VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Data Structures using C Lab

(23CS3PCDST)

Submitted by

Shreya Sathyanarayana (1BM23CS318)

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

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
Sep-2024 to Jan-2025

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

CERTIFICATE

This is to certify that the Lab work entitled "Data Structures using C Lab (23CS3PCDST)" carried out by **Shreya Sathyanarayana (1BM23CS318),** who is bona fide student of **B.M.S. College of Engineering.** It is in partial fulfilment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum. The Lab report has been approved as it satisfies the academic requirements in respect of Data Structures using C Lab (23CS3PCDST) work prescribed for the said degree.

Namratha S Assistant Professor Department of CSE, BMSCE	Dr. Kavitha Sooda Professor &HOD Department of CSE, BMSCE

Index

Sl. No.	Date	Experiment Title	Page No.
1	30-09-24	working of stack using an array	4-6
2	07-10-24	convert a given valid parenthesized infix arithmetic expression to postfix expression	7-9
3	14-10-24 21-10-24	A. working of a queue of integers using an array B. working of a circular queue of integers using an array	10-12 13-15
4	28-10-24	Implement Singly Linked List with following operations a) Create a linked list. b) Insertion of a node at first position, at any position and at end of list. Display the contents of the linked list.	16-20
5	11-10-24	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.	21-25
6	2-12-24	A. Implement Single Link List with following operations: Sort the linked list, Reverse the linked list, Concatenation of two linked lists. B. Implement Single Link List to simulate Stack & Queue Operations.	25-29 29-32
7	16-12-24	Implement doubly link list with primitive operations	33-37
8	23-12-24	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.	
9	23-12-24	A. traverse a graph using BFS method. B. check whether given graph is connected or not using DFS method.	

Github Link:
https://github.com/shreyasathyanarayana1/DS-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

Code:

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

int top=-1;
int stack[size];
int item;

void push(){
   if(top==size-1){
      printf("Stack Overload\n");
   }
   else{
      top+=1;
      stack[top]=item;
   }
}

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

void display(){
   if(top==-1){
      printf("Stack is empty!");
   }
   else{
```

```
        printf("Content of the stacks:");
        for(int i=0;i<=top;i++){
            printf("%d ",stack[i]);
        }printf("\n");
    }
}

void main(){
    int choice;
    while(1){
        printf("Enter your options:\n");
        printf("1.Push\n2.Pop\n3.Display\n4.Exit\n");
        printf("Enter your choice:");
        scanf("%d",&choice);
        switch(choice){
            case 1:printf("Enter the element to be pushed in:");scanf("%d",&item);push();break;
            case 2:if(top==-1){
                printf("stack is empty!\n");
            }else{
                printf("%d popped from stack\n", stack[top]);
            }
            pop();
            break;
            case 3:display();
            break;
            case 4:exit(0);
        }
    }
}
```

```
Enter your choice:1
Enter the element to be pushed in:45
Enter your options:
1.Push
2.Pop
3.Display
4.Exit
Enter your choice:1
Enter the element to be pushed in:67
Enter your options:
1.Push
2.Pop
3.Display
4.Exit
Enter your choice:3
Content of the stacks:45 67
Enter your options:
1.Push
2.Pop
3.Display
4.Exit
Enter your choice:2
67 popped from stack
Enter your options:
1.Push
2.Pop
3.Display
4.Exit
Enter your choice:
```

```
Q  Implement stacks & ops using arrays

#include <stdbool.h>
#include <stdio.h>
  int top=-1, size, ~~10~~;
  void init()
    {
     printf ("Enter size of stack : \n");
     scanf ("%d", &size);
     top = -1;
    }
  void push (int[] arr, int x) {
    if (! is full())
      {
       top ++;
       arr [top] = x;
       printf ("Pushed %d to stack \n", x);
      }
    else
      {
       printf ("Overflow \n");
      }
    }
  int pop (int arr[])
    {
     if (is Empty())
      {
       printf ("underflow \n");
       return 0;
      }
    else
      {
```

```
store
67

int temp = arr[top];
  top--;
  return temp;
  }
 }
int top (int arr[])
  {
   return is Empty()? top:0;
  }
bool isempty()
   {
    return top = size-1;
   }
void main()
   {
    init();
    int arr [size];
    printf ("Enter element:");
    for (i=0; i<size, i++)
      {
       printf(" 1:Push 2:Pop");
       printf("3: Display 4: Exit");
       printf ("Enter the choice");
       scanf ("%d", &choice);
       switch (choice)
       {
        case 1: printf ("Enter the item to be inserted:")
        scanf ("%d", &item);
              push ();
              break;
```

```
case 2: item_deleted = pop();
        if (item_deleted == -1)
            pf ("Stack is empty");
        else
            pf ("%d", item_deleted);

case 3: display()
        break;

default: exit (0);

  }
 }
}
```

Seen

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)

Code:

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

#define MAX 5

char stack[MAX];
int top = -1;

void push(char c) {
    if (top < MAX - 1) {
        stack[++top] = c;
    }
}
char pop() {
    if (top >= 0) {
        return stack[top--];
    }
    return '\0';
}
char peek() {
    if (top >= 0) {
        return stack[top];
    }
    return '\0';
}

int precedence(char c) {
    switch (c) {
        case '+': return 1;
        case '-': return 1;
        case '*': return 2;
        case '/': return 2;
        case '^': return 3;
        default: return 0;
    }
}
```

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


void infixToPostfix(const char *infix, char *postfix) {
    int i = 0, j = 0;
    while (infix[i]) {
        if (isalnum(infix[i])) {
            postfix[j++] = infix[i];
        } else if (infix[i] == '(') {
            push(infix[i]);
        } else if (infix[i] == ')') {
            while (top != -1 && peek() != '(') {
                postfix[j++] = pop();
            }
            pop();
        } else if (isOperator(infix[i])) {
            while (top != -1 && precedence(peek()) >= precedence(infix[i])) {
                postfix[j++] = pop();
            }
            push(infix[i]);
        }
        i++;
    }
    while (top != -1) {
        postfix[j++] = pop();
    }
    postfix[j] = '\0';
}

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

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

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

    return 0;
}
```

```
Enter an infix expression: abcd^e-fgh*+^*+i-
Postfix expression: abcde^fgh*-^*+i+-
```

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

```
#include <stdio.h>
#include <stdlib.h>
#include <string.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");
  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==')')
    {
      while (stackIndex >=0 && stack [stackIndex]!
        =')')
        {
          result [resultIndex++] = stack [stackIndex--];
        }
      stackIndex--;
    }
  else
    {
      while (stackIndex >=0 && (prec(c) < prec(stack[stackIndex])
      || prec(c)==prec (stack [stackIndex]) &&
      associativity (c) =='L'))
```

67

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

read from user

Output:

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

Nandana M
7/4/2024

Program 3a

Write A Program 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

Code:

```
#include <stdio.h>
#define max_size 4
int queue [max_size];
int front =-1;
int rear=-1;

void insert(int value){
   if (rear==max_size -1){
      printf("Queue overflow! Cannot insert elements");
   }
   else{
      if(front == -1){
         front =0;
      }

      queue[++rear]=value;
      printf("Insert %d into queue",value);
   }
}

void delete(){
   if(front==-1 || front>rear){
      printf("Queue underflow!Cannot delete ");
   }
   else{
      printf("Deleted %d from the queue",queue[front]);
      front++;
   }
}

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

```
    }
}

int main(){
    int choice, value;
    while(1){
        printf("\n1.Insert");
        printf("\n2.Delete");
        printf("\n3.Display");
        printf("\n4.Exit");
        printf("\nEnter your choice:");
        scanf("%d",&choice);
        switch (choice){
            case 1: printf("Enter a value to insert:");
                    scanf("%d",&value);
                    insert(value);
                    break;
            case 2: delete();
            break;
            case 3: display();
            break;
            case 4: return 0;
            default: printf("Invalid choice! Please try again\n");
        }
    }
```

```
Enter your choice:1
Enter a value to insert:4
Insert 4 into queue
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice:1
Enter a value to insert:5
Insert 5 into queue
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice:1
Enter a value to insert:9
Insert 9 into queue
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice:3
Queue Elements
4 5 9

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

Simulate the working of a queue of integers using an array.

```
int queue [MAX];
int front, rear = -1;
int isfull()
  {
  return rear == max-1;
  }
int isempty()
  {
  return front == -1; || front > rear;
  }

void enqueue (int value)
  {
  if (isfull())
  {
  printf ("Queue overflow")
  }
  if (front == -1)
  {
  front = 0;
  }
  rear ++;
queue [rear] = value;
pf ("%d inserted to queue", value);
  }
```

pass by parameters

67

```
void dequeue
  {
  if (is empty())
  {
  pf("Queue is empty");
  }
  pf ("%d deleted from queue", queue [front]);
  front ++;
```

```
if front > rear
  {
  front = rear = -1;
  }
  }
  void display()
    {
    if (isempty())
    {
    pf(" Queue is empty")
    }
    pf(" Queue elements:")
    for (int i = front ; i <= rear; i++)
    {
    pf ("%d", queue [i])
    }
  int main()
    {
    int choice, value;
  do {
  printf(" Queue operations:"
```

```
printf(" 1. insert")
pf ("2. delete")
pf ("3. display")
pf ("4. exit")
pf ("Enter your choice")
sf ("%d", &choice);

switch (choice)
  {
  case 1: pf ("Enter value to insert");
          sf ("%d", &value);
          enqueue (value);
          break;
  case 2: dequeue ();
          break;

  case 3: display();
          break;
  case 4: pf ("Exiting")
          break;
  default: pf ("Invalid choice");
  }

while (choice != 4)
  return 0;
  }
```

execute
Namdev. M.
19/10/24

o/p seen
19/10/24

store
67

Output:

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

Enter your choice: 1
Enter the value to insert: 3
3 inserted into the queue

Enter your choice: 2
3 deleted from queue

Enter your choice: 3
Queue elements are: 4 5 9

Enter your choice: 4
Exiting...

Enter value to insert: 9
Queue overflow! cannot insert 9.
```

display
insertion
deletion
display
exit
overflow

Program 3b

Write A Program 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

Code:

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

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

int isFull() {
   return (front == (rear + 1) % MAX);
}

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

void insert(int value) {
   if (isFull()) {
      printf("Queue Overflow: Unable to insert %d\n", value);
      return;
   }
   if (isEmpty()) {
      front = 0; // Set front to 0 if the queue is empty
   }
   rear = (rear + 1) % MAX;
   queue[rear] = value;
   printf("Inserted %d into the queue\n", value);
}

void delete() {
   if (isEmpty()) {
      printf("Queue Underflow: Unable to delete from the queue\n");
      return;
   }
   int deletedValue = queue[front];
   if (front == rear) {
      front = -1; // Queue becomes empty
      rear = -1;
   } else {
      front = (front + 1) % MAX;
```

```
    }
    printf("Deleted %d from the queue\n", deletedValue);
}

void display() {
    if (isEmpty()) {
        printf("Queue is empty\n");
        return;
    }
    printf("Queue elements: ");
    int i = front;
    while (1) {
        printf("%d ", queue[i]);
        if (i == rear) break;
        i = (i + 1) % MAX;
    }
    printf("\n");
}

int main() {
    int choice, value;

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

        switch (choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                insert(value);
                break;
            case 2:
                delete();
                break;
            case 3:
                display();
                break;
            case 4:
                exit(0);
            default:
                printf("Invalid choice. Please try again.\n");
```

```
        }
    }

    return 0;
}
```

```
Circular Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter value to insert: 85
Inserted 85 into the queue

Circular Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter value to insert: 56
Inserted 56 into the queue

Circular Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
Queue elements: 85 56

Circular Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Deleted 85 from the queue
```

WAP to simulate the working of a circular queue of integers using an array. Provide the fol. operations insert, delete, display.
The program should display appropriate messages for queue empty & queue overflow conditions.

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

#define SIZE 5

int queue[SIZE], front = 1, rear = -1;   pass by parameters

int isfull() {
    if((front == 0 && rear == SIZE-1)||(front == rear+1))
    {
    return 1;
    }
    return 0;
    }

int isempty() {
        if(front==-1) {
            return 1;
        }
        return 0;
    }
void insert (int element) {
    if (isFull()) {
    printf("Queue overflow!!");
```

```
return;
}
if(front == -1)
    {
    front = 0;
    }
rear = (rear + 1) % SIZE;
queue[rear] = element;
printf("Inserted %d ", element);
}
Void delete()
{
    if (isEmpty()) {
    printf("Queue Underflow!!");
        return;
    }
    printf("Deleted %d ", queue[front]);
    if (front == rear) {
        front = rear = -1;
    }
    else {
        front = (front + 1) % SIZE;
    }
    }
void display() {
        if (isempty()){
            printf("Queue is empty");
            return;
        }
        int i = front;
        printf("Queue elements:");
        while(1) {
```

```
printf("%d", queue[i]);
    if (i == rear) {
        break;
    }
    i = (i+1) % SIZE;

int main() {
    int choice, element;

    while(1) {
        pf("\n Circular Queue Operations \n");
        pf("Insert");
        pf("Delete");
        pf("Display");
        pf("Exit");
        pf("Enter your choice");
        sf("%d", &choice);

switch (choice){
    case 1: printf("Enter the element to insert");
            sf("%d", &element);
            insert(element);
            break;
    case 2:
            delete();
            break;
    case 3:
            display();
            break;
```

store
67

```
    Case 4:
        exit();
    default:
        printf("Invalid choice");
    }
}
return 0;
}
```

Write & execute
Nandeesha M.
24/10/2021

Output

Circular queue operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter the choice: 1
Enter the element to insert: 3
Inserted 3.

Circular queue operations:
1. Insert
2. Delete
3. display
4. Exit
Enter your choice: 1
Enter the element to insert: 4
Inserted 4

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

Circular Queue operations
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
Queue elements: 3 4

Circular Queue operations
1. Insert
2. Delete
3. Display
4. Exit
Enter your Choice. 4

Program 4

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.

Code:

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

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

void createList(struct Node** head);
void insertAtBeginning(struct Node** head, int data);
void insertAtPosition(struct Node** head, int data, int position);
void insertAtEnd(struct Node** head, int data);
void displayList(struct Node* head);

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

   while (1) {
      printf("\nMenu:\n");
      printf("1. Create a linked list\n");
      printf("2. Insert at the beginning\n");
      printf("3. Insert at a specific position\n");
      printf("4. Insert at the end\n");
      printf("5. Display the list\n");
      printf("6. Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);

      switch (choice) {
         case 1:
            createList(&head);
            break;
         case 2:
            printf("Enter data to insert at the beginning: ");
            scanf("%d", &data);
            insertAtBeginning(&head, data);
            break;
         case 3:
```

```
            printf("Enter data to insert: ");
            scanf("%d", &data);
            printf("Enter position to insert (starting from 1): ");
            scanf("%d", &position);
            insertAtPosition(&head, data, position);
            break;
        case 4:
            printf("Enter data to insert at the end: ");
            scanf("%d", &data);
            insertAtEnd(&head, data);
            break;
        case 5:
            displayList(head);
            break;
        case 6:
            printf("Exiting the program.\n");
            exit(0);
        default:
            printf("Invalid choice. Please try again.\n");
    }
}

return 0;
}

void createList(struct Node** head) {
    int data, choice;
    do {
        printf("Enter data to insert: ");
        scanf("%d", &data);
        insertAtEnd(head, data);
        printf("Do you want to add another node? (1 for Yes, 0 for No): ");
        scanf("%d", &choice);
    } while (choice != 0);
}

void insertAtBeginning(struct Node** head, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed.\n");
        return;
    }
    newNode->data = data;
    newNode->next = *head;
    *head = newNode;
    printf("Node inserted at the beginning.\n");
}
```

```
void insertAtPosition(struct Node** head, int data, int position) {
    if (position < 1) {
        printf("Invalid position.\n");
        return;
    }

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

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

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

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

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

void insertAtEnd(struct Node** head, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed.\n");
        return;
    }
    newNode->data = data;
    newNode->next = NULL;

    if (*head == NULL) {
```

```
        *head = newNode;
    } else {
        struct Node* temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
    printf("Node inserted at the end.\n");
}

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

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

```
Menu:
1. Create a linked list
2. Insert at the beginning
3. Insert at a specific position
4. Insert at the end
5. Display the list
6. Exit
Enter your choice: 2
Enter data to insert at the beginning: 12
Node inserted at the beginning.

Menu:
1. Create a linked list
2. Insert at the beginning
3. Insert at a specific position
4. Insert at the end
5. Display the list
6. Exit
Enter your choice: 2
Enter data to insert at the beginning: 23
Node inserted at the beginning.

Menu:
1. Create a linked list
2. Insert at the beginning
3. Insert at a specific position
4. Insert at the end
5. Display the list
6. Exit
Enter your choice: 3
Enter data to insert: 2
Enter position to insert (starting from 1): 2
Node inserted at position 2.
```

```
Menu:
1. Create a linked list
2. Insert at the beginning
3. Insert at a specific position
4. Insert at the end
5. Display the list
6. Exit
Enter your choice: 5
Linked list contents: 23 -> 2 -> 12 -> NULL

Menu:
1. Create a linked list
2. Insert at the beginning
3. Insert at a specific position
4. Insert at the end
5. Display the list
6. Exit
Enter your choice: 6
Exiting the program.
```

3. Implement a singly linked list with operations insert [beginning, end], display contents

```
#include <stdio.h>
#include <stdlib.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 insert at beginning (Struct Node ** head, int data){
    struct Node* newNode = createNode (data);
    newNode -> next = *head;
    *head = newNode;
}

void insert_at_position (struct Node ** head, int data, int pos){
    if (pos == 1) {
        newNode -> next = *head;
        *head = newNode;
        return;
    }
    Struct Node* temp = *head;
```

```
for (int i = 1; i < pos-1 && temp != NULL; i++) {
        temp = temp -> next;
    }
    if (temp == NULL){
        printf ("Position out of range \n");
    }
    else {
        newNode -> next = temp -> next;
        temp -> next = newNode;
    }
}

void insert_at_end (struct Node** head, int data){
    Struct node* newNode = create Node (data);
    if (head == NULL){
        *head = newNode;
        return;
    }
    Struct Node * temp = *head
    while (temp -> next != NULL){
        temp = temp -> next;
    }
    temp -> next = newNode;
}

void display (struct Node * head){
    if (head == NULL) {
        printf ("Empty list! \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;

insert_at_beginning (&head, 6);
    display (head);

insert_at_pos (&head, 15, 3);
    display (head);

insert_at_end (&head, 30);
    display (head);

    return 0;
}
```

execute
Nmadkar
28/10/2025

Output:
Menu:
1. Insert at beginning
2. Insert at end
3. Display the list
4. Exit

```
Enter your choice: 1
Enter Value to insert at the beginning: 2

Enter your choice: 2
Enter value to insert at the end: 3

Enter your choice: 3
~~Enter value to insert at the beginning~~
linked list: 2 -> 3 -> NULL

Enter your choice: 4
Exiting program
```

Program 5

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.

Code:

```
#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));
   newNode->data = data;
   newNode->next = NULL;
   return newNode;
};

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

void deleteFirst(struct Node** head) {
   if (*head == NULL) {
      printf("The list is empty.\n");
      return;
   }
   struct Node* temp = *head;
   *head = (*head)->next;
   free(temp);
}

void deleteElement(struct Node** head, int key) {
   if (*head == NULL) {
      printf("The list is empty.\n");
      return;
   }
   struct Node *temp = *head, *prev = NULL;
   if (temp != NULL && temp->data == key) {
      *head = temp->next;
```

```
        free(temp);
        return;
    }
    while (temp != NULL && temp->data != key) {
        prev = temp;
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Element %d not found.\n", key);
        return;
    }
    prev->next = temp->next;
    free(temp);
}

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

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("\nMenu:\n");
      printf("1. Insert element at the end\n 2. Delete first element\n 3.Delete specified element\n
4.Delete last element\n 5.Display list\n 6.Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);

      switch (choice) {
         case 1:
            printf("Enter value to insert: ");
            scanf("%d", &value);
            insertatfirst(&head, value);
            break;
         case 2:
            deleteFirst(&head);
            break;
         case 3:
            printf("Enter value to delete: ");
            scanf("%d", &value);
            deleteElement(&head, value);
            break;
         case 4:
            deleteLast(&head);
            break;
         case 5:
            displayList(head);
            break;
         case 6:
            exit(0);
         default:
            printf("Invalid choice.\n");
      }
   }
   return 0;
}
```

```
Menu:
1. Insert element at the end
 2. Delete first element
 3.Delete specified element
 4.Delete last element
 5.Display list
 6.Exit
Enter your choice: 2

Menu:
1. Insert element at the end
 2. Delete first element
 3.Delete specified element
 4.Delete last element
 5.Display list
 6.Exit
Enter your choice: 5
38 -> 23 -> 14 -> NULL

Menu:
1. Insert element at the end
 2. Delete first element
 3.Delete specified element
 4.Delete last element
 5.Display list
 6.Exit
Enter your choice: 4
```

```
Menu:
1. Insert element at the end
 2. Delete first element
 3.Delete specified element
 4.Delete last element
 5.Display list
 6.Exit
Enter your choice: 1
Enter value to insert: 38

Menu:
1. Insert element at the end
 2. Delete first element
 3.Delete specified element
 4.Delete last element
 5.Display list
 6.Exit
Enter your choice: 1
Enter value to insert: 45

Menu:
1. Insert element at the end
 2. Delete first element
 3.Delete specified element
 4.Delete last element
 5.Display list
 6.Exit
Enter your choice: 5
45 -> 38 -> 23 -> 14 -> NULL
```

```
Menu:
1. Insert element at the end
 2. Delete first element
 3.Delete specified element
 4.Delete last element
 5.Display list
 6.Exit
Enter your choice: 5
38 -> 23 -> NULL

Menu:
1. Insert element at the end
 2. Delete first element
 3.Delete specified element
 4.Delete last element
 5.Display list
 6.Exit
Enter your choice: 3
Enter value to delete: 23

Menu:
1. Insert element at the end
 2. Delete first element
 3.Delete specified element
 4.Delete last element
 5.Display list
 6.Exit
Enter your choice: 5
38 -> NULL
```

11/11/24

WAP to implement singly linked list with the fol. operations:

a) Create a linked list
b) Deletion of first element, specified element & last element in the list.
c) Display the contents of the linked list

```
#include <stdio.h>
#include <stdlib.h>
struct Node{
    int data;
    struct node* next;
};
struct Node* create Node(int data){
    struct Node* newNode = (struct Node*) malloc (size of struct node);

    newNode -> data = data;
    newNode -> next = NULL;
    return newNode;
}
void delete FromFist (struct Node** head){
    if (*head == NULL){
    printf("List is empty\n");
    return;
    }
struct Node* temp = *head;
 *head = temp -> next;
 free (temp);
}
Void delete From End (struct Node **head){
    if(*head == NULL){
    printf("List is empty");
    return;
}
```

store
67

```
struct node* temp = *head;
 if (temp -> next == NULL){
  free(temp);
  *head = NULL;
  return;
}
while (temp -> next -> next != NULL){
temp = temp -> next;
}
free (temp -> next);
temp -> next = NULL;
}
void delete at Position (struct Node** head, int position){
 if (*head == NULL){
 printf("List is empty");
}
struct Node* temp = *head;
 if (position == 0){
 delete FromFirst (head);
 return;
}
for(int i=0; temp != NULL && i<position-1; i++){
 temp = temp -> next;
}
if (temp == NULL || temp -> next == NULL){
printf("Position out of range");
return;
}
struct Node* next = temp -> next -> next;
free (temp -> next)
temp -> next = next;
}
```

```
void insert Atlist (struct Node ** head, int data){
    struct Node* newNode = create Node (data);
    newNode -> next = *head;
    *head = newNode;
}
void insert At Position (struct Node ** head, int position,
                int data){
    struct Node* newNode = (reate Node (data);
        if position == 0 ){
            insert At first (head, data);
            return;
    }
    struct Node* temp = *head;
    For(int i= 0; temp != NULL && i< position -1, i++){
    temp = temp -> next;
    }
    if (temp == NULL){
        printf("Position out of range");
        free(new Node);
    }
    newNode -> next = temp -> next;
    temp -> next = newNode;
}
void insert At End (struct Node * head, int data){
struct Node* newNode = create Node (data);
 if (*head == NULL){
    *head = newNode;
    }
}
struct Node * temp = *head;
while (temp -> next != NULL){
    temp = temp -> next;
```

```
temp -> next = newNode;

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

int main() {
    struct Node* head = NULL;
    int choice, data;
    while(1) {
        printf("Linked list operations:");
        printf("1.Insertion at first position\n 2. Insertion at
                specified position\n 3. Insertion at end\n 4. Display
                \n 5. Deletion at end\n");
        pf("Enter your choice")
        scanf("%d", &choice);
        switch(choice) {
            case 1: printf("Enter element to be inserted:\n")
                    scanf("%d", &data);
                    insertAtFirst(&head, data);
                    break;
            case 2: pf("Enter element to be inserted:\n");
                    sf("%d", &data);
                    insertAtPosition(&head, position, data);
                    break;
```

```
            case 3: printf("Enter element to insert at the end:");
                    sf("%d", &data);
                    insertAtEnd(&head, data);
                    break;
            case 4: printf();
                    break;

            case 5: Printf("Enter element to be deleted:");
                    Scanf("%d", &data);
                    deleteAtFirst(&head, data);
                    break;

            case 6: printf("Enter element to be deleted:");
                    sf("%d", &data);
                    deleteAtposition(&head, position, data);
                    break;

            case 7: printf("Enter element to be deleted:");
                    scanf("%d", &data);
                    deleteAtEnd(&head, data);
                    break;

            default: printf("Invalid choice");
        }
    }
}
```

Namita M
1/1/2024

```
Linked list Operations:
  1. Insertion at First Position
  2. Insertion at Specified Position
  3. Insertion at End
  4. Display
  5. Deletion at first position
  6. Deletion at specified position.
  7. Deletion at end.
Enter your choice: 1
Enter element to be inserted 12

Enter your choice: 2
Enter position and element to be inserted: 2 14

Enter your choice: 3
Enter element to be inserted at the end: 9

Enter your choice: 5

Enter your choice: 4
12 -> 14 -> 9 -> NULL

Enter your choice: 6
Enter position to delete: 2

Enter your choice: 7
```

Namita M
1/1/2024

Program 6a

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

Code:

```
#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));
   if (!newNode) {
      printf("Memory error\n");
      return NULL;
   }
   newNode->data = data;
   newNode->next = NULL;
   return newNode;
}

void insertNode(Node** head, int data) {
   Node* newNode = createNode(data);
   if (*head == NULL) {
      *head = newNode;
      return;
   }
   Node* lastNode = *head;
   while (lastNode->next) {
      lastNode = lastNode->next;
   }
   lastNode->next = newNode;
}

void printlist(Node* head) {
   Node* current = head;
   while (current) {
      printf("%d -> ", current->data);
      current = current->next;
   }
   printf("NULL\n");
```

```
}

void sortlist(Node* head) {
    if (head == NULL) {
        return;
    }
    Node* current;
    Node* nextNode;
    int temp;
    for (current = head; current != NULL; current = current->next) {
        for (nextNode = current->next; nextNode != NULL; nextNode = nextNode->next) {
            if (current->data > nextNode->data) {
                temp = current->data;
                current->data = nextNode->data;
                nextNode->data = temp;
            }
        }
    }
}

void reverselist(Node** head) {
    Node* prev = NULL;
    Node* current = *head;
    Node* next = NULL;
    while (current) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    *head = prev;
}

void concatenate(Node** head1, Node* head2) {
    if (*head1 == NULL) {
        *head1 = head2;
        return;
    }
    Node* lastNode = *head1;
    while (lastNode->next) {
        lastNode = lastNode->next;
    }
    lastNode->next = head2;
}

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

```
    Node* head2 = NULL;

    insertNode(&head1, 1);
    insertNode(&head1, 3);
    insertNode(&head1, 5);

    insertNode(&head2, 2);
    insertNode(&head2, 4);
    insertNode(&head2, 6);

    printf("Linked list 1: ");
    printlist(head1);

    printf("Linked list 2: ");
    printlist(head2);

    sortlist(head1);
    printf("Sorted Linked list 1: ");
    printlist(head1);

    reverselist(&head2);
    printf("Reversed Linked list 2: ");
    printlist(head2);

    concatenate(&head1, head2);
    printf("Concatenated Linked list: ");
    printlist(head1);

    return 0;
}
```

```
Linked list 1: 1 -> 3 -> 5 -> NULL
Linked list 2: 2 -> 4 -> 6 -> NULL
Sorted Linked list 1: 1 -> 3 -> 5 -> NULL
Reversed Linked list 2: 6 -> 4 -> 2 -> NULL
Concatenated Linked list: 1 -> 3 -> 5 -> 6 -> 4 -> 2 -> NULL
```

store
67 ____/____

Wap to implement a Singly linked list with the fol. operations

Sort the linked list
Reverse the linked list
Concatinating 2 linked list.

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

Node* createNode (int data){
    Node* newNode = (Node*) malloc(sizeof (Node));
    if(!newNode){
        printf("Memory error \n");
        return NULL;
    }
    newNode -> data = data
    newNode -> next = NULL
    return newNode;
}
void insertNode (Node** head, int data){
    Node* newNode = createNode (data);
    if (*head == NULL){
        *head = newNode;
        return;
    }
    Node* lastNode = *head;
while (lastNode -> next){
    lastNode = lastNode -> next;
}
lastNode -> next = newNode;
}
void printlist(Node* head){
    while head{
    printf("%d->", head -> data);
    head = head -> next;
    }
    printf (NULL \n");
}
void sortList (Node* head){
    Node* current = head;
    int temp;
    if (head == NULL){
        return;
    }
    while (current){
        Node* nextNode = current -> next;
        while (nextNode){
            if (current -> data > nextNode -> data){
                temp = current -> data;
                current -> data = nextNode -> data
                nextNode -> data = temp;
            }
            nextNode = nextNode -> next;
        }
        current = current -> next;
    }
}
```

67 ____/____

```
void reverseList (Node** head){
    Node* prev = NULL;
    Node* current = *head;
    Node* next = NULL;
    while (current){
        next = current -> next;
        current -> next = prev;
        prev = current;
        current = next;
    }
    *head = prev;
}

void concatenate (Node** head1, Node* head2){
    if (head1 == NULL){
        *head1 = head2;
        return;
    }
    Node* lastNode = *head1;
    while (lastNode -> next){
        lastNode = lastNode -> next;
    }
    lastNode -> next = head2;
}

int main()
    Node* head1 = NULL;
    Node* head2 = NULL;
    insertNode(&head1, 1);
            (&head1, 3);
            (&head1, 5);
    insertNode(&head2, 2);
            (&head2, 4);
            (&head2, 6);
    printf("Linked list 1:");
    printlist (head1);
    printf("Linked list 2:");
    printlist (head 2);
    Sort list (head 1);
    pf("Sorted linked list 1:")
    printlist (head 1);
    reverselist (&head2);
    pf("Reversed linked list 2:");
    printlist (head 2);
    concatenate (&head1, head2);
    pf("Concatenated linked list:");
    print list (head1);
    return 0;
}
```

Output:
Linked list 1: 1 -> 3 -> 5 -> NULL
Linked list 2: 2 -> 4 -> 6 -> NULL
Sorted linked list 1: 1 -> 3 -> 5 -> NULL
Reversed linked list 2: 6 -> 4 -> 2 -> NULL
Concatenated linked list: 1 -> 3 -> 5 -> 6 -> 4 -> 2 -> NULL

Program 6b

Write A Program to Implement Single Link List to simulate Stack & Queue Operations.

Code:

```
#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));
   if (!newNode) {
      printf("Memory allocation error\n");
      return NULL;
   }
   newNode->data = data;
   newNode->next = NULL;
   return newNode;
}

void push(Node** top, int data) {
   Node* newNode = createNode(data);
   if (!newNode) return;
   newNode->next = *top;
   *top = newNode;
   printf("%d pushed to stack\n", data);
}

int pop(Node** top) {
   if (*top == NULL) {
      printf("Stack Underflow\n");
      return -1;
   }
   Node* temp = *top;
   int poppedData = temp->data;
   *top = temp->next;
   free(temp);
   printf("%d popped from stack\n", poppedData);
   return poppedData;
}

void displayStack(Node* top) {
   if (top == NULL) {
```

```
        printf("Stack is empty\n");
        return;
    }
    printf("Stack: ");
    Node* temp = top;
    while (temp) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

void enqueue(Node** front, Node** rear, int data) {
    Node* newNode = createNode(data);
    if (!newNode) return;
    if (*rear == NULL) {
        *front = *rear = newNode;
    } else {
        (*rear)->next = newNode;
        *rear = newNode;
    }
    printf("%d enqueued to queue\n", data);
}

int dequeue(Node** front, Node** rear) {
    if (*front == NULL) {
        printf("Queue Underflow\n");
        return -1;
    }
    Node* temp = *front;
    int dequeuedData = temp->data;
    *front = temp->next;
    if (*front == NULL) {
        *rear = NULL;
    }
    free(temp);
    printf("%d dequeued from queue\n", dequeuedData);
    return dequeuedData;
}

void displayQueue(Node* front) {
    if (front == NULL) {
        printf("Queue is empty\n");
        return;
    }
    printf("Queue: ");
    Node* temp = front;
```

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

int main() {
    Node* stackTop = NULL;

    printf("\n--- Stack Operations ---\n");
    push(&stackTop, 10);
    push(&stackTop, 20);
    push(&stackTop, 30);
    displayStack(stackTop);
    pop(&stackTop);
    displayStack(stackTop);

    Node* queueFront = NULL;
    Node* queueRear = NULL;

    printf("\n--- Queue Operations ---\n");
    enqueue(&queueFront, &queueRear, 1);
    enqueue(&queueFront, &queueRear, 2);
    enqueue(&queueFront, &queueRear, 3);
    displayQueue(queueFront);
    dequeue(&queueFront, &queueRear);
    displayQueue(queueFront);

    return 0;
}
```

```
--- Stack Operations ---
10 pushed to stack
20 pushed to stack
30 pushed to stack
Stack: 30 -> 20 -> 10 -> NULL
30 popped from stack
Stack: 20 -> 10 -> NULL

--- Queue Operations ---
1 enqueued to queue
2 enqueued to queue
3 enqueued to queue
Queue: 1 -> 2 -> 3 -> NULL
1 dequeued from queue
Queue: 2 -> 3 -> NULL
```

store
67 ____/____

WAP to implement simply linked list to simulate stack & queue operations.

```
#include <stdio.h>
#include <stdlib.h>
struct Node{
    int data;
    struct Node* next;
} Node;
struct Stack{
    Node* top;
} Stack;
struct queue{
    Node* front;
    Node* rear;
} Queue;
Node *createNode(int data){
    Node *newNode = (Node*) malloc(sizeof(Node));
    if (!newNode){
        printf("Memory error\n");
        return NULL;
    }
    newNode -> data = data;
    newNode -> next = NULL;
    return newNode;
}
void initStack(stack* stack){
    stack -> top = NULL;
}
int isEmptyStack(stack* stack){
    return stack -> top == NULL;
}
```

```
void push(stack *stack, int data){
    Node * newNode = create Node(data);
    if stack -> top){
        newNode -> next = stack -> top;
        stack -> top = newNode;
    }
    int pop(stack *stack){
        if(isEmptyStack(stack)) {
            printf("Empty stack");
            return -1;
        }
        int data = stack -> top -> data;
        Node *temp = stack -> top;
        stack -> top = stack -> top -> next;
        free(temp);
        return data;
    }
    void initQueue(Queue *queue){
        queue -> front = NULL;
        queue -> rear = NULL;
    }
    int isEmptyQueue(Queue *queue){
        return queue -> front == NULL;
    }
    void enqueue(Queue *queue, int data){
        Node* newNode = create Node(data);
        if (isEmptyQueue(queue)){
            queue -> front = queue -> rear = newNode
            queue -> rear = newNode;
        }
    }
```

Program 7

Write A Program to Implement doubly link list with primitive operations
a) Create a doubly linked list.
b) Insert a new node to the left of the node.
c) Delete the node based on a specific value
d) Display the contents of the list

Code:

```
#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));
   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 = newNode;
   } else {
      newNode->next = *head;
      (*head)->prev = newNode;
      *head = newNode;
   }
}

void insertAtPosition(struct Node** head, int data, int position) {
   if (position < 1) {
      printf("Invalid position!\n");
      return;
   }

   struct Node* newNode = createNode(data);
   if (position == 1) {
      insertAtBeginning(head, data);
      return;
```

```
    }

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

void insertAtEnd(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;
    newNode->prev = temp;
}

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

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

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

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

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

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

}

```
Doubly Linked List Operations:
1. Insert at Beginning
2. Insert at Position
3. Insert at End
4. Display List
5. Exit
Enter your choice: 1
Enter data to insert at beginning: 1

Doubly Linked List Operations:
1. Insert at Beginning
2. Insert at Position
3. Insert at End
4. Display List
5. Exit
Enter your choice: 2
Enter data to insert: 2
Enter position: 2

Doubly Linked List Operations:
1. Insert at Beginning
2. Insert at Position
3. Insert at End
4. Display List
5. Exit
Enter your choice: 3
Enter data to insert at end: 3
```

```
Doubly Linked List Operations:
1. Insert at Beginning
2. Insert at Position
3. Insert at End
4. Display List
5. Exit
Enter your choice: 4
List contents: 1 2 3

Doubly Linked List Operations:
1. Insert at Beginning
2. Insert at Position
3. Insert at End
4. Display List
5. Exit
Enter your choice: 5
Exiting program.
```

WAP to implement doubly linked list with primitive operations:
a) Create a doubly linked list
b) Insert a new node at a specific position.
d) Display contents of 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));
    newNode -> data = data;
    newNode -> prev = NULL;
    newNode -> next = NULL;
    return newNode;
}
Void createList(Struct Node** head) {
    int n, data;
    printf("Enter number of nodes:");
    scanf("%d", &n);
    for(int i=0; i<n; i++) {
        printf("Enter data for node %d", i+1);
        scanf("%d", &data);
        insertAtEnd(head, data);
    }
}
```

```
Void insertAtBeg (struct Node** head, int data) {
    Struct Node* newNode = create Node (data);
    if (head == NULL) {
        *head = newNode;
    }
    else {
        newNode -> next = *head;
        (*head) -> prev = newNode;
        *head = newNode;
    }
}

Void insertAtPos (struct Node** head, int data, int pos) {
    if (pos<1) {
        printf("Invalid position!\n");
        return;
    }
    Struct Node* newNode= createNode (data);
    if (pos == 1) {
        insertAtBeg (head, data);
        return;
    }
    Struct Node* temp=*head;
    for(int i=1; temp!=NULL && i<pos-1; i++) {
        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;
}
Void insertAtEnd (Struct Node** head, int data) {
    Struct Node* newNode = create Node (data);
    if (head == NULL) {
        *head = newNode;
        return;
    }
    Struct Node* temp = *head;
    while (temp -> next != NULL) {
        temp = temp -> next;
    }
    temp -> next = newNode;
    newNode -> prev = temp;
}
Void display (Struct Node *head) {
    if (head == NULL) {
        printf ("Empty list");
        return;
    }
    Struct Node* temp= head;
    printf("List contents: ");
    while (temp != NULL) {
        printf("%d", temp->data);
        temp= temp -> next;
    }
    printf("\n");
}
```

```
void main()
    struct Node* head=NULL;
    int choice, data, pos;
    while(1){
    printf("Doubly Linked List Operations:\n");
    printf("1. Create List\n");
    printf("2. Insert at Beginning\n");
    printf("3. Insert at position\n");
    printf("4. Insert at end\n");
    printf("5. Display list\n");
    printf("6. Exit\n");
    printf("Enter your choice:");
    scanf("%d", &choice);
    switch (choice){
        case 1: create List(&head);
                break;
        case 2: printf("Enter data to insert at beginning:");
                scanf("%d", &data);
                insertAtBeg(&head, data);
                break;
        case 3: printf("Enter data to insert:");
                scanf("%d", &data);
                printf("Enter position:");
                scanf("%d", &pos);
                insertAtPos(&head, data, position);
                break;
        case 4: printf("Enter data to insert at end:");
                scanf("%d", &data);
                insert At End(&head, data);
                break;
        case 5: display(head);
                break;
```

```
        case 6: printf("Exiting program\n");
                exit(0);
        default: printf("Invalid choice!");
    }
}
}

Output:
Doubly Linked List Operations:
1. Insert at Beginning
2. Insert at Position
3. Insert at End
4. Display List
5. Exit

Enter your choice: 1
Enter data to insert at Beginning: 1

Enter your choice: 2
Enter data to insert: 2
Enter position: 2

Enter your choice: 3
Enter data to insert at end: 3

Enter your choice: 4
List contents: 1 2 3
```

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.

Code:

```
#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 = NULL;
    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 if (data > root->data) {
        root->right = insert(root->right, data);
    }
    return root;
}

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

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

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

int main() {
    struct Node* root = NULL;
    int n, value;

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

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

    printf("\nIn-order Traversal: ");
```

```
    inOrder(root);

    printf("\nPre-order Traversal: ");
    preOrder(root);

    printf("\nPost-order Traversal: ");
    postOrder(root);

    return 0;
}
```

```
Enter the number of elements to insert in the BST: 5
Enter 5 elements:
12 23 45 65 3

In-order Traversal: 3 12 23 45 65
Pre-order Traversal: 12 3 23 45 65
Post-order Traversal: 3 65 45 23 12
```

WAP
a) To construct a BST
b) To traverse the tree using all methods i.e., in-order, pre-order, post order.
c) 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 = NULL;
    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 if (data > root -> data) {
        root -> right = insert(root -> right, data);
    }
    return root;

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

void preOrder(struct Node* root) {
    if (root != NULL) {
        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 n, value;

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

    printf("Enter %d element:\n", n);
    for(int i = 0; i<n; i++){
      scanf("%d", &value);
      root = insert(root, value);
    }
    printf("\n In-order Traversal:");
      inOrder(root);

    printf("\n Pre-Order Traversal:");
      preOrder(root);

    printf("\n PostOrder Traversal:");
      postOrder(root);

    return 0;
}
```

Output:

```
Enter the number of elements in the BST: 5
Enter 5 elements:
12 23 45 65 3

In-Order Traversal: 3 12 23 45 65
Pre-Order Traversal: 12 3 23 45 65
Post-Order Traversal: 3 65 45 23 12
```

Program 9a

Write a program to traverse a graph using BFS method.

Code:

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

#define MAX 100

struct Queue {
    int items[MAX];
    int front, 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 item = q->items[q->front];
    q->front++;
    if (q->front > q->rear) {
        q->front = q->rear = -1;
    }
    return item;
}

void BFS(int graph[MAX][MAX], int n, int startVertex) {
    int visited[MAX] = {0};
```

```
    struct Queue q;
    initQueue(&q);

    visited[startVertex] = 1;
    enqueue(&q, startVertex);

    printf("BFS Traversal: ");

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

        for (int i = 0; i < n; i++) {
            if (graph[currentVertex][i] == 1 && !visited[i]) {
                visited[i] = 1;
                enqueue(&q, i);
            }
        }
    }
    printf("\n");
}

int main() {
    int graph[MAX][MAX], n, startVertex;

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

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

    printf("Enter the starting vertex (0 to %d): ", n - 1);
    scanf("%d", &startVertex);

    BFS(graph, n, startVertex);

    return 0;
}
```

```
Enter the number of vertices in the graph: 4
Enter the adjacency matrix of the graph:
0 1 0 0
1 0 1 1
0 1 0 1
0 1 1 0
Enter the starting vertex (0 to 3): 0
BFS Traversal: 0 1 2 3
```

WAP to traverse a graph using BFS method

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

#define MAX 100

struct Queue {
    int items[MAX];
    int front, 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 item = q->items[q->front];
    q->front++;
    if(q->front > q->rear){
        q->front = q->rear = -1;
    }

    return item;
}

void BFS(int graph[MAX][MAX], int n, int startVertex){
    int visited[MAX] = {0};
    struct Queue q;
    initQueue(&q);

    visited[startVertex] = 1;
    enqueue(&q, startVertex);

    printf("BFS Traversal:");

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

        for(int i = 0; i < n; i++){
            if(graph[currentVertex][i] == 1 && !visited[i]){
                visited[i] = 1;
                enqueue(&q, i);
            }
        }
    }
    printf("\n");
}

int main(){
    int graph[MAX][MAX], n, startVertex;

    printf("Enter the no. of vertices in the graph:");
    scanf("%d", &n);

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

    printf("Enter the starting vertex (0 to %d):", n-1);
    scanf("%d", &startVertex);

    BFS(graph, n, startVertex);

    return 0;
}
```

Output

```
Enter the number of vertices in the graph: 4
Enter the adjacency matrix of the graph:
    0 1 0 0
    1 0 1 1
    0 1 0 1
    0 1 1 0
Enter the starting vertex (0 to 3): 0
BFS Traversal: 0 1 2 3
```

Program 9b

Write a program to traverse through a graph using DFS method.

Code:

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

#define MAX 100

int graph[MAX][MAX];
int visited[MAX];

void DFS(int vertex, int n) {
   printf("%d ", vertex);
   visited[vertex] = 1;

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

int main() {
   int n, startVertex;

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

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

   printf("Enter the starting vertex (0 to %d): ", n - 1);
   scanf("%d", &startVertex);

   for (int i = 0; i < n; i++) {
      visited[i] = 0;
   }

   printf("DFS Traversal: ");
   DFS(startVertex, n);
   printf("\n");
```

```
    return 0;
}
```

```
Enter the number of vertices in the graph: 5
Enter the adjacency matrix of the graph:
0 1 1 0 0
1 0 1 1 0
1 1 0 1 1
0 1 1 0 1
0 0 1 1 0
Enter the starting vertex (0 to 4): 0
DFS Traversal: 0 1 2 3 4
```

Write a program to traverse through a graph using DFS method.

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

#define MAX 100

int graph[MAX][MAX];
int visited[MAX];

void DFS(int vertex, int n){
    printf("%d", vertex);
    visited[vertex] = 1;

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

int main(){
    int n, startVertex;

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

```
printf("Enter the starting vertex (0 to %d):", n-1);
scanf("%d", &startVertex);

for(int i=0; i<n; i++){
    visited[i]=0;
  }

printf(" DFS Traversal:");
DFS(startVertex, n);
printf("\n");

return 0;
}
```

Output

```
Enter the number of vertices in the graph: 5
Enter the adjacent matrix of graph:
0 1 1 0 0
1 0 1 1 0
1 1 0 1 1
0 1 1 0 1
0 0 1 1 0
Enter the starting vertex (0 to 4): 0
DFS Traversal: 0 1 2 3 4
```