VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Data Structures using C Lab

(23CS3PCDST)

Submitted by

SUMANTH S SHETTY (1BM23CS348)

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

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

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

CERTIFICATE

This is to certify that the Lab work entitled "Data Structures using C Lab (23CS3PCDST)"carried out by **SUMANTH S SHETTY(1BM23CS348),** 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. 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.

Geetha N Assistant Professor Department of CSE, BMSCE	Dr.Kavitha Sooda Professor &HOD Department of CSE, BMSCE

INDEX

https://github.com/SumanthS117/1BM23CS348_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

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

#define MAX 5

typedef struct {
    int arr[MAX];
    int top;
} Stack;

void push(Stack* s, int value) {
    if (s->top == MAX - 1) {
        printf("Stack Overflow\n");
        return;
    }
    s->arr[++(s->top)] = value;
    printf("Pushed %d onto the stack\n", value);
}

int pop(Stack* s) {
    if (s->top == -1) {
        printf("Stack Underflow\n");
        return -1;
    }
    return s->arr[(s->top)--];
}

void display(Stack* s) {
    if (s->top == -1) {
```

```
        printf("Stack is empty\n");
        return;
    }
    printf("Stack elements: ");
    for (int i = 0; i <= s->top; i++) {
        printf("%d ", s->arr[i]);
    }
    printf("\n");
}

int main() {
    Stack s;
    s.top = -1;
    int choice, value;

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

        switch (choice) {
            case 1:
                printf("Enter value to push: ");
                scanf("%d", &value);
                push(&s, value);
                break;
            case 2:
                value = pop(&s);
                if (value != -1) {
                    printf("Popped value: %d\n", value);
                }
                break;
            case 3:
```

```
            display(&s);
            break;
        case 4:
            exit(0);
        default:
            printf("Invalid choice\n");
      }
   }
   return 0;
}
```

OUTPUT:

```
PS C:\Users\hp\Desktop\cbeg> ./1pc

Stack Operations:
1. Push 2. Pop 3. Display 4. Exit Enter your choice: 1
Enter value to push: 23
Pushed 23 onto the stack

Stack Operations:
1. Push 2. Pop 3. Display 4. Exit Enter your choice: 1
Enter value to push: 34
Pushed 34 onto the stack

Stack Operations:
1. Push 2. Pop 3. Display 4. Exit Enter your choice: 1
Enter value to push: 45
Pushed 45 onto the stack

Stack Operations:
1. Push 2. Pop 3. Display 4. Exit Enter your choice: 3
Stack elements: 23 34 45

Stack Operations:
1. Push 2. Pop 3. Display 4. Exit Enter your choice: 2
Popped value: 45

Stack Operations:
1. Push 2. Pop 3. Display 4. Exit Enter your choice: 2
Popped value: 34

Stack Operations:
1. Push 2. Pop 3. Display 4. Exit Enter your choice: 2
Popped value: 23

Stack Operations:
1. Push 2. Pop 3. Display 4. Exit Enter your choice: 2
Stack Underflow

Stack Operations:
1. Push 2. Pop 3. Display 4. Exit Enter your choice: 3
Stack is empty
```

Bafna Gold
Date: Page:

```
#include <stdio.h>

#define MAX 3

struct Stack {
    int top;
    int items[max];
};

void initstack( struct stack *s)
{
    s->top = -1;
}
int isfull(struct Stack *s) {
    return
    if ( s->top == max-1)
    { return 1;
    }
    else return 0;
}
int isempty (struct stack *s){
    if (s->top == -1)
        return 1;
    else
        return 0;
}
void push( struct stack *s, int data) {
    if (!isefull(s)) {
        top++;
        s->items[top] = data;
    } else printf ("It is full");
}
```

```
void pop(struct stack *s)
{   if (isempty(s)!=1)
      {  s->top
         S->items[top] int re_items= s-> items [top];
// S->items[top]= S-> items[top-1];
           top--;
      }
    else
     printf("Underflow");

}

void
void peek(struck stack *s)
 {
     if (isempty(s)!=1)
      {
        printf [s->items->[top]
        printf("%d", s->items[top]);
      }
     else
       printf("Empty stack");
 }

void traversal (struct Stack *s)
  {  printf int i=0;
       while(i != max)
                top
     { printf("%d", s->items[i]);
       i++;
     }

  }
```

Bafna Gold
Date: Page:

```
int
void main()
{
    struct Stack *v = (struct stack *) malloc(sizeof(struct Stack));
    initstack(v);
int x = isfull(v);
int y = isempty(v);
    push(v, 30);
    pop(v);
    peek(v);
    traversal(v);
}

if (x == 1)
    printf("Full");
else if(
```

2 → 3
1 → 1

operation
pop(v);
pop(v); o/p
pop(v); underflow → peek(v) → emptystack o/p

push(v, 30);
push(v, 40);
push(v, 50);
push(v, 60); → overflow

peek(v); → 60

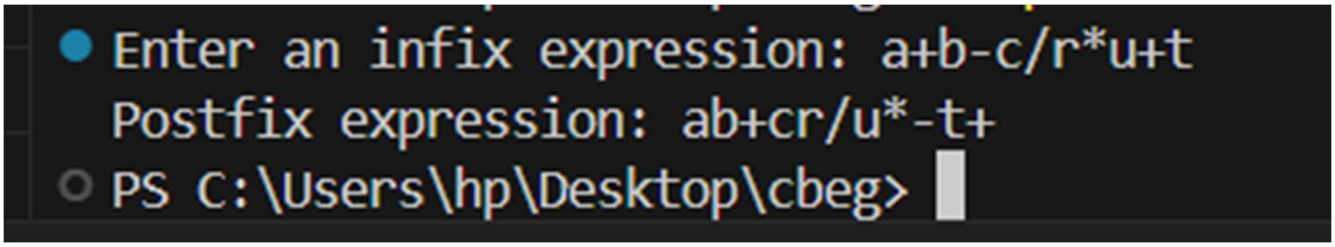
Program 2

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

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

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

void infixToPostfix(char* infix, char* postfix) {
    char stack[MAX];
    int top = -1, j = 0;
    for (int i = 0; infix[i] != '\0'; i++) {
        if (isalnum(infix[i])) {
            postfix[j++] = infix[i];
        } else if (infix[i] == '(') {
            stack[++top] = infix[i];
        } else if (infix[i] == ')') {
            while (top != -1 && stack[top] != '(') {
                postfix[j++] = stack[top--];
            }
            top--;
        } else {
            while (top != -1 && precedence(stack[top]) >= precedence(infix[i])) {
                postfix[j++] = stack[top--];
            }
            stack[++top] = infix[i];
        }
    }
    while (top != -1) {
```

```
        postfix[j++] = stack[top--];
    }
    postfix[j] = '\0';
}

int main() {
    char infix[MAX], postfix[MAX];
    strcpy(infix, "A*(B+C)");
    infixToPostfix(infix, postfix);
    printf("%s\n", postfix);
    return 0;
}
```

OUTPUT:



2) Convert infix expression to postfix

7/10/14

```
#include<stdio.h>
#include<stdlib.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 1;
  else
      return -1; }

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

void intopost(const char *s) {
  int len = strlen(s);
```


}

```
else {
  while (stackIndex >= 0 && (prec(c)
  < prec(stack[stackIndex]) || (prec(c) ==
  prec(stack[stackIndex]) && associativity(c) == 'L'))
  {
    result[resultIndex++] = stack[stackIndex--];
  }
  stack[++stackIndex] = c;
 }
}
while (stackIndex >= 0) {
  result[resultIndex++] = stack[stackIndex--];
}
result[resultIndex] = '\0';
printf("%s\n", result);

free(result);
free(stack); }

int main() {
  char exp[100]; scanf("%s", exp);
// char exp[] = "a+b*(c^d-e)^(f+g+h)";

  infixtopostfix(exp);
  return 0;
}
```

o/p

Postfix expression is

abcd^e-fgh*+^+c-

a b c d ^ e - f g h * + ^ + c -

Dynamic i/p

Bafna Gold
Date: Page:

~~int precedance~~
~~→ This is used~~
~~if C = ^~~

Pseudocode

1) Create an empty stack and an empty array called postfix, char stack[max] char postfix[max]
2) iterate through each character in the given expression)
   i) if the character is an operand insert it into postfix
      postfix[j++] = operand
   else push it to stack
   If prec(stack(Top) < prec(operator)
      push operator to stack
   else pop Top; and insert to postfix then push operator to stack
at the end postfix[' '] = '\0';
iterate through postfix and print

Program 3

3a) WAP to simulate the working of a queue of integers using an array. Provide the following operations: Insert, Delete, Display The program should print appropriate messages for queue empty and queue overflow conditions

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

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

void insert() {
   int value;
   if (rear == MAX - 1) {
      printf("Queue Overflow\n");
      return;
   }
   printf("Enter value to insert: ");
   scanf("%d", &value);
   if (front == -1)
      front = 0;
   queue[++rear] = value;
}

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

void display() {
   if (front == -1) {
      printf("Queue is empty\n");
      return;
```

```
    }
    printf("Queue: ");
    for (int i = front; i <= rear; i++)
        printf("%d ", queue[i]);
    printf("\n");
}

int main() {
    int choice;
    do {
        printf("\n1. Insert\n2. Delete\n3. Display\n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: insert(); break;
            case 2: delete(); break;
            case 3: display(); break;
            case 4: printf("Exiting...\n"); break;
            default: printf("Invalid choice\n");
        }
    } while (choice != 4);

    return 0; }
```

OUTPUT:

```
PS C:\Users\hp\Desktop\cbeg> ./3pc
1. Insert 2. Delete 3. Display 4. Exit   Enter your choice: 1
Enter value to insert: 23
1. Insert 2. Delete 3. Display 4. Exit   Enter your choice: 1
Enter value to insert: 25
1. Insert 2. Delete 3. Display 4. Exit   Enter your choice: 1
Enter value to insert: 3
1. Insert 2. Delete 3. Display 4. Exit   Enter your choice: 3
Queue: 23 25 3
1. Insert 2. Delete 3. Display 4. Exit   Enter your choice: 2
Deleted: 23
1. Insert 2. Delete 3. Display 4. Exit   Enter your choice: 2
Deleted: 25
1. Insert 2. Delete 3. Display 4. Exit   Enter your choice: 2
Deleted: 3
1. Insert 2. Delete 3. Display 4. Exit   Enter your choice: 2
Queue Underflow
1. Insert 2. Delete 3. Display 4. Exit   Enter your choice: 1
Enter value to insert: 23
1. Insert 2. Delete 3. Display 4. Exit   Enter your choice: 1
Enter value to insert: 34
1. Insert 2. Delete 3. Display 4. Exit   Enter your choice: 1
Enter value to insert: 45
1. Insert 2. Delete 3. Display 4. Exit   Enter your choice: 1
Enter value to insert: 456
1. Insert 2. Delete 3. Display 4. Exit   Enter your choice: 1
Enter value to insert: 78
1. Insert 2. Delete 3. Display 4. Exit   Enter your choice: 1
Queue Overflow
```

14/10/24

Implement linear queue and show
1) insert
2) delete
3) display

```
#include<stdio.h>
#include<stdlib.h>
struct Queue {
  int size;
  int f;
  int r;
  int *arr;
 }

int isfull(struct queue *q)
{ return if(q->r==q->size-1);
 }

int isempty(struct *q)
 {
   return (q->f==q->r);
 }

void enque(struct queue *q, int val)
 { if (isfull(q))
    { printf("Overflow\n");
      exit(0);
    }
   else
     q->arr[++q->r] = val;
  }

int deque(struct queue *q)
{ a=-1;
  if (isempty(q))
  { printf("underflow\n");
    exit(0);
  }
  else { q->f++;
  int a = q->arr[q->f];
  q->f++;
  return a;
 }
```

```
return q;
}
int main(){
 Struct queue q;
 q.size = 10;
 q.f = -1;
 q.r = -1;
 q.arr = (int *) malloc (q.size * sizeof

void disp(Struct queue *q)
{ if(!isempty(q)
 {for(i=q->front; i<=q->rear; i++)
  { printf("element %d is %d", i, q->arr[i]);
  }
 }
 else
 { printf ("underflow \n");
    exit(0);
 }
}

int main(){
 struct queue q; int ch;
  q.size = 10;
  q.f = -1;
  q.r = -1;
  q.arr = (int *) malloc (q.size * sizeof(int));
  printf("Enter
  printf("Enter your choice \n");
  scanf("%
  printf(" 1. insert at rear 2. Delete at front 3. Display \n")
  scanf("%d", &ch);
```

```
switch(ch)
{
    case 1: printf("Enter element");
            scanf("%d", &val);
            enqueue(&q, val);
            break;

    case 2: /*dequeue(&q); */ printf("popped %d", dequeue(&q));
            break;

    case 3: disp(&q);
            break;

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

}
```

Enter your choice
1. Insert
2. delete
3. display
4. quit
1
Enter element to be inserted 17
Enter your choice
1. insert
2. delete
3. displays
4. quit
1. ~~insert~~
Enter element to be inserted 23
Enter your choice
1. insert
2. delete
3. display
4. quit

Bafna Gold
Date: Page:

1.
Enter element to be inserted
33
overflow
Enter your choice
1. insert
2. delete
3. display
4. quit → 3 element 1 17 element 2 33
2.
popped 17
Enter your choice
1. insert
2. delete
3. display
4. quit
2.
Popped 33
Enter your choice
1. insert
2 delete
3. display
4. quit
2.
underflow
Enter your choice
33
invalid

3b) WAP to simulate the working of a circular queue of integers using an array. Provide the following operations: Insert, Delete & Display The program should print appropriate messages for queue empty and queue overflow conditions

```
#include <stdio.h>
#define MAX 5
int queue[MAX], front = -1, rear = -1;
void insert(int value) {
   if ((rear + 1) % MAX == front) printf("Queue is full\n");
   else {
      if (front == -1) front = 0;
      rear = (rear + 1) % MAX;
      queue[rear] = value;
   }
}
void delete() {
   if (front == -1) printf("Queue is empty\n");
   else {
      printf("Deleted: %d\n", queue[front]);
      if (front == rear) front = rear = -1;
      else front = (front + 1) % MAX;
   }
}
void display() {
   if (front == -1) printf("Queue is empty\n");
   else {
      int i = front;
      while (i != rear) {
         printf("%d ", queue[i]);
         i = (i + 1) % MAX;
      }
      printf("%d\n", queue[rear]);
   }
}
int main() {
   int choice, value;
   while (1) {
```

```
        printf("1. Insert\n2. Delete\n3. Display\n4. 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:
                return 0;
            default:
                printf("Invalid choice\n");
        }
    }
}
```

OUTPUT:

```
PS C:\Users\hp\Desktop\cbeg> ./4pc
1. Insert 2. Delete 3. Display 4. Exit Enter your choice: 1
Enter value to insert: 23
1. Insert 2. Delete 3. Display 4. Exit Enter your choice: 1
Enter value to insert: 25
1. Insert 2. Delete 3. Display 4. Exit Enter your choice: 1
Enter value to insert: 27
1. Insert 2. Delete 3. Display 4. Exit Enter your choice: 1
Enter value to insert: 28
1. Insert 2. Delete 3. Display 4. Exit Enter your choice: 1
Enter value to insert: 29
1. Insert 2. Delete 3. Display 4. Exit Enter your choice: 1
Enter value to insert: 31
Queue is full
1. Insert 2. Delete 3. Display 4. Exit Enter your choice: 3
23 25 27 28 29
1. Insert 2. Delete 3. Display 4. Exit Enter your choice: 2
Deleted: 23
1. Insert 2. Delete 3. Display 4. Exit Enter your choice: 2
Deleted: 25
1. Insert 2. Delete 3. Display 4. Exit Enter your choice: 2
Deleted: 27
1. Insert 2. Delete 3. Display 4. Exit Enter your choice: 2
Deleted: 28
1. Insert 2. Delete 3. Display 4. Exit Enter your choice: 2
Deleted: 29
1. Insert 2. Delete 3. Display 4. Exit Enter your choice: 2
Queue is empty
```

21/10/24 Implement circular queue using array
Show 1) insert
2) delete
3) display

```
#
#define size 3

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

void insert(int ele){
 if((front==0 && rear==size-1) || (rear==
    (front-1)%(size-1)))
 { printf("Queue Overflow");
   return;
 }
 else if(front==-1){
    front=rear=0;
    queue[rear]=ele;
 }
 else if(rear==size-1 && front!=0){
    rear=0;
    queue[rear]=ele;
 }
 else{ rear++;
    queue[rear]=ele;
 }
 else
 printf("%d has been inserted in", ele);
}

void delete(){
 if(front==-1){
  printf("Queue underflow\n");
  return;
 }
```

```
int temp = queue[front];
printf("%d has been deleted in", temp);
if(front == rear){
    front = rear = -1;
}
else if (front == size-1){
    front = 0;
}
else { front++; }
}
void display(){
if(front == -1){
  printf("Queue is empty\n");
  return;
}
printf("Elements of queue are:");
if (rear >= front){
  for(int i = front; i<=rear; i++){
    printf("%d", queue[i]);
  }
}
else { for(int i=front; i<size; i++){
    printf("%d", queue[i]);
  }
for(int i=0; i<=rear; i++){
  printf("%d", queue[i]);
}
}
```

```
printf("\n");
}
int main(){
int choice, ele;
while (true) {
printf("Enter choice");
printf("1.Insert \n 2. delete \n 3.display 4.Exit");
printf("Ente
scanf("%d", &choice);
switch(choice){
case 1: printf("Element");
        scanf("%d", &ele);
        insert(ele);
        break;
case 2 : delete();
        break;
case 3 : display()
        break;
case 4 : return 0;

default : Invalidprintf("Invalid\n");

}
}
}
```

o/p seen

21/10

Program 4

4) WAP to Implement Singly Linked List with following operations a) Createalinkedlist. b) Insertion of a node at first position, at any position and at end of list. Display the contents of the linked list.

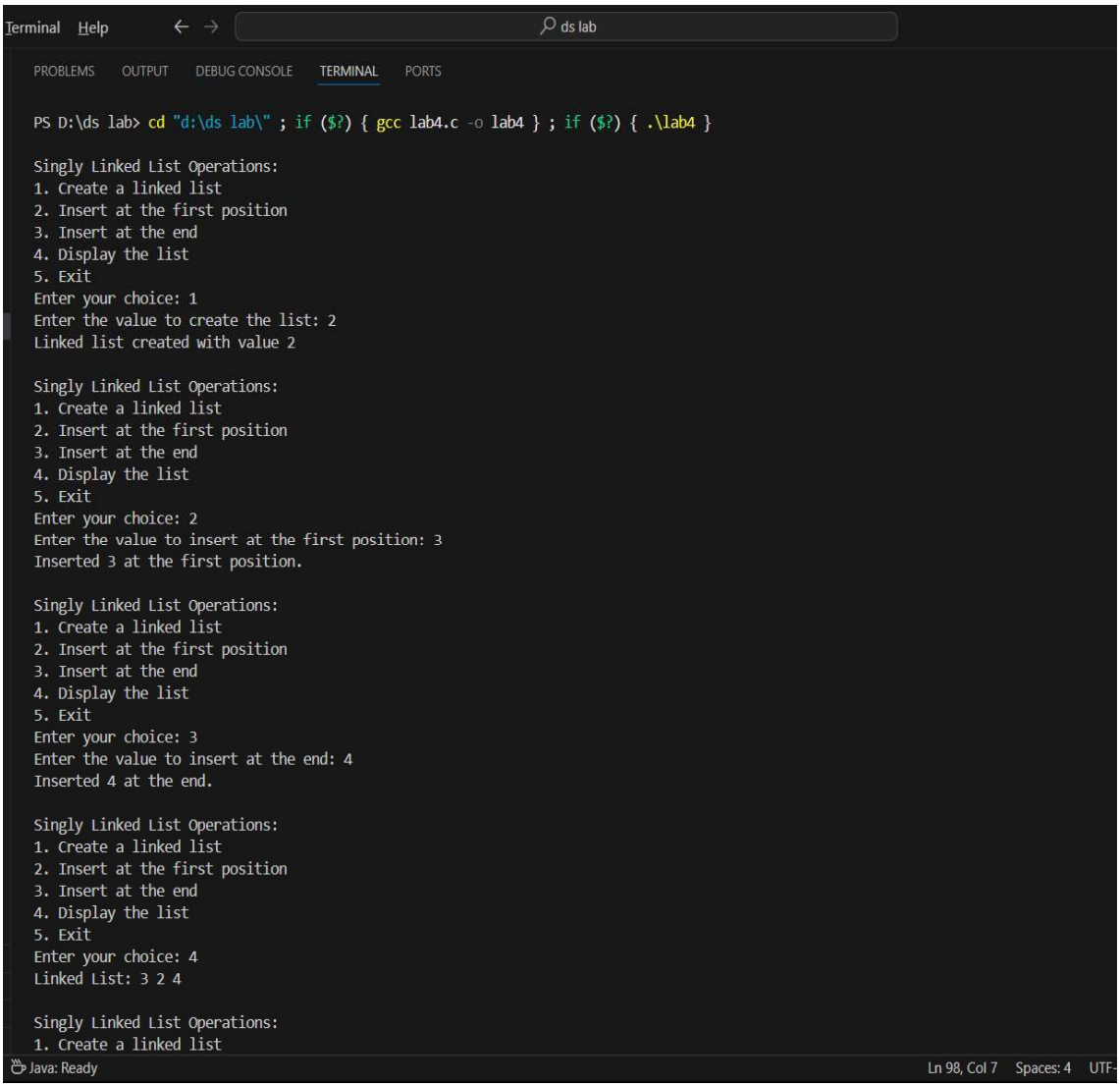
```
#include <stdio.h>
#include <stdlib.h>

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

struct Node* head = NULL;

void createLinkedList(int n) {
    int data, i;
    struct Node *newNode, *temp;

    if (n <= 0) return;

    head = (struct Node*)malloc(sizeof(struct Node));
    printf("Enter data for node 1: ");
    scanf("%d", &data);
    head->data = data;
    head->next = NULL;
    temp = head;

    for (i = 2; i <= n; i++) {
        newNode = (struct Node*)malloc(sizeof(struct Node));
        printf("Enter data for node %d: ", i);
        scanf("%d", &data);
        newNode->data = data;
        newNode->next = NULL;
        temp->next = newNode;
        temp = temp->next;
    }
```

```
}

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

void insertAtEnd(int data) {
   struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));
   struct Node *temp = head;

   newNode->data = data;
   newNode->next = NULL;

   if (head == NULL) {
      head = newNode;
      return;
   }

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

void insertAtPosition(int data, int position) {
   struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));
   struct Node *temp = head;
   int i;

   newNode->data = data;

   if (position == 1) {
      newNode->next = head;
      head = newNode;
```

```
        return;
    }

    for (i = 1; i < position - 1 && temp != NULL; i++) {
        temp = temp->next;
    }

    if (temp != NULL) {
        newNode->next = temp->next;
        temp->next = newNode;
    } else {
        printf("Position out of range\n");
    }
}

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

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

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

int main() {
    int n, choice, data, position;

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

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

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

    return 0;}
```

OUTPUT:


Terminal Help
ds lab
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS D:\ds lab> cd "d:\ds lab\" ; if ($?) { gcc lab4.c -o lab4 } ; if ($?) { .\lab4 }
Singly Linked List Operations:
1. Create a linked list
2. Insert at the first position
3. Insert at the end
4. Display the list
5. Exit
Enter your choice: 1
Enter the value to create the list: 2
Linked list created with value 2
Singly Linked List Operations:
1. Create a linked list
2. Insert at the first position
3. Insert at the end
4. Display the list
5. Exit
Enter your choice: 2
Enter the value to insert at the first position: 3
Inserted 3 at the first position.
Singly Linked List Operations:
1. Create a linked list
2. Insert at the first position
3. Insert at the end
4. Display the list
5. Exit
Enter your choice: 3
Enter the value to insert at the end: 4
Inserted 4 at the end.
Singly Linked List Operations:
1. Create a linked list
2. Insert at the first position
3. Insert at the end
4. Display the list
5. Exit
Enter your choice: 4
Linked List: 3 2 4
Singly Linked List Operations:
1. Create a linked list
Java: Ready
Ln 98, Col 7 Spaces: 4 UTF-

28/10/24

WAP to implement simple linked list
1) insert at beginning
2) insert at ~~beg~~ end
3) display
4) ~~e~~

```
#include<stdio.h>
#include<stdlib.h>
Struct Node {
   int data;
   Struct Node * next;
   };
~~Struct Node~~
Struct Node * createNode(int data){
Struct Node * newnode = (Struct Node *) malloc
(sizeof (Struct Node));
 newnode->data=data;
 newnode->next = NULL;
 return newnode;
}

void insertend (Struct Node ** head, int data)
{
  Struct Node * newnode = createNode(data);
  if (*head == NULL)
    *head = newNode;
  }
  else {
    struct Node * temp = *head;
     while (temp-> next ) = NULL
       temp = temp->next;
     }
    temp -> next = newNode;
  }
```

```
void insertatBeg(Struct Node **head, int data)
{
  struct Node * temp=hed new node = CreateNode(data
    newNode->next = *head;
    *head = newNode;
  }

voiddisplay(Struct Node *head){
  struct Node *temp = head;
  while(temp != NULL){
  Printf("%d", temp->data);
  temp=temp->next
  }
  }

int main(){
  struct Node *head = NULL;
  int ch, data;
  while(true)
{ printf("Enterchoice 1. insert at beg 2. insert at
beg 3. display 4. Exit \n");
  scanf("%d", &ch);
  {  switch(ch)
      { case 1: printf("Enter data");
                scanf("%d", &data);
                insertatbeg(&head, data);
                break;

        case 2: Printf("Enter data");
                scanf("%d", &data);
                insertend(&head, data);
                break;
```

```
case 3 : display(head);
         break;
case 4 : exit(0);
default : printf(" Invalid ");
  }
}
return 0;
}
```

Program

28/10

Op screen

enter 1. insert begining 2. insert end 3. display 4. exit
enter data 223

enter 1. insert 2. insert end 3. display 4. exit 1
enter data 45

enter 1. insert begining 2. insert end 3. display 4. exit
enter data 443

enter 1. insert beginning 2. insert end 3. display 4. exit
enter data 22

enter 1. insert beginning 2. insert end 3. display
4. exit 3
-22-45--223--443-
enter 1. insert beginning 2. insert end 3. display 4. exit 4

Program 5

5)WAP to Implement Singly Linked List with following operations a) Create a linked list. b) Deletion of first element, specified element and last element in the list. c) Display the contents of the linked list.

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

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

struct Node* head = NULL;

void createLinkedList(int n) {
    int data, i;
    struct Node *newNode, *temp;

    if (n <= 0) return;

    head = (struct Node*)malloc(sizeof(struct Node));
    printf("Enter data for node 1: ");
    scanf("%d", &data);
    head->data = data;
    head->next = NULL;
    temp = head;

    for (i = 2; i <= n; i++) {
        newNode = (struct Node*)malloc(sizeof(struct Node));
        printf("Enter data for node %d: ", i);
        scanf("%d", &data);
        newNode->data = data;
        newNode->next = NULL;
        temp->next = newNode;
        temp = temp->next;
    }
```

```
}

void deleteFirst() {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }

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

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

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

    free(temp->next);
    temp->next = NULL;
}

void deleteSpecified(int key) {
    if (head == NULL) {
        printf("List is empty.\n");
        return; }
```

```
    struct Node *temp = head, *prev;

    if (head->data == key) {
        head = head->next;
        free(temp);
        return;
    }

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

    if (temp == NULL) {
        printf("Element not found.\n");
        return;
    }

    prev->next = temp->next;
    free(temp);
}

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

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

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

```
int main() {
    int n, choice, data;

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

    while (1) {
        printf("\nMenu: 1. Delete first element 2. Delete last element 3. Delete specified
element  4. Display list  5. Exit");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                deleteFirst();
                break;
            case 2:
                deleteLast();
                break;
            case 3:
                printf("Enter element to delete: ");
                scanf("%d", &data);
                deleteSpecified(data);
                break;
            case 4:
                displayList();
                break;
            case 5:
                exit(0);
            default:
                printf("Invalid choice.\n");
        }
    }

    return 0;}
```

OUTPUT:

```
PS C:\Users\hp\Desktop\cbeg> ./lpc
Enter number of nodes: 4
Enter data for node 1: 1
Enter data for node 2: 2
Enter data for node 3: 3
Enter data for node 4: 4

Menu: 1. Delete first element 2. Delete last element 3. Delete specified element  4. Display list  5. ExitEnter your choice: 1

Menu: 1. Delete first element 2. Delete last element 3. Delete specified element  4. Display list  5. ExitEnter your choice: 4
2 -> 3 -> 4 -> NULL

Menu: 1. Delete first element 2. Delete last element 3. Delete specified element  4. Display list  5. ExitEnter your choice: 2

Menu: 1. Delete first element 2. Delete last element 3. Delete specified element  4. Display list  5. ExitEnter your choice: 4
2 -> 3 -> NULL

Menu: 1. Delete first element 2. Delete last element 3. Delete specified element  4. Display list  5. ExitEnter your choice: 3
Enter element to delete: 2

Menu: 1. Delete first element 2. Delete last element 3. Delete specified element  4. Display list  5. ExitEnter your choice: 4
3 -> NULL
```

11/19 24

Bafna Gold
Date: Page:

WAP to implement simple linked list
1) Create a Linked list
2) delete first, last, specified element
3) display

10 15 12 16
13

```
#include <stdio.h>
#include <stdlib.h>
struct node{
  int data;
  struct node *next;
};
struct node *create(int data){
struct node *newnode=(struct node*) malloc
(sizeof(struct node));
   newnode->data=data;
   newnode->next=NULL;
   return newnode;
}
void insert(struct node *head, int data)
{  struct node *newnode=create(data);
   if(*head==NULL){
        *head=newnode;
       }
   else{
   struct node *temp=*head;
   while(temp->next!=NULL){
     temp=temp->next;
   }
   temp->next=newnode;
   }
}
```

```
void delf(struct node *head)
{ struct node *p= ~~struct node*~~) head;
    p = p->next;
    head = p;
}

void dellast(struct node *head)
{ struct node *p = head;
  struct node *q = head->next;

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

    p->next = NULL;
}

void delspec(struct node *head, int ~~data~~ info)
{ struct node *p = head;
  struct node *q = head->next;

    while(q->data != info)
    {   q = q->next;
        if(p->data == info)
        {   p = p->next;
            head = p;
            return;
        }
    while(q->data != info)
    {   q = q->next;
        p = p->next;
    }
    p->next = q->next;
}
```

```
void show( struct node *head)
{ struct node *p = head;

  while (p != NULL)
  { printf("element is %d\n", p->data);
    p = p->next;
  }

void / int main()
  // struct node *s = (struct node*)malloc
  //(sizeof(struct node));
  struct node head = NULL;

  while(1)
  { printf("Enter choice \n 1. insert 2. delete
    at first 3. delete at end 4. delete at specific
5. Show 6. quit");
    scanf("%d", &ch);
    switch(ch)
    { case 1: printf("Enter data");
              scanf("%d", &d);
              insert(head, d);
              break;
      case 2: delf(head);
              break;
      case 3: dellast(head);
              break;
      case 4: printf("Enter element to be
                      deleted");
              scanf("%d", &d);
              delspec(head, d);
              break;
      Case 5: show(head); break.
      case 6: exit(0);
```

```
void delind (struct node *head, int pos)
{ struct node *p = head;
                *q = q->next
  struct node  *q = head;
  while (i < pos)
  {     p = q;
        q = q->next;
  }
   p->next = q->next;
}
```

Procedure

o Enter choice 1. insert 2. del at first 3. del at last
4. deleted at pos 5-Show 6. Exit
1.
Enter element
2
Enter choice 1. insert 2. del at first 3. del at last 4.
del at pos 5-Show 6 Exit
1
Enter element
3
Enter choice 1. insert 2. del at first 3. del at
4 del at pos 5. Show 6 Exit
enter element 4
Enter choice 1. insert 2. del at first 3. del at end
4 del at pos 5 Show 6. Exit
2.
Enter choice 1. insert 2. del at first 3. del at
4 del at pos 5-Show 6. Exit
3

Program 6

6a) WAP to Implement Single Link List with following operations: Sort the linked list, Reverse the linked list, Concatenation of two linked lists

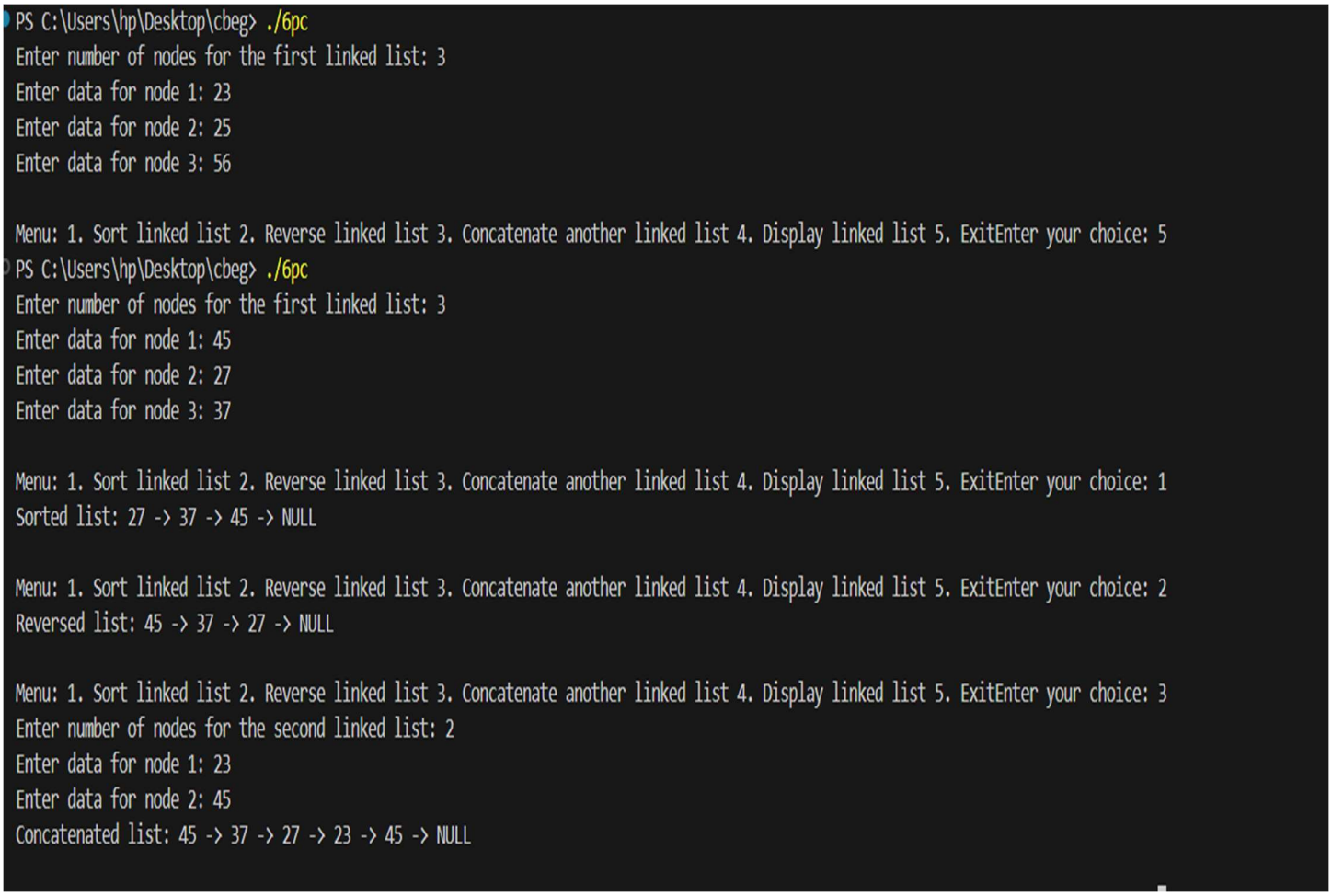
```
#include <stdio.h>
#include <stdlib.h>

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

struct Node* head = NULL;
struct Node* head2 = NULL;

void createLinkedList(struct Node** head, int n) {
    int data, i;
    struct Node *newNode, *temp;

    if (n <= 0) return;

    *head = (struct Node*)malloc(sizeof(struct Node));
    printf("Enter data for node 1: ");
    scanf("%d", &data);
    (*head)->data = data;
    (*head)->next = NULL;
    temp = *head;

    for (i = 2; i <= n; i++) {
        newNode = (struct Node*)malloc(sizeof(struct Node));
        printf("Enter data for node %d: ", i);
        scanf("%d", &data);
        newNode->data = data;
        newNode->next = NULL;
        temp->next = newNode;
        temp = temp->next;
    }
```

```
}

void sortLinkedList(struct Node* head) {
    struct Node *i, *j;
    int tempData;

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

    for (i = head; i != NULL; i = i->next) {
        for (j = i->next; j != NULL; j = j->next) {
            if (i->data > j->data) {
                tempData = i->data;
                i->data = j->data;
                j->data = tempData;
            }
        }
    }
}

void reverseLinkedList(struct Node** head) {
    struct Node *prev = NULL, *current = *head, *next = NULL;

    while (current != NULL) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    *head = prev;
}

void concatenateLinkedLists(struct Node** head1, struct Node* head2) {
    struct Node *temp = *head1;
```

```
    if (*head1 == NULL) {
        *head1 = head2;
        return;
    }

    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = head2;
}

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

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

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

int main() {
    int n, choice, m;

    printf("Enter number of nodes for the first linked list: ");
    scanf("%d", &n);
    createLinkedList(&head, n);

    while (1) {
        printf("\nMenu: 1. Sort linked list 2. Reverse linked list 3. Concatenate another
linked list 4. Display linked list 5. Exit");
        printf("Enter your choice: ");
```

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

        switch (choice) {
            case 1:
                sortLinkedList(head);
                printf("Sorted list: ");
                displayList(head);
                break;
            case 2:
                reverseLinkedList(&head);
                printf("Reversed list: ");
                displayList(head);
                break;
            case 3:
                printf("Enter number of nodes for the second linked list: ");
                scanf("%d", &m);
                createLinkedList(&head2, m);
                concatenateLinkedLists(&head, head2);
                printf("Concatenated list: ");
                displayList(head);
                break;
            case 4:
                displayList(head);
                break;
            case 5:
                exit(0);
            default:
                printf("Invalid choice.\n");
        }
    }

    return 0;
}
```

OUTPUT:


PS C:\Users\hp\Desktop\cbeg> ./6pc
Enter number of nodes for the first linked list: 3
Enter data for node 1: 23
Enter data for node 2: 25
Enter data for node 3: 56
Menu: 1. Sort linked list 2. Reverse linked list 3. Concatenate another linked list 4. Display linked list 5. ExitEnter your choice: 5
PS C:\Users\hp\Desktop\cbeg> ./6pc
Enter number of nodes for the first linked list: 3
Enter data for node 1: 45
Enter data for node 2: 27
Enter data for node 3: 37
Menu: 1. Sort linked list 2. Reverse linked list 3. Concatenate another linked list 4. Display linked list 5. ExitEnter your choice: 1
Sorted list: 27 -> 37 -> 45 -> NULL
Menu: 1. Sort linked list 2. Reverse linked list 3. Concatenate another linked list 4. Display linked list 5. ExitEnter your choice: 2
Reversed list: 45 -> 37 -> 27 -> NULL
Menu: 1. Sort linked list 2. Reverse linked list 3. Concatenate another linked list 4. Display linked list 5. ExitEnter your choice: 3
Enter number of nodes for the second linked list: 2
Enter data for node 1: 23
Enter data for node 2: 45
Concatenated list: 45 -> 37 -> 27 -> 23 -> 45 -> NULL

Bafna Gold
Date: Page:

WAP to implement singly linked list with following operation
1) sort the linked list
2) reverse
3) concatenate

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

Struct node{
    int ;
typedef struct node2
int data;
Struct node * next;
} node;

node * create(){};
void print( node *head);
node * sort( node *head );
node * reversed(node *head);
node * conc(node * head1, node * head2);

int main(){
node *head1 = NULL; *head2 = NULL;
int ch, lc;
printf("Enter element for 1st LinKed list");
head1 = create();
printf("Enter for 2nd LinKed list");
head2 = create();
```

```
printf("\n choose \n 1. sort 2. reverse 3. concatenate
  4. exit ");
scanf("%d", &ch);

switch (ch) {
case 1: printf ("choose 1. linkedlist1 2. linked list2)
        scanf("%d", &lc);
        if (lc == 1) {
         head1 = sort1(head1);
         printf("After sorting (1");
         print1(head1);
        }
        else {head2 = sort1(head2);
         printf("After sorting");
         println(head2);
        }
        break;
case 2: printf ("choose 1. linkedlist1 2. Linked list2)
        scanf("%d", &lc);
        if (lc == 1){
         head1 = reverse1(head1);
         printf("After reversing");
         println(head1);
        }
        else {
         head2 = reverse1(head2);
         printf("After reversing");
         print1(head2);
        }
        break;
```

```
case 3: head1 = con (head1, head2);
        printf (" Concatenated list");
        print(" head1);
        break;

default: print (" Invalid choice");
}
  return 0;
}

node *create() {
node *head = NULL; node *temp = NULL, node *newnode = NULL;
 int data;
printf(" Enter elements enter (-1) to end ");
while(1) {
 scanf(" %d", &data);
  if (data == -1) break;
 newnode = (node *) malloc (size of (node));
  newnode → data = data;
  newnode → next = NULL;
  if(head == NULL)
 { head = newnode;
  }
  else { temp → next = newnode; }
     temp = newnode;
   }
  return head;
}
```

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

node *sortl(node *head){
  if(head == NULL || head->next == NULL)
    return 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;
      }
    }
  }
  return head;
}

node *reversed(node *head){
  node *prev = NULL; node *curr = head; node *next = NULL;
  while(curr != NULL){
    next = curr->next;
    curr->next = prev;
    prev = curr;
    curr = next;
  }
  return prev;
}
```

```
node * concatenate(node *head1, node *head2)
{ if (head1==NULL) return head2;
  if (head2==NULL) return head1;

  node *temp = head1;
  while(temp->next!=NULL){
     temp = temp->next;
  }
     temp->next = head2;
    return head1;
  }
```

O/P

Enter elements for linked list1:
Enter(-1 to end): 2 4 3 7 6 -1
Enter elements for linked list2:
Enter(-1 to end): 9 7 6 -3 -9 -1

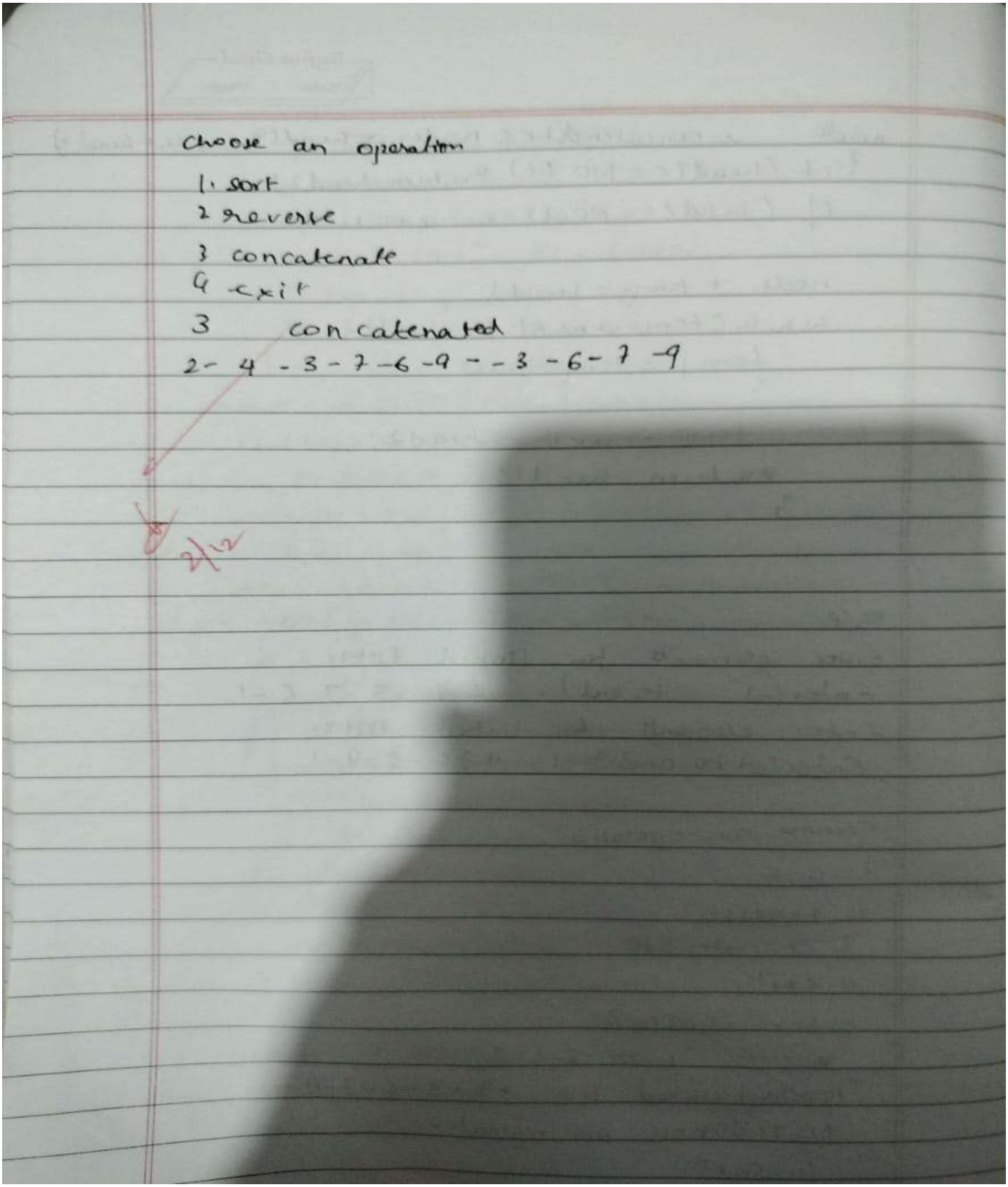
Choose an operation
1. Sort
2. reverse
3. Concatenate
4. exit
enter choice 2
reverse 1 or 2 2
reversed linked list -9 -3 -6 -7 -9 -
enter Choose an operation
1. sort
2. reverse
3. concatenate
4. exit
1.
sort 1 or 2 1
Sorted 2 -3 -4 -6 -7

```
node * concatenate( node *head1, node *head2)
{ if (head1 == NULL) return head2;
  if (head2 == NULL) return head1;

  node *temp = head1;
  while (temp->next != NULL) {
      temp = temp->next;
  }
      temp->next = head2;
    return head1;
  }
```

O/P

Enter elements for linked list1:
Enter(-1 to end): 2 4 3 7 6 -1
Enter elements for linked list2:
Enter(-1 to end): 9 7 6 -3 -9 -1

Choose an operation
1. Sort
2. Reverse
3. Concatenate
4. exit
enter choice 2
reverse 1 or 2 2
reversed linked list -9 -3 -6 -7 -9 -
enter Choose an operation
1. Sort
2. Reverse
3. Concatenate
4. Exit
1.
Sort 1 or 2 1
Sorted 2 -3 -9 -6 -7


choose an operation
1. sort
2 reverse
3 concatenate
4 exit
3 concatenated
2- 4 - 3 - 7 - 6 - 9 - - 3 - 6 - 7 - 9

6b)WAP to Implement Single Link List to simulate Stack & Queue Operations.

```
#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 push(struct Node** top, int data) {
    struct Node* newNode = createNode(data);
    newNode->next = *top;
    *top = newNode;
}

int pop(struct Node** top) {
    if (*top == NULL) {
        printf("Stack Underflow\n");
        return -1;
    }
    struct Node* temp = *top;
    int popped = temp->data;
    *top = (*top)->next;
    free(temp);
    return popped;
}
```

```
void enqueue(struct Node** front, struct Node** rear, int data) {
    struct Node* newNode = createNode(data);
    if (*rear == NULL) {
        *front = *rear = newNode;
        return;
    }
    (*rear)->next = newNode;
    *rear = newNode;
}

int dequeue(struct Node** front) {
    if (*front == NULL) {
        printf("Queue Underflow\n");
        return -1;
    }
    struct Node* temp = *front;
    int dequeued = temp->data;
    *front = (*front)->next;
    free(temp);
    return dequeued;
}

int main() {
    struct Node* stack = NULL;
    struct Node* queueFront = NULL;
    struct Node* queueRear = NULL;
    int choice, data;

    while (1) {
        printf("\n1. Push to Stack 2. Pop from Stack 3. Enqueue to Queue 4. Dequeue
from Queue 5. Exit ");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter data to push to stack: ");
```

```
            scanf("%d", &data);
            push(&stack, data);
            break;
        case 2:
            data = pop(&stack);
            if (data != -1) {
                printf("Popped from stack: %d\n", data);
            }
            break;
        case 3:
            printf("Enter data to enqueue to queue: ");
            scanf("%d", &data);
            enqueue(&queueFront, &queueRear, data);
            break;
        case 4:
            data = dequeue(&queueFront);
            if (data != -1) {
                printf("Dequeued from queue: %d\n", data);
            }
            break;
        case 5:
            exit(0);
        default:
            printf("Invalid choice\n");
    }
  }
return 0;}
```

OUTPUT:

```
PS C:\Users\hp\Desktop\cbeg> ./6bc

1. Push to Stack 2. Pop from Stack 3. Enqueue to Queue 4. Dequeue from Queue 5. Exit Enter your choice: 1
Enter data to push to stack: 23

1. Push to Stack 2. Pop from Stack 3. Enqueue to Queue 4. Dequeue from Queue 5. Exit Enter your choice: 1
Enter data to push to stack: 45

1. Push to Stack 2. Pop from Stack 3. Enqueue to Queue 4. Dequeue from Queue 5. Exit Enter your choice: 3
Enter data to enqueue to queue: 45

1. Push to Stack 2. Pop from Stack 3. Enqueue to Queue 4. Dequeue from Queue 5. Exit Enter your choice: 3
Enter data to enqueue to queue: 67

1. Push to Stack 2. Pop from Stack 3. Enqueue to Queue 4. Dequeue from Queue 5. Exit Enter your choice: 2
Popped from stack: 45

1. Push to Stack 2. Pop from Stack 3. Enqueue to Queue 4. Dequeue from Queue 5. Exit Enter your choice: 4
Dequeued from queue: 45
```

Bafna Gold
Date: Page:

;mplement Stacks and Queues using single linked list perform

1) push
2) pop
3) display
4) enqueue
5) dequeue
5) display

```
struct node {
    int data;
    struct node *next;
    };
Struct node * stop = NULL;
Struct node * queue_front = NULL;
void push( struct node **top, int data)
  { struct node *newnode = (struct node *)
                          malloc (sizeof(struct node));

      newnode -> data = data;
      newnode -> next = stop;
      stop = newnode;

    }

void pop (struct node **top);
  {     top = top -> next
        int *t = stop;
  }          t -> next = NULL;
            *top = (*top) -> next;
printf(" popped %d", t -> data);
    free(t);
  }
```

```
void ~~enque~~ (;
void stackshow(~~struct node~~ *top)
 { if (top == NULL)
    { printf ("empty");
      return;
    }
  struct node *t = stop;
  while (t!=NULL)
  { printf ("%d", t->data);
    t=t->next;
   }
  }
void enque(~~struct~~ int value) {
 struct node * nn = (struct node *)malloc (sizeof(struct
 Node));
 if (newnode == NULL) {
    printf ("Queue overflow\n");
     return;
    }
   nn->data = value;
   nn->next = NULL;
   if (queue_rear == NULL){
    queue_front = queue_rear = nn;
  }
  printf ("%d enqued into queue\n",
    value);
}
```

```
void deque ()
if (queue_front == NULL)
printf ("Queue underflow \n");
    return;
  }
struct node *t = queue_front;
printf ("%d dequed", t->data);
queue_front = queue_front->next;

if (queue_front == NULL)
  queue_rear = NULL;
  }
  free (temp);
}

void dq ()
{ if (queue_front == NULL)
  {
    printf ("Queue empty \n");
    return;
  }
struct node * temp = queue_front;
printf ("Queue is");
while (temp != NULL)
{ print ("%d", temp->data)
  temp = temp->next;
  }
}
```

Bafna Gold
Date: Page:

o/p

choose op 1. stack push 2. stack pop 3. stack display 4. enque 5. deque 6. disp Queue 7 exit
1
Enter value 6
choose op 1. stack push 2. stack pop 3. stack display 4. enque 5. deque 6. disp Queue 7 exit
1
Enter value 7
choose op 1. stack push 2. stack pop 3. stack display 4. enque 5. deque 6. disp Queue 7 exit
2
Popped 7
choose op 1. stack push 2. stack pop 3. stack display 4. enque 5. deque 6. disp Queue 7 exit
3
stack elements: 6
choose op 1. stack push 2. stack pop 3. stack display 4. enque 5. deque 6. disp Queue 7 exit
4
Enter value 3
choose op 1. stack push 2. stack pop 3. stack display 4. enque 5. deque 6. disp Queue 7 exit
4
Enter value 4
choose op 1. stack push 2. stack pop 3. stack disp 4. enque 5. deque 6. disp Que 7 exit
5
dequed 3
choose op 1. stack push 2. stack pop 3. stack disp 4. enque 5. deque 6. disp Queue 7 exit
6
Queue: 4

16/12

```
int main(){
int ch, v;
while(1){
printf("choose op 1. stack push 2. Stack pop
3. stack display 4. Queue enque 5. 6. Queue
display 7. exit\n");
printf("Enter choice");
scanf("%d", ch);
switch(ch){
case 1: printf("Enter value to push");
        scanf("%d", &value);
        push(v); &v);
        break;
case 2; pop();
        break;

Case 3: stackshow();
        break;

case 5: deque printf("Enter value");
        scanf("%d", &v);
        enqueue(v);
        break;
case 6: dequeue();
        break;
case 7: exit(0);

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

Program 7

7) WAP to Implement doubly link list with primitive operations a) Create a doubly linked list. b) Insert a new node to the left of the node. c) Delete the node based on a specific value d) 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));
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}

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

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

    if (temp == NULL) return;

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

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

    temp->prev = newNode;
}

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

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

    if (temp == NULL) return;

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

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

    free(temp);
}

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

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

```
    }

    printf("\n");
}

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

    while (1) {
        printf("\n1. Create Node 2. Insert Node to the Left 3. Delete Node 4. Display
List 5. Exit");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter data to create node: ");
                scanf("%d", &data);
                if (head == NULL) {
                    head = createNode(data);
                } else {
                    struct Node* temp = head;
                    while (temp->next != NULL) {
                        temp = temp->next;
                    }
                    temp->next = createNode(data);
                    temp->next->prev = temp;
                }
                break;
            case 2:
                printf("Enter target node data to insert left of: ");
                scanf("%d", &target);
                printf("Enter data to insert: ");
                scanf("%d", &data);
                insertLeft(&head, target, data);
                break;
```

```
            case 3:
                printf("Enter data of node to delete: ");
                scanf("%d", &data);
                deleteNode(&head, data);
                break;
            case 4:
                displayList(head);
                break;
            case 5:
                exit(0);
            default:
                printf("Invalid choice\n");
        }
    }

    return 0;
}
```

OUTPUT:


PS C:\Users\hp\Desktop\cbeg> ./7c
1. Create Node 2. Insert Node to the Left 3. Delete Node 4. Display List 5. ExitEnter your choice: 1
Enter data to create node: 345
1. Create Node 2. Insert Node to the Left 3. Delete Node 4. Display List 5. ExitEnter your choice: 5
PS C:\Users\hp\Desktop\cbeg> gcc 7c.c -o 7c
PS C:\Users\hp\Desktop\cbeg> ./7c
1. Create Node 2. Insert Node to the Left 3. Delete Node 4. Display List 5. ExitEnter your choice: 1
Enter data to create node: 23
1. Create Node 2. Insert Node to the Left 3. Delete Node 4. Display List 5. ExitEnter your choice: 1
Enter data to create node: 45
1. Create Node 2. Insert Node to the Left 3. Delete Node 4. Display List 5. ExitEnter your choice: 1
Enter data to create node: 67
1. Create Node 2. Insert Node to the Left 3. Delete Node 4. Display List 5. ExitEnter your choice: 2
Enter target node data to insert left of: 2
Enter data to insert: 56
1. Create Node 2. Insert Node to the Left 3. Delete Node 4. Display List 5. ExitEnter your choice: 3
Enter data of node to delete: 67
1. Create Node 2. Insert Node to the Left 3. Delete Node 4. Display List 5. ExitEnter your choice: 4
23 45

Bafna Gold
Date: Page:

Doubly linked list

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

struct node *create(int data) {
    struct node *nn = (struct node*) malloc(sizeof(
                      struct node));

    nn -> data = data;
    nn -> prev = NULL;
    nn -> next = NULL;
    return nn;

}

void insert(struct node **head, int data)
{
    if (*head == NULL)
    {
        *head = create(data);
    }
    else
    {
        (*head) -> prev = create(data)
        (*head) -> prev -> next = *head;
        *head = *head -> prev;
    }
}
```

```
void (struct node **head, int pos, int data)
{
  struct node *t = *head;
  while(--pos > 0 && t != NULL)
  { t = t->next;
  }
  (create(data))->prev = t;
  (create(data))->next = t->next;
  t->next->prev = create(data);
struct node *nn = create(data);

   t->next =
      nn->next = t->next;
      t->next->prev = nn;
      t = nn->prev;
      t->next = nn;

}

void insertend(struct node **head, int data){
  struct node *t = *head;
  while(t->next != NULL)
  { t = t->next;
  }
  struct node *nn = create(data);
    t->next = nn;
    nn->prev = t;
    nn->next = NULL;

  }

void show(struct node *head);
{
  struct node *t = head;
  while(t != NULL)
  { printf("%d ", t->data);
```

```
int main()
{ int c, v, p;  struct node *head = NULL;  while(1) {
printf("Enter operation 1. insert at beg
2. position, 3. end" 4-now & exit");
scanf("%d", &c);
  switch(c)
{
  case 1: printf("Enter data");
          scanf("%d", &v);
          insertbeg(&head, data);
          break;
  case 2: printf("Enter position &data");
          scanf("%d %d", &p, &v);
          void inspos(&head, p, v);
          break;
  case 3: printf("Enter data");
          scanf("%d", &v);
          void insertend(&head, v);
          break;
  case 4: show(head);
          break;
  case 5: exit(0);
  default: printf("Try again");
}
}
}
```

output

Menu:
Insert at beginning
Insert at pos
Insert at end
Display List
Exit

Enter choice 1
Enter data to insert at beginning 23

Menu:
1. Insert at beginning
2. Insert at position
3. Insert at end
4. display List
5. exit
~~Ento~~ 3
Insert data to insert at end 43

Menu
1. Insert beginin
2. Insert at position
3. Insert at end
~~Exit~~ 2.
Insert position & data 2 & 37

Menu
1 Insert at beginning.
2. Insert at position
3. Insertion at end
4 display
5. exit
4
2 3 3 7 4 3

Program 8

8)Write a program a) ToconstructabinarySearchtree. b) To traverse the tree using all the methods i.e., inorder, preorder and 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* node, int data) {
    if (node == NULL) {
        return createNode(data);
    }
    if (data < node->data) {
        node->left = insert(node->left, data);
    } else if (data > node->data) {
        node->right = insert(node->right, data);
    }
    return node;
}

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 choice, data;

    while (1) {
        printf("1. Insert Element  2. Inorder Traversal 3. Preorder Traversal 4. Postorder
Traversal 5. Exit ");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter data to insert: ");
                scanf("%d", &data);
                root = insert(root, data);
                break;
```

```
            case 2:
                printf("Inorder Traversal: ");
                inorder(root);
                printf("\n");
                break;
            case 3:
                printf("Preorder Traversal: ");
                preorder(root);
                printf("\n");
                break;
            case 4:
                printf("Postorder Traversal: ");
                postorder(root);
                printf("\n");
                break;
            case 5:
                exit(0);
            default:
                printf("Invalid choice\n");
        }
    }

    return 0;
}
```

OUTPUT:

```
PS C:\Users\hp\Desktop\cbeg> ./8c
 1. Insert Element  2. Inorder Traversal 3. Preorder Traversal 4. Postorder Traversal 5. Exit Enter your choice: 1
 Enter data to insert: 90
 1. Insert Element  2. Inorder Traversal 3. Preorder Traversal 4. Postorder Traversal 5. Exit Enter your choice: 1
 Enter data to insert: 45
 1. Insert Element  2. Inorder Traversal 3. Preorder Traversal 4. Postorder Traversal 5. Exit Enter your choice: 1
 Enter data to insert: 110
 1. Insert Element  2. Inorder Traversal 3. Preorder Traversal 4. Postorder Traversal 5. Exit Enter your choice: 1
 Enter data to insert: 30
 1. Insert Element  2. Inorder Traversal 3. Preorder Traversal 4. Postorder Traversal 5. Exit Enter your choice: 1
 Enter data to insert: 60
 1. Insert Element  2. Inorder Traversal 3. Preorder Traversal 4. Postorder Traversal 5. Exit Enter your choice: 120
 Invalid choice
 1. Insert Element  2. Inorder Traversal 3. Preorder Traversal 4. Postorder Traversal 5. Exit Enter your choice: 1
 Enter data to insert: 120
 1. Insert Element  2. Inorder Traversal 3. Preorder Traversal 4. Postorder Traversal 5. Exit Enter your choice: 2
 Inorder Traversal: 30 45 60 90 110 120
 1. Insert Element  2. Inorder Traversal 3. Preorder Traversal 4. Postorder Traversal 5. Exit Enter your choice: 3
 Preorder Traversal: 90 45 30 60 110 120
 1. Insert Element  2. Inorder Traversal 3. Preorder Traversal 4. Postorder Traversal 5. Exit Enter your choice: 4
 Postorder Traversal: 30 60 45 120 110 90
```

23/12

Bafna Gold
Date: Page:

1) WAP to Simulate BST 2) insert 3) execute inorder preorder postorder

```
struct node{
 int data;
 struct node *left;
 struct node *right;
 };

struct node *create(int data)
 {
  struct node *nn= (struct node *) malloc (sizeof
 (struct node));
    node->data = data;
     node->right=NULL;
      node->left = NULL;
      return nn;
    }
struct node *insert(struct node *root, int data)
  { if (root==NULL)
  {  return create(data);
  }
    if (data > root->data)
       {
root->right=insert (root->right, data);
       }
    else {
          root->left = insert(root->left, d
        return root;
}
```

```
void preorder(struct node *root){
    if(root==NULL){
        return;
    }

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

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

void disp(struct node *root){
printf("Inorder");
    inorder(root);
printf("preorder");
    preorder(root);}
```

```
printf("Postorder\n");
  postorder(root);
  }

Int main() {
 Struct node *root=NULL;
    int c, v;
 While(true) {
    printf("\n1. Insert 2. display 3. Exit");
    printf("Enter Choice");
      scanf("%d", &c);

    Switch(c)
    {
    case 1: printf("Enter value to insert:");
            scanf("%d", &value);
             root = insert(root, v);
             break;

    case 2: disp(root);
            break;

    case 3: exit(0);

    default = printf("Try again");
    }
    }

    return 0;
    }
```

BST o/p

1. Insert node
2. display Tree
3. Exit

Enter choice: 1
Enter value to insert 23

1. Insert node
2. display Tree
3. exit

Enter choice: 1
Enter value to insert 34

1. Insert node
2. display Tree
3. exit

Enter choice: 1
Enter value to insert 90

1. Insert node
2. display Tree
3. exit

Enter your choice: 2
Inorder: 23 34 90
preorder: 23 34 90
postorder: 90 34 23

24/12

Program 9

9) a) Write a program to traverse a graph using BFS method.
b) Write a program to check whether given graph is connected or not using DFS method.

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

int adj[MAX][MAX], visited[MAX], queue[MAX], front = -1, rear = -1;

// Function to enqueue an element
void enqueue(int vertex) {
    if (rear == MAX - 1)
        return;
    queue[++rear] = vertex;
    if (front == -1)
        front = 0;
}

// Function to dequeue an element
int dequeue() {
    if (front == -1)
        return -1;
    int vertex = queue[front++];
    if (front > rear)
        front = rear = -1;
    return vertex;
}

// BFS Function
void bfs(int start, int n) {
    int i;
    enqueue(start);
    visited[start] = 1;
```

```
    printf("BFS Traversal: ");
    while (front != -1) {
        int current = dequeue();
        printf("%d ", current);

        for (i = 0; i < n; i++) {
            if (adj[current][i] == 1 && !visited[i]) {
                enqueue(i);
                visited[i] = 1;
            }
        }
    }
    printf("\n");
}

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

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

// Main function
int main() {
    int n, e, u, v, start, choice;

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

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

```
// Initialize adjacency matrix
for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
        adj[i][j] = 0;

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

printf("Enter the starting vertex: ");
scanf("%d", &start);

do {
    // Reset visited array
    for (int i = 0; i < n; i++)
        visited[i] = 0;

    printf("\nChoose Traversal Method:\n");
    printf("1. BFS\n2. DFS\n3. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            bfs(start, n);
            break;
        case 2:
            printf("DFS Traversal: ");
            dfs(start, n);
            printf("\n");
            break;
        case 3:
            printf("Exiting...\n");
            break;
        default:
```

```
            printf("Invalid choice! Please try again.\n");
        }
    } while (choice != 3);

    return 0;
}
```

OUTPUT:

```
PS C:\Users\hp\Desktop\cbeg> ./9c
Enter the number of vertices: 6
Enter the number of edges: 5
Enter the edges (u v):
0 1
1 2
2 3
0 4
0 5
Enter the starting vertex: 0

Choose Traversal Method:
1. BFS
2. DFS
3. Exit
Enter your choice: 1
BFS Traversal: 0 1 4 5 2 3

Choose Traversal Method:
1. BFS
2. DFS
3. Exit
Enter your choice: 2
DFS Traversal: 0 1 2 3 4 5
```

23/12 DWAP to simulate DFS and BFS graph

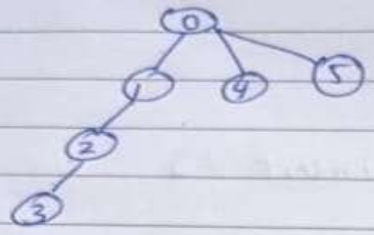
```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100

int adj[max][max], visited[max], queue[max],
    front=-1, rear=-1;

void enq(int v)
{
    if(rear == max-1)
        return;
    queue[++rear] = vertex;
    if(front == -1)
        front=0;
}
int dequeue(){
    if(front==-1)
        return -1;
    int v=queue[front++];
    front = rear=-1;
    return v;
}

void bfs(int start, int n){
    int i;
    enqueue(start);
    visited[start]=1;
    printf("BFS");
    while(front!=-1){
        int current=dequeue();
        printf("%d", current);
```

```
printf("Enter start");
scanf("%d", &start);

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

printf("choose method 1.B
                    3.Ex

scanf("%d", &c);

switch(c){
case 1: printf("BFS");
        bfs(start, n);
        break;
case 2: printf("DFS");
        dfs(start, n);
        break;
case 3: exit(0);
default: printf("Try again");
}
while(true);

return 0;
}
```

```
for (i=0; i<n; i++){
  if (adj[current][i] == 1 && ! visited[i]){
      enqueue(i);
      visited[i] = 1;
    }
  }
}
printf("\n");
}

void dfs(int v, int n){
  printf("%d ", v);
  visited[v] = 1;
  for (int i=0; i<n; i++){
    if (adj[v][i] == 1 && ! visited[i]){
        dfs(i, n);
    }
  }
}

int main(){
  int n, e, u, v, start, i;
  printf("Enter vertices & edges");
  scanf("%d%d", &n, &e);

  for(int adj[n][n] = {0};

  printf("Enter edges (u v):");
  for(int i=0; i<e; i++)
    scanf("%d %d", &u, &v);
    adj[u][v] = adj[v][u] = 1;
}
```

Bafna Gold
Date: Page:

Sample graph



Choose Traversal method
1. BFS
2. DFS
3. Exit

→ Enter number of vertices 6
Enter number of edge 5
Enter the edges (u v)
0 1
1 2
2 3
0 4
0 5
Enter starting vertex 0

BFS Traversal : BFS Traversal : 0 1 4 5 2 3
Choose Traversal Method
1. BFS
2. DFS
3. Exit
2
DFS Traversal: 0 1 2 3 4 5