 4
Enter value to enqueue into queue: 15
15 enqueued into the queue.

Menu:
1. Push (Stack)
2. Pop (Stack)
3. Display Stack
```

```
Enter your choice: 4
Enter value to enqueue into queue: 15
15 enqueued into the queue.

Menu:
1. Push (Stack)
2. Pop (Stack)
3. Display Stack
4. Enqueue (Queue)
5. Dequeue (Queue)
6. Display Queue
7. Exit
Enter your choice: 4
Enter value to enqueue into queue: 25
25 enqueued into the queue.

Menu:
1. Push (Stack)
2. Pop (Stack)
3. Display Stack
4. Enqueue (Queue)
5. Dequeue (Queue)
6. Display Queue
7. Exit
Enter your choice: 6
Queue: 15 -> 25 -> NULL

Menu:
1. Push (Stack)
2. Pop (Stack)
3. Display Stack
4. Enqueue (Queue)
5. Dequeue (Queue)
6. Display Queue
7. Exit
Enter your choice: 5
15 dequeued from the queue.
```

```
Menu:
1. Push (Stack)
2. Pop (Stack)
3. Display Stack
4. Enqueue (Queue)
5. Dequeue (Queue)
6. Display Queue
7. Exit
Enter your choice: 6
Queue: 25 -> NULL

Menu:
1. Push (Stack)
2. Pop (Stack)
3. Display Stack
4. Enqueue (Queue)
5. Dequeue (Queue)
6. Display Queue
7. Exit
Enter your choice: 7
Exiting...

Process returned 0 (0x0)   execution time : 167.726 s
Press any key to continue.
```

b) WAP to implement Single linked list to Stimulate Stack & Queue operation:

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

typedef struct Node{
    int data;
    struct Node* next;
}
Node;
Node * createNode (int data){
    Node* newNode = (Node* ) malloc
    (sizeof(Node));
    newNode -> data = data;
    newNode -> next = NULL;
    return newNode;
}
void push (Node** top, int data){
    Node* newNode = createNode (data);
    newNode -> next = * top;
    * top = newNode;
}
void pop (Node** top ){
    if (*top == NULL){
    printf (" Stack underflow\n");
    return -1;
}
}
Node* temp = * temp;
int data = temp -> data;
* top = (*top) -> next;
free (temp);
return data;
}
void enqueue(Node** rear, Node** front, int data){
    Node* newNode = createNode( data);
    if (*rear == NULL){
        * rear = * front = newNode;
    }
    else{
        (*rear) -> next = newNode;
        * rear = newNode;
    }
}
int dequeue (Node** front){
    if (*front == NULL){
        printf(" Queue underflow\n");
        return -1;
}
Node *temp = * front;
int data = temp -> data;
* front = (*front) -> next;
    free (temp);
    return data;
}
```

OUTPUT

Output

Stack operation: 30
popped from Stack: 30
popped from Stack: 20

Queues operations:
Dequeue from queue: 100
Dequeue from queue: 200

}

LAB PROGRAM 7:

WAP to Implement doubly link list with primitive operations
a) Create a doubly linked list.
b) Insert a new node at the beginning.
c) Insert the node based on a specific location
d) Insert a new node at the end.
e) 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 data) {

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

    if (newNode == NULL) {

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

        exit(1);

    }

    newNode->data = data;
```

```
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}

void insertAtBeginning(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head != NULL) {
        (*head)->prev = newNode;
        newNode->next = *head;
    }
    *head = newNode;
    printf("%d inserted at the beginning.\n", data);
}

void insertAtEnd(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        printf("%d inserted at the end.\n", data);
        return;
    }
    struct Node* temp = *head;
    while (temp->next != NULL) {
```

```
        temp = temp->next;
    }
    temp->next = newNode;
    newNode->prev = temp;
    printf("%d inserted at the end.\n", data);
}

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

    if (position == 1) {
        insertAtBeginning(head, data);
        return;
    }

    struct Node* newNode = createNode(data);
    struct Node* temp = *head;
    for (int i = 1; i < position - 1; i++) {
        if (temp == NULL) {
            printf("Position out of bounds.\n");
            free(newNode);
```

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

    if (temp == NULL) {
        printf("Position out of bounds.\n");
        free(newNode);
        return;
    }

    newNode->next = temp->next;
    newNode->prev = temp;
    if (temp->next != NULL) {
        temp->next->prev = newNode;
    }
    temp->next = newNode;
    printf("%d inserted at position %d.\n", data, position);
}

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

```
    }

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

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

    while (1) {
        printf("\nMenu:\n");
        printf("1. Insert at the beginning\n");
        printf("2. Insert at the end\n");
        printf("3. Insert at a specific position\n");
        printf("4. Display the 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 value to insert at the end: ");
        scanf("%d", &value);
        insertAtEnd(&head, value);
        break;
    case 3:
        printf("Enter value to insert: ");
        scanf("%d", &value);
        printf("Enter position to insert: ");
        scanf("%d", &position);
        insertAtPosition(&head, value, position);
        break;
    case 4:
        displayList(head);
        break;
    case 5:
        printf("Exiting...\n");
```

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

    return 0;
}
```

OUTPUT:

```
Menu:
1. Insert at the beginning
2. Insert at the end
3. Insert at a specific position
4. Display the list
5. Exit
Enter your choice: 1
Enter data to insert at the beginning: 45

Menu:
1. Insert at the beginning
2. Insert at the end
3. Insert at a specific position
4. Display the list
5. Exit
Enter your choice: 1
Enter data to insert at the beginning: 50

Menu:
1. Insert at the beginning
2. Insert at the end
3. Insert at a specific position
4. Display the list
5. Exit
Enter your choice: 3
Enter data to insert: 78
Enter position to insert at: 1

Menu:
1. Insert at the beginning
2. Insert at the end
3. Insert at a specific position
4. Display the list
5. Exit
Enter your choice: 4
Doubly Linked List: 78 <-> 50 <-> 45 <-> NULL

Menu:
1. Insert at the beginning
2. Insert at the end
3. Insert at a specific position
4. Display the list
5. Exit
Enter your choice: 2
Enter data to insert at the end: 34

Menu:
1. Insert at the beginning
2. Insert at the end
3. Insert at a specific position
4. Display the list
5. Exit
Enter your choice: 4
Doubly Linked List: 78 <-> 50 <-> 45 <-> 34 <-> NULL
```

7a) WAP program to implement doubly linked list with primitive operations

a) Create a doubly linked list

b) Insert new node in doubly linked list

c) Insert the node based on specific location

d) Insert a new node at the end

e) Display the contents of linked list.

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

Struct Node {
    int data;
Struct node* next;
Struct node* prev;
};

Struct Node* Create Node(int data)
    Struct Node* newNode=(Struct Node*) malloc (sizeof
                (Structnode));
        newNode -> data = data;
        NewNode -> next = NULL;
        new Node -> prev = NULL;
        return newNode;
    }
void insertAtBeginning (Structnode* head, int data){
  Struct Node newNode = createNode (data);
    Struct Node* temp = * head;
```

```
if(*head != NULL) (*head)->p...
  *head = newNode;
}
Void insert at a specific position (Struct Node** head,
          int data, int position) {
Struct node* NewNode = Create Node (data)
Struct Node* temp = *head;
   if (pos == 1) {
   insert beg ( head, data);
   return;
}
for(int i=1 ; i<pos-1 && temp != NULL ; i++)
    temp = temp->next;
  printf(" position is out of bounds");
  return;
}
NewNode->next = temp->next;
  newNode->prev = temp;
 if ( temp->next != NULL) temp->next
  ->prev = new Node;
temp->next = newNode;
}
```

```
Void insert at End (Struct Node** head, int data) {
   Struct Node* newNode = Create Node(data);
   Struct Node* temp = head;
     if(*head == NULL)
        *head = newNode;
        return;
     }
   while(temp->next != NULL)
      temp = temp->next;
   temp->next = newNode;
   newNode->prev = temp;
}
void display(Struct Node* head) {
   Struct Node* temp = head;
 printf (" Doubly linked list:");
   while (temp != NULL) {
     printf("%d ->", temp->data);
     temp = temp->next;
   }
   printf (" NULL\n");
   }
```

Namratha M.
16/12/2024

Output:

Menu:

1. Insert at the beginning
2. Insert at the end
3. Insert at specific position
4. Display the list
5. Exist

Enter your choice: 1

Enter data to insert at the beginning: 25

Menu:

1. Insert at the beginning
2. Insert at the end
3. Insert at the specific position
4. Display the list
5. Exist

Enter your choice: 2

Enter data to insert at the end 52

Menu:

1. Insert at the beginning
2. Insert at the ~~position~~ end.
3. Insert at the specific position
4. Display the list
5. Exist

Enter your choice: 2

Menu:

1. Insert at the beginning
2. Insert at the end
3. Insert at the specific position
4. Display the list
5. Exist

Enter your choice: 4

Menu:

1. Insert at the beginning
2. Insert at the end
3. Insert at a specific position
4. Display the list
5. Exist

Enter your choice: 5

Existing....

}

LAB PROGRAM 8:

Write a program

a) To construct a binary Search tree.

b) To traverse the tree using all the methods i.e., in-

order, preorder and post order

c) To display the elements in the tree

PROGRAM:

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


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


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


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

```
    int elements[] = {1,2,3,4,5,6,7};
    int n = sizeof(elements) / sizeof(elements[0]);

    for (int i = 0; i < n; i++) {
        root = insert(root, elements[i]);
    }


    printf("In-order Traversal: ");
    inorder(root);
    printf("\n");


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


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

    return 0;
}
```

OUPUT:

```
In-order Traversal: 1 2 3 4 5 6 7
Pre-order Traversal: 1 2 3 4 5 6 7
Post-order Traversal: 7 6 5 4 3 2 1

Process returned 0 (0x0)   execution time : 0.016 s
Press any key to continue.
```

8a) write a programm

a) to construct a binary search tree

b) To convert the tree using all the Methods i.e, inorder, preorder, postorder, display all traversal list

a) program:

```
#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;
    new Node -> left = new Node -> 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 Inorder (Struct Node* root) {
    if (root != NULL) {
     Inorder (root -> left);
      printf ("%d", root -> data);
      inorder (root -> right);
    }

  void preorder (Struct Node* root) {
       if (root != NULL) {
     printf ("%d", root -> data);
     preorder (root -> left);
     preorder (root -> right);
   }
  }
void postorder (Struct Node* root) {
    if (root != NULL) {
     postorder (root -> left);
     postorder (root -> Right);
     printf ("%d", root -> data);
    }
  }

  int main () {
     Struct Node* root = NULL;
       int element [] = {50, 30, 20, 40, 70, 60, 80, };
       int n = Sizeof (elements) / Sizeof (elements [0]);
       for (int i = 0; i < n; i++) {
```

```
    root = insert (root, elements [i]);
}

printf("In-order Traversal: ");
 inorder (root);
printf("\n");

printf("pre-order Traversal: ");
 preorder (root);
 printf("\n");

printf("post-order Traversal: ");
 postorder (root);
  printf("\n");
 return 0;
}
```

Output:

In-order traversal : 20, 30, 40, 50, 60, 70, 80

pre-order traversal : 50, 30, 20, 40, 70, 60, 80

post-order traversal : 20, 40, 30, 60, 80, 70, 50,

LAB PROGRAM 9:

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

5 b) Write a program to check whether given graph is connected or not using DFS method

PROGRAM:

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

#define MAX 100

struct Queue {
   int items[MAX];
   int front;
   int rear;
};

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

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

void enqueue(struct Queue* q, int value) {
   if (q->rear == MAX - 1) {
      printf("Queue is full\n");
      return;
   }
   if (q->front == -1) q->front = 0;
   q->rear++;
```

```
    q->items[q->rear] = value;
}

int dequeue(struct Queue* q) {
    if (isEmpty(q)) {
        printf("Queue is empty\n");
        return -1;
    }
    int value = q->items[q->front];
    if (q->front == q->rear) {
        q->front = q->rear = -1;
    } else {
        q->front++;
    }
    return value;
}

void BFS(int graph[MAX][MAX], int start, int n) {
    bool visited[MAX] = {false};
    struct Queue q;
    initQueue(&q);
    enqueue(&q, start);
    visited[start] = true;

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

        for (int i = 0; i < n; i++) {
            if (graph[node][i] == 1 && !visited[i]) {
                enqueue(&q, i);
                visited[i] = true;
            }
        }
    }
}
```

```
void DFS(int graph[MAX][MAX], int node, bool visited[MAX], int n) {
    visited[node] = true;
    printf("%d ", node);

    for (int i = 0; i < n; i++) {
        if (graph[node][i] == 1 && !visited[i]) {
            DFS(graph, i, visited, n);
        }
    }
}
int main() {
    int n = 6;
    int graph[MAX][MAX] = {
        {0, 1, 1, 0, 0, 0},
        {1, 0, 1, 1, 0, 0},
        {1, 1, 0, 1, 0, 0},
        {0, 1, 1, 0, 1, 1},
        {0, 0, 0, 1, 0, 1},
        {0, 0, 0, 1, 1, 0}
    };

    printf("BFS starting from node 0: ");
    BFS(graph, 0, n);
    printf("\n");
    bool visited[MAX] = {false};
    printf("DFS starting from node 0: ");
    DFS(graph, 0, visited, n);
    printf("\n");


    return 0;
}
```

OUTPUT:

```
BFS starting from node 0: 0 1 2 3 4 5
DFS starting from node 0: 0 1 2 3 4 5

Process returned 0 (0x0)   execution time : 0.000 s
Press any key to continue.
```

9th (a) Write program to traverse a graph.

(b) Write a program to traverse a graph using DFS metho

program:

```
#include <stdio.h>
#include < stdlib.h>
#include <stdbool.h>
#define MAX 100
struct queue{
    int items[MAX];
    int front;
    int rear;
void init queue(struct queue* q) {
    q->front = -1;
   q. rear = -1;
}
int isEmpty (struct queue* q) {
    return q-> front = = -1;
void enqueue (struct queue *q, int value) {
        if (q-> rear = = MAX - 1) {
        printf ("queue is full \n");
        return;
      }
      if (q-> front = = -1) q-> front = 0.
      q-> rear++;
      q-> items [q-> rear] = value;
int dequeue (struct queue *q) {
```

```
if (isEmpty(q)) {
  printf("Queue is Empty \n");
  return -1;
}
int value = q->items[q->front];
if (q->front == q->rear) {
  q->front = q->rear = -1;
}
else {
  q->front++;
}
return value;
}
void BFS (int graph[MAX][MAX], int start, int n) {
bool visited[MAX] = {false};
  struct Queue q;
  initializeQueue(&q);
enqueue(&q, start);
visited[start] = true;
while (!isEmpty(&q)) {
  int node = dequeue(&q);
  printf("%d", node);
  for (int i = 0; i < n; i++) {
  if (graph[node][i] == 1 && !visited[i]) {
      enqueue(&q, i);
      visited[i] = true;
    }
  }
}
}
void DFS(int graph[MAX][MAX], int node, bool visited
       [MAX], int n) {
  Visited[node] = true;
  printf("%d", node);
  for (int i = 0; i < n; i++) {
   if (graph[node][i] == 1 && !visited[i]) {
       DFS(graph, i, visited, n);
     }
   }
}
int main() {
  int n = 6;
  int graph[MAX][MAX] = {
      {0, 1, 1, 0, 0, 0},
      {1, 0, 1, 1, 0, 0},
      {1, 1, 0, 1, 0, 0},
      {0, 1, 1, 0, 1, 1},
      {0, 0, 0, 1, 0, 1},
      {0, 0, 0, 1, 1, 0}
```

```
printf ("BFS Starting from node 0: ");
BFS(graph 0, n);
printf ("\n");

bool visited [MAX] = {false};
printf ("DFS Starting from node 0: ");
DFS (graph, 0, visited, n);
printf ("\n");

return 0;
}
```

Output:

BFS Starting from node 0: 0, 1, 2, 3, 4, 5
DFS Starting from node 0: 0 1 2 3 4 5

-:COMPLETE:-