VISVESVARAYATECHNOLOGICALUNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT on

Data Structures using C Lab (23CS3PCDST)

Submitted by

SHRINANDA SHIVPRASAD DINDE (1BM23CS324)

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 2023-2027

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 **SHRINANDA SHIVPRASAD DINDE (1BM23CS324),** 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.

Lab Faculty Prasad G.R Professor Department of CSE, BMSCE	Dr. Kavita Sooda Professor and Head Department of CSE, BMSCE

INDEX

GITHUB LINK: https://github.com/shrinanda27/DSA-C.git

Program 1

Write a program to simulate the working of stack using an array with the following: a) Push b) Pop c) display

→ push, pop and display.

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

#define SIZE 10
int A[SIZE];
int top = -1;

bool isEmpty (){
    return top == -1;
}
void push
bool isFull (){
    return top == SIZE - 1;
}
void push (int x){
    if (isFull()){
        printf (" stack overflow! unable to push %d \n", x);
        return;
    }
    top = top + 1;
    A[top] = x;
    printf (" Pushed %d onto the stack \n", x);
}
int POP(){
    if (isEmpty ()){
        printf (" stack underflow! unable to POP \n");
        return -1;
    }
    return A[top--];
}
```

```
int getTop() {
    if (isEmpty()) {
        Printf ("Stack is empty!")
        return -1;
    }
    return A[top];
}

Void display() {
    if (isEmpty()) {
        Printf ("Stack is empty\n");
        return;
    }
    Printf ("Stack Elements: ");
    for (int i = top; i >= 0; i--) {
        Printf ("%d", A[i]);
    }
    Printf ("\n");
}

int main() {
    int choice, value;
    While (1) {
        Printf ("Stack operations \n");
        Printf ("1. Push \n 2. Pop \n 3. Display \n
                4. get Top Element  5. exit);
        Printf ("Enter your choice: ");
        scanf ("%d", &choice);

Switch (choice) {
    case 1:
        Printf ("Enter value to push: ");
        scanf ("%d", &value);
```

```
            push(value);
            break;
        case 2:
            value = pop();
            if (value != -1){
                printf("Popped %d \n", value);
            }
            break;
        case 3:
            display();
            break;
        case 4:
            value = getTop();
            if (value != -1){
                Printf(" Top element: %d \n", value);
            }
            break;
        case 5:
            Printf(" exiting. \n");
            exit(0);
        default:
            Printf(" Invalid choice");
    }
}
return 0;
}
```

Seen

CODE

```
#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>
struct stack{
   int size;
   int top;
   int *arr;
};
bool isEmpty(struct stack *ptr){
   return ptr->top==-1;
}
bool isFull(struct stack *ptr){
   return ptr->top==ptr->size-1;
}
void push(int value, struct stack *ptr){
   if (isFull(ptr)){
      printf("Stack OverFlow!! cannot push %d into the stack\n", value);
      return;
   } else {
      ptr->top++;
      ptr->arr[ptr->top]=value;
      printf("%d is pushed into the stack.\n", value);
   }
}
int pop(struct stack *ptr){
   if (isEmpty(ptr)){
      printf("Stack UnderFlow!! cannot pop an element from the stack\n");
      return -1;
   } else {
      int value = ptr->arr[ptr->top];
      ptr->top--;
      printf("%d is popped from the stack.\n", value);
      return value;
   }
}
int peek(struct stack *ptr, int i){
   int arrayIndex = ptr->top-i+1;
```

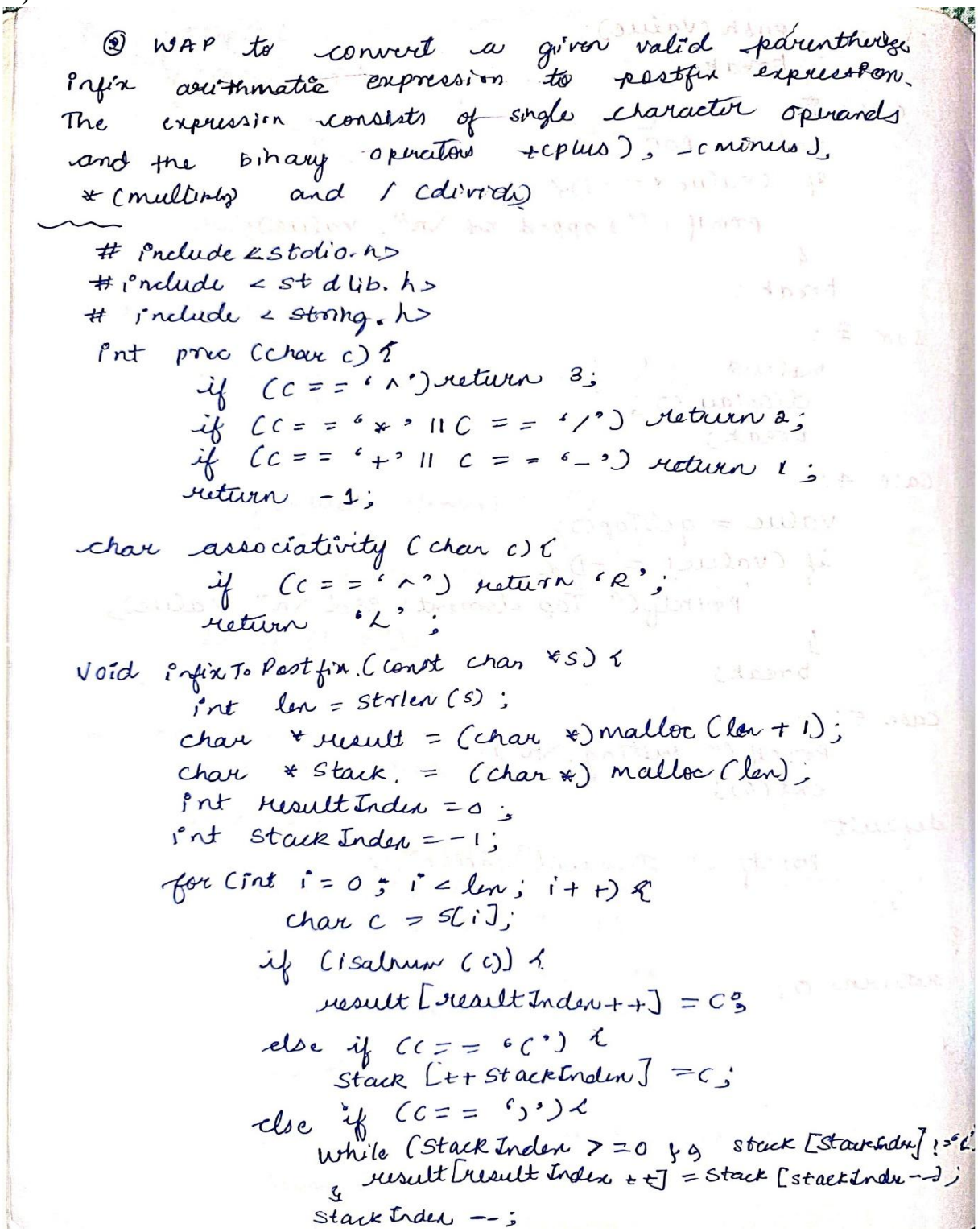
```
    if (arrayIndex<0){
        printf("Invalid Position\n");
        return -1;
    } else {
        return ptr->arr[arrayIndex];
    }
}
void display(struct stack *ptr){
    if (isEmpty(ptr)){
        printf("Stack is empty.\n");
        return;
    }
    printf("Stack elements: ");
    for (int i=0; i<=ptr->top; i++){
        printf("%d ", ptr->arr[i]);
    }
    printf("\n");
}
int stacktop(struct stack *ptr){
    return ptr->arr[ptr->top];
}
int stackbottom(struct stack *ptr){
    return ptr->arr[0];
}
void main(){
    struct stack *s = (struct stack *)malloc(sizeof(struct stack));
    s->size = 80;
    s->top = -1;
    s->arr = (int *)malloc(s->size*sizeof(int));
    printf("Stack has been created successfully!\n");
    int choice, element, i;
    while(1){
        printf("\nOptions: 1. Push  2. Pop  3. Peek  4. Display  5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch(choice){
            case 1:
                printf("Enter the element to push: ");
```

```
            scanf("%d", &element);
            push(element, s);
            break;
        case 2:
            pop(s);
            break;
        case 3:
            printf("Enter the position to peek: ");
            scanf("%d", &i);
            printf("Element at position %d is %d\n", i, peek(s, i));
            break;
        case 4:
            display(s);
            break;
        case 5:
            printf("Exiting...\n");
            exit(0);
        default:
            printf("Invalid choice! Please try again.\n");
    }
  }
}
```

```
Stack has been created successfully!

Options: 1. Push  2. Pop  3. Peek  4. Display  5. Exit
Enter your choice: 1
Enter the element to push: 10
10 is pushed into the stack.

Options: 1. Push  2. Pop  3. Peek  4. Display  5. Exit
Enter your choice: 1
Enter the element to push: 20
20 is pushed into the stack.

Options: 1. Push  2. Pop  3. Peek  4. Display  5. Exit
Enter your choice: 1
Enter the element to push: 30
30 is pushed into the stack.

Options: 1. Push  2. Pop  3. Peek  4. Display  5. Exit
Enter your choice: 4
Stack elements: 10 20 30

Options: 1. Push  2. Pop  3. Peek  4. Display  5. Exit
Enter your choice: 3
Enter the position to peek: 2
Element at position 2 is 20

Options: 1. Push  2. Pop  3. Peek  4. Display  5. Exit
Enter your choice: 2
30 is popped from the stack.

Options: 1. Push  2. Pop  3. Peek  4. Display  5. Exit
Enter your choice: 4
Stack elements: 10 20

Options: 1. Push  2. Pop  3. Peek  4. Display  5. Exit
Enter your choice: 5
Exiting...
```

Lab program 2

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



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

#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int prec(char c){
    if(c=='^')return 3;
    if(c=='*'||c=='/') return 2;
    if(c=='+'||c=='-') return 1;
    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;
    for(int i=0;i<len;i++){
        char c=s[i];
        if(isalnum(c)){
            result[resultIndex++]=c;
        }
        else if(c=='('){
            stack[++stackIndex]=c;
        }
        else if(c==')'){
            while(stackIndex>=0&&stack[stackIndex]!='('){
                result[resultIndex++]=stack[stackIndex--];
            }
            stackIndex--;
```

```
else {
    while (stackIndex >= 0 && (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[] = "a+b*(c^d-e)^(f+g*h)-i";
    printf("Expression = %s\n", exp);
    infix To Postfix(exp);
    return 0;
}
```

Seer

```
// output
expression = a+b*(c^d-e)^(f+g*h)-i
abcd^e-fgh*+^*+i-
```

CODE

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
struct stack{
    int size;
    int top;
    char *arr;
};
int stackTop(struct stack* sp){
    return sp->arr[sp->top];
}
int isEmpty(struct stack *ptr){
    return ptr->top == -1;
}
int isFull(struct stack *ptr){
    return ptr->top == ptr->size - 1;
}
void push(struct stack* ptr, char val){
    if(isFull(ptr)){
        printf("Stack Overflow! Cannot push %c to the stack\n", val);
        return;
    } else {
        ptr->top++;
        ptr->arr[ptr->top] = val;
    }
}
char pop(struct stack* ptr){
    if(isEmpty(ptr)){
        printf("Stack Underflow! Cannot pop from the stack\n");
        return -1;
    } else {
        char val = ptr->arr[ptr->top];
        ptr->top--;
        return val;
    }
}
int precedence(char ch){
```

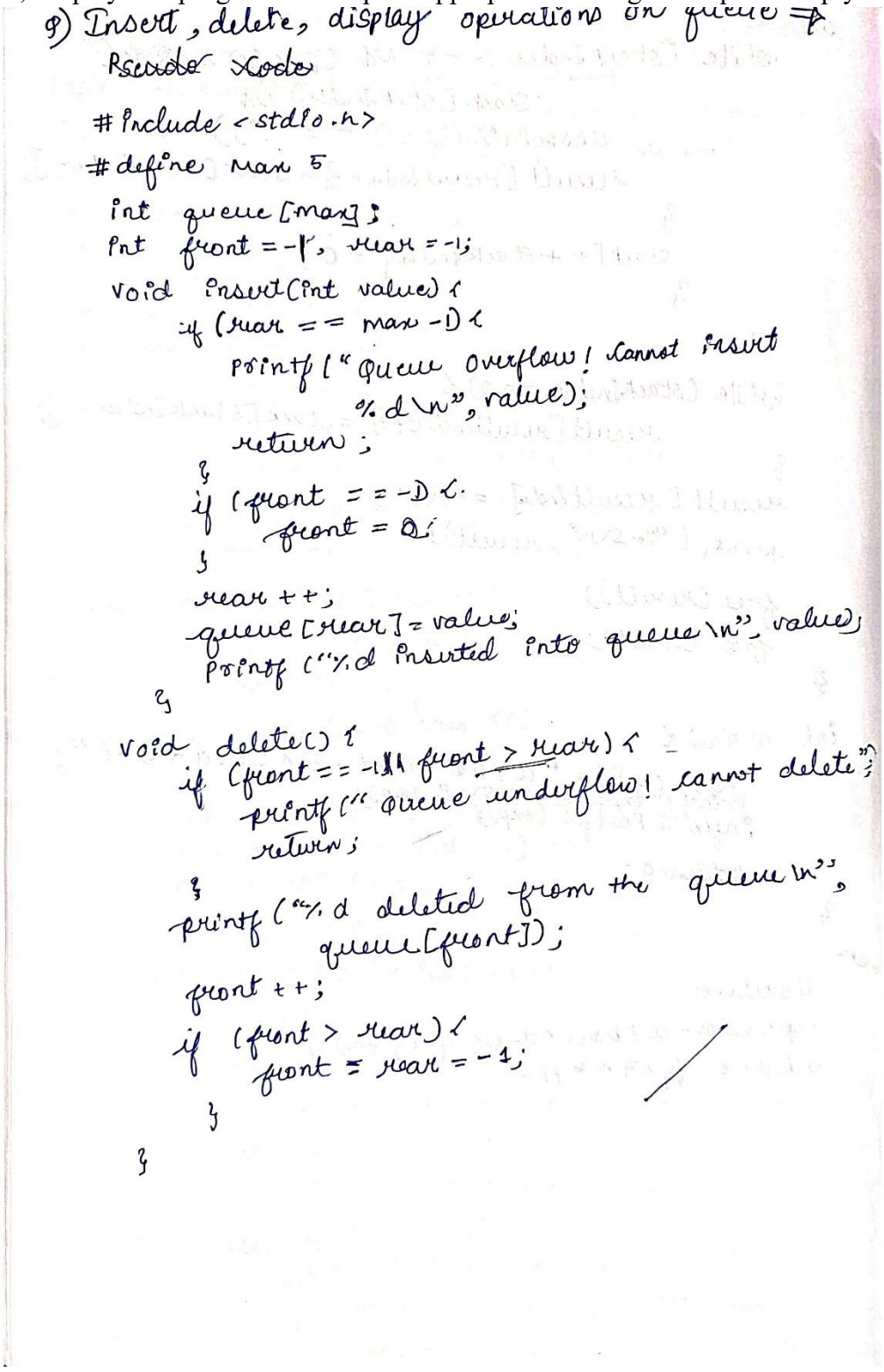
```
    if(ch == '*' || ch == '/')
        return 3;
    else if(ch == '+' || ch == '-')
        return 2;
    else
        return 0;
}
int isOperator(char ch){
    return ch == '+' || ch == '-' || ch == '*' || ch == '/';
}
char* infixToPostfix(char* infix){
    struct stack *sp = (struct stack*) malloc(sizeof(struct stack));
    sp->size = 20;
    sp->top = -1;
    sp->arr = (char *) malloc(sp->size * sizeof(char));
    char *postfix = (char*) malloc((strlen(infix) + 1) * sizeof(char));
    int i = 0;
    int j = 0;
    while(infix[i] != '\0'){
        if((infix[i] >= 'a' && infix[i] <= 'z') || (infix[i] >= 'A' && infix[i] <= 'Z')){
            postfix[j++] = infix[i++];
        }
        else if(infix[i] == '('){
            push(sp, infix[i]);
            i++;
        }
        else if(infix[i] == ')'){
            while(!isEmpty(sp) && stackTop(sp) != '('){
                postfix[j++] = pop(sp);
            }
            pop(sp);
            i++;
        }
        else{
            while(!isEmpty(sp) && precedence(stackTop(sp)) >= precedence(infix[i])){
                postfix[j++] = pop(sp);
            }
            push(sp, infix[i]);
```

```
            i++;
        }
    }
    while(!isEmpty(sp)){
        postfix[j++] = pop(sp);
    }
    postfix[j] = '\0';
    return postfix;
}
int main(){
    char *infix = "(x-y/z)*(k+d)";
    printf("Postfix expression: %s\n", infixToPostfix(infix));
    return 0;
}
```

```
Postfix expression: xyz/-kd+*
```

Lab programme 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 condition



9) Insert, delete, display operations on queue =>

Source Code

```
#include <stdio.h>
#define max 5
int queue[max];
int front = -1, rear = -1;
void insert(int value) {
    if (rear == max - 1) {
        printf("Queue Overflow! Cannot insert
                %d\n", value);
        return;
    }
    if (front == -1)
        front = 0;
    rear++;
    queue[rear] = value;
    printf("%d inserted into queue\n", value);
}
void delete() {
    if (front == -1 || front > rear) {
        printf("Queue underflow! cannot delete");
        return;
    }
    printf("%d deleted from the queue\n",
            queue[front]);
    front++;
    if (front > rear) {
        front = rear = -1;
    }
}
```

```
void display() {
    if (front == -1) {
        printf ("Queue is empty \n");
        return;
    }
    printf ("Queue elements : ");
    for (int i = front; i <= rear; i++) {
        printf ("%d", queue[i]);
    }
    printf ("\n");
}
int main() {
    int choice, value;
    while (1) {
        printf ("\n Queue Operations: \n");
        printf ("1. Insert \n");
        printf ("2. Delete \n");
        printf ("3 Display \n");
        printf ("4 Exit \n");
        printf ("Enter your choice: "
                "%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: printf ("exiting... \n");
                return 0;
```

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

//output

queue operations
1. Insert
2. delete
3. display
4. exit
enter your choice: 2
queue underflow ! cannot delete
queue operations
1. Insert
2. delete
3. display
4 exit
enter your choice: 1
value to insert: 1
.
.
.
.
.

enter your choice: 1
queue overflow can not insert!
que operations
1. Insert
2. delete
3. display
4 exit
enter your choi: 5
1 2 3 4 5
.

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

struct queue{
    int size;
    int f;
    int r;
    int *arr;
};
bool isFULL(struct queue *q){
    if(q->r == q->size - 1){
        return true;
    } else {
        return false;
    }
}
bool isEmpty(struct queue *q){
    if(q->r == q->f){
        return true;
    } else {
        return false;
    }
}
void enqueue(struct queue *q, int val){
    if(isFULL(q)){
        printf("Queue Overflow! Cannot enqueue %d into queue\n", val);
    } else {
        q->r = q->r + 1;
        q->arr[q->r] = val;
        printf("%d enqueued into the queue.\n", val);
    }
}
int dequeue(struct queue *q){
    if(isEmpty(q)){
        printf("Queue Underflow!\n");
        return -1;
    } else {
```

```
        q->f++;
        int val = q->arr[q->f];
        return val;
    }
}
void display(struct queue *q){
    if(isEmpty(q)){
        printf("Queue is empty!\n");
    } else {
        printf("Queue elements: ");
        for(int i = q->f + 1; i <= q->r; i++){
            printf("%d ", q->arr[i]);
        }
        printf("\n");
    }
}
int main(){
    struct queue q;
    q.size = 50; q.f = -1;
    q.r = -1;
    q.arr = (int *)malloc(q.size * sizeof(int));
    int choice, val;
    while(1){
        printf("\nOptions: 1. Enqueue  2. Dequeue  3. Display  4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch(choice){
            case 1:
                printf("Enter the element to enqueue: ");
                scanf("%d", &val);
                enqueue(&q, val);
                break;
            case 2:
                val = dequeue(&q);
                if(val != -1){
                    printf("Dequeued element: %d\n", val);
                }
                break;
```

```
            case 3:
                display(&q);
                break;
            case 4:
                printf("Exiting...\n");
                exit(0);
            default:
                printf("Invalid choice! Please try again.\n");
        }
    }
    return 0;
}
```

```
Options: 1. Enqueue  2. Dequeue  3. Display  4. Exit
Enter your choice: 1
Enter the element to enqueue: 10
10 enqueued into the queue.

Options: 1. Enqueue  2. Dequeue  3. Display  4. Exit
Enter your choice: 1
Enter the element to enqueue: 20
20 enqueued into the queue.

Options: 1. Enqueue  2. Dequeue  3. Display  4. Exit
Enter your choice: 1
Enter the element to enqueue: 30
30 enqueued into the queue.

Options: 1. Enqueue  2. Dequeue  3. Display  4. Exit
Enter your choice: 3
Queue elements: 10 20 30

Options: 1. Enqueue  2. Dequeue  3. Display  4. Exit
Enter your choice: 2
Dequeued element: 10

Options: 1. Enqueue  2. Dequeue  3. Display  4. Exit
Enter your choice: 3
Queue elements: 20 30

Options: 1. Enqueue  2. Dequeue  3. Display  4. Exit
Enter your choice: 4
Exiting...
```

LAB PROGRAM 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 messagesfor queue empty and queue overflow conditions

circular queue

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

struct circularqueue {
    int size;
    int f;
    int r;
    int *arr;
};

bool isFull(struct circularqueue *q) {
    if ((q->r+1)% (q->size) == q->f) {
        return true;
    }
    else {
        return false;
    }
}

bool isEmpty(struct circularqueue *q) {
    if (q->r == -1) {
        return true;
    }
    else {
        return false;
    }
}
```

```
void enqueue (struct circularqueue *q, int val)
    if (isFull(q)) {
        printf ("queue overflow \n");
    }
    else {
        if (q->r == -1) {
            q->f = q->r = 0;
        }
        else {
            q->r = (q->r + 1) % q->size;
        }
        q->arr[q->r] = val;
    }
}

int dequeue (struct circularqueue *q) {
    if (isEmpty(q)) {
        printf ("queue is empty! \n");
        return -1;
    }
    else {
        int val = q->arr[q->f];
        if (q->f == q->r) {
            q->f = q->r = -1;
        }
        else {
            q->f = (q->f + 1) % q->size;
        }
        return val;
    }
}
```

```
void display (struct circularqueue *q) {
    if (isEmpty (q)) {
        printf ("queue is empty! \n");
        return;
    }
    else {
        int i = q -> f;
        printf ("Queue elements : ");
        while (i != q -> r) {
            printf ("%d ", q -> arr [i]);
            i = (i+1) % q -> size ;
        }
        Printf ("%d \n", q -> arr [i]);
}

int main () {
    Struct circularqueue q;
    q. size = 5 ;
    q. f = q. r = -1;
    q. arr = (int *) malloc (q. size * sizeof (int));
    enqueue (&q, 10);
    enqueue (&q, 11);
    enqueue (&q, 12);
    enqueue (&q, 13);
    display (&q);
    prentf ("%d \n", dequeue (&q));
    ~~prentf (&q);~~
```

```
enqueue(&q, 14);
display(&q);

enqueue(&q, 14);
display(&q);
printf("%d\n", dqueue(&q));
display(&q);
return 0;
}
```

// Output

```
10 11 12 13
10
11 12 13 14
11 12 13 14 15
11
12 13 14 15
```

CODE

```
#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>
struct circularqueue{
   int size;
   int f;
   int r;
   int *arr;
};
bool isFull(struct circularqueue *q){
   if((q->r + 1) % q->size == q->f){
      return true;
   } else {
      return false;
   }
}
bool isEmpty(struct circularqueue *q){
   if(q->r == -1){
      return true;
   } else {
      return false;
   }
}
void enqueue(struct circularqueue *q, int val){
   if(isFull(q)){
      printf("Queue OverFlow\n");
   } else {
      if(q->r == -1){
         q->f = q->r = 0;
      } else {
         q->r = (q->r + 1) % q->size;
      }
      q->arr[q->r] = val;
      printf("%d enqueued into the queue.\n", val);
   }
}
int dequeue(struct circularqueue *q){
```

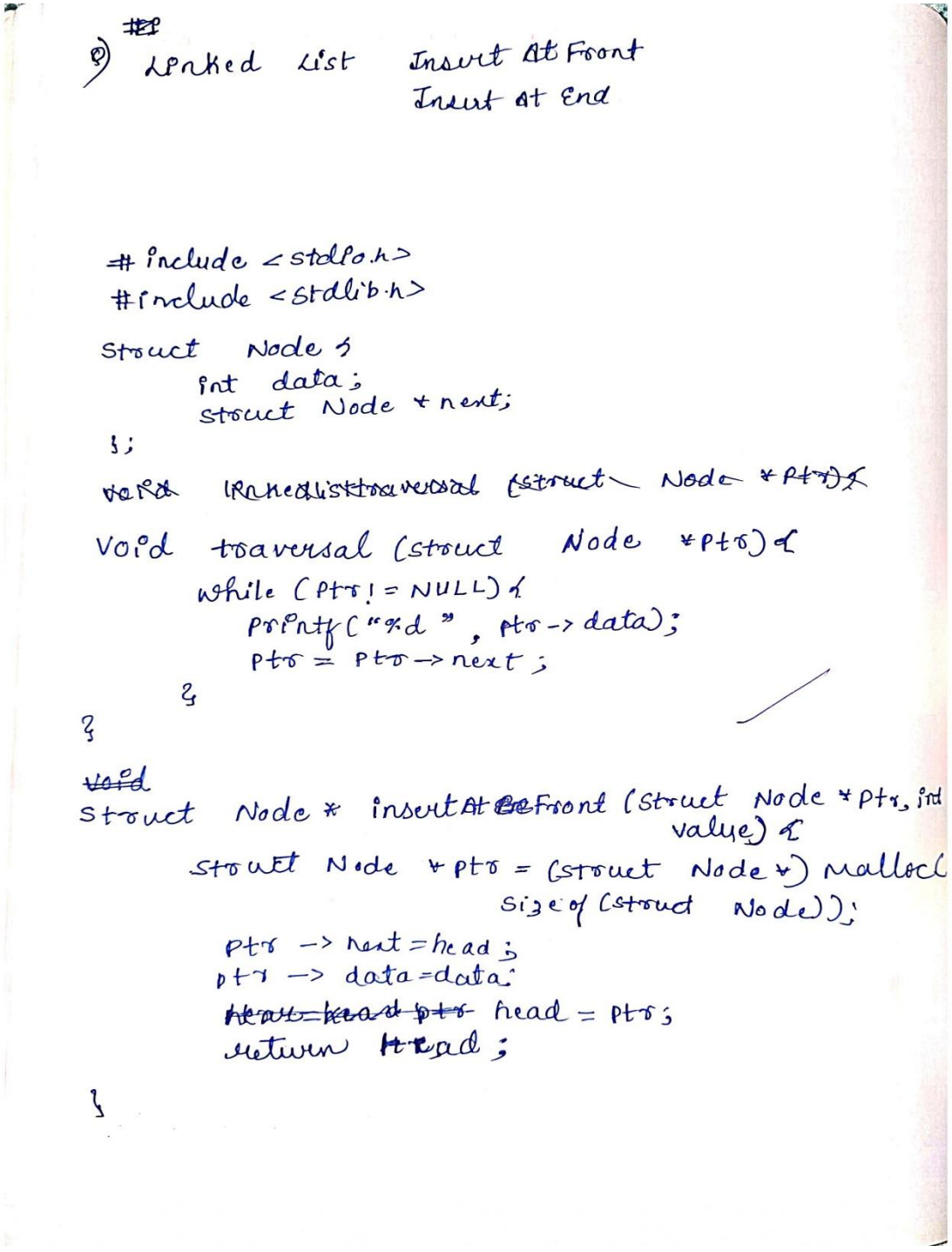
```
    if(isEmpty(q)){
        printf("Queue is empty!\n");
        return -1;
    } else {
        int val = q->arr[q->f];
        if(q->f == q->r){
            q->f = q->r = -1;
        } else {
            q->f = (q->f + 1) % q->size;
        }
        return val;
    }
}
void display(struct circularqueue *q){
    if(isEmpty(q)){
        printf("Queue is empty!\n");
        return;
    }
    int i = q->f;
    printf("Queue elements: ");
    while(i != q->r){
        printf("%d ", q->arr[i]);
        i = (i + 1) % q->size;
    }
    printf("%d\n", q->arr[i]);
}
int main(){
    struct circularqueue q;
    q.size = 5;
    q.f = q.r = -1;
    q.arr = (int*) malloc(q.size * sizeof(int));
    int choice, val;
    while(1){
        printf("\nOptions: 1. Enqueue  2. Dequeue  3. Display  4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch(choice){
            case 1:
```

```
            printf("Enter the element to insert: ");
            scanf("%d", &val);
            enqueue(&q, val);
            break;
        case 2:
            val = dequeue(&q);
            if(val != -1){
                printf("Deleted element: %d\n", val);
            }
            break;
        case 3:
            display(&q);
            break;
        case 4:
            printf("Exiting\n");
            exit(0);
        default:
            printf("Invalid choice!\n");
      }
   }
   return 0;
}
```

```
Options: 1. Enqueue  2. Dequeue  3. Display  4. Exit
Enter your choice: 1
Enter the element to insert: 10
10 enqueued into the queue.

Options: 1. Enqueue  2. Dequeue  3. Display  4. Exit
Enter your choice: 1
Enter the element to insert: 20
20 enqueued into the queue.

Options: 1. Enqueue  2. Dequeue  3. Display  4. Exit
Enter your choice: 3
Queue elements: 10 20

Options: 1. Enqueue  2. Dequeue  3. Display  4. Exit
Enter your choice: 2
Deleted element: 10

Options: 1. Enqueue  2. Dequeue  3. Display  4. Exit
Enter your choice: 4
Exiting
```

LAB PROGRAM 4

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



9) Linked List Insert At Front
Insert At End

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

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

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

struct Node * insertAtFront (struct Node *ptr, int value) {
    struct Node *ptr = (struct Node *) malloc(sizeof(struct Node));
    ptr->next = head;
    ptr->data = data;
    head = ptr;
    return head;
}
```

```
struct Node * InsertAtEnd(struct Node * head,
                          int data) {
    struct Node * ptr = (struct Node *) malloc(
                        sizeof(struct Node));
    ptr -> data = data;
    struct Node * p = head;
    while (p -> next != NULL) {
        p = p -> next;
    }
    p -> next = ptr;
    ptr -> next = NULL;
    return head;
}
void main() {
    struct Node * head = NULL;
    int choice, data;
    while (1) {
        printf("\noperations\n");
        printf("1. Insert At beginning \n");
        printf("2. Insert At end \n");
        printf("3. Traversal \n");
        printf("4. Exit \n");
        printf("Enter your choice:");
        scanf("%d", &choice);
        switch(choice) {
            case 1:
                printf("enter data");
                scanf("%d", &data);
                head = insertAtFirst(head, data);
                break;
```

```
        case 2:
            printf("enter data to insert at the end:");
            scanf("%d", &data);
            head = insertAtend(head, data);
            break;
        case 3:
            printf("Linked List: ");
            traversal(head);
            printf("\n");
            break;
        case 4:
            exit(0);
        default:
            printf("Invalid choice!");
    }
}
```

~~return~~

```
// output
```

Operations
1. Insert at the beginning
2. Insert at the end
3. Traversal
4. exit

enter your choice: 1
enter data to insert at the beginning = 45

operations:
1. Insert at the beginning
2. Insert at the end
3. Traversal
4. ext

Enter your choice: 2
enter data to insert at end =4

operations:
1. Insert at the beginning.
2. Insert at the end.
3. Traversal
4. Exit.
enter your choice: 3
Linked list: 45 4

operations:
1. Insert at the beginning
2. Insert at the end
3. Traversal
4. Exit
Enter your choice: 4

Code

```
#include <stdio.h>
#include <stdlib.h>
struct Node{
    int data;
    struct Node *next;
};
void linkedlisttraversal(struct Node *ptr){
    while(ptr != NULL){
        printf("%d ", ptr->data);
        ptr = ptr->next;
    }
}
struct Node *insertAtFirst(struct Node *head, int data){
    struct Node *ptr = (struct Node *)malloc(sizeof(struct Node));
    ptr->next = head;
    ptr->data = data;
    head = ptr;
    return head;
}
struct Node *insertAtEnd(struct Node *head, int data){
    struct Node *ptr = (struct Node *)malloc(sizeof(struct Node));
    ptr->data = data;
    ptr->next = NULL;
    if(head == NULL){
        return ptr;
    }
    struct Node *p = head;
    while(p->next != NULL){
        p = p->next;
    }
    p->next = ptr;
    return head;
}
int main(){
    struct Node *head = NULL;
    int choice, data;
    while(1){
        printf("\nMenu:\n");
```

```
        printf("1. Insert at the beginning\n");
        printf("2. Insert at the end\n");
        printf("3. Traverse the list\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch(choice){
            case 1:
                printf("Enter data to insert at the beginning: ");
                scanf("%d", &data);
                head = insertAtFirst(head, data);
                break;
            case 2:
                printf("Enter data to insert at the end: ");
                scanf("%d", &data);
                head = insertAtEnd(head, data);
                break;
            case 3:
                printf("Linked List: ");
                linkedlisttraversal(head);
                printf("\n");
                break;
            case 4:
                exit(0);
            default:
                printf("Invalid choice! Please try again.\n");
        }
    }
    return 0;
}
```

```
Menu:
1. Insert at the beginning
2. Insert at the end
3. Traverse the list
4. Exit
Enter your choice: 1
Enter data to insert at the beginning: 10

Menu:
1. Insert at the beginning
2. Insert at the end
3. Traverse the list
4. Exit
Enter your choice: 2
Enter data to insert at the end: 20

Menu:
1. Insert at the beginning
2. Insert at the end
3. Traverse the list
4. Exit
Enter your choice: 3
Linked List: 10 20

Menu:
1. Insert at the beginning
2. Insert at the end
3. Traverse the list
4. Exit
Enter your choice: 4
```

LAB PROGRAM 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

```
Q) a) Create a linked list
   b) Deletion of first element, specified
      and last element
   c) Display.

#include<stdio.h>

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

struct Node* head = NULL;
struct Node *
void create (int data, struct Node *hea d){
    struct Node* newNode = (struct Node *) malloc
                (sizeof (struct Node));
    new Node ->data = data;
    newNode -> next = NULL;
    if (head == NULL) {
          head = newNode;
    } else {
          Struct Node * temp = head;
          while (temp -> next -> next != NULL) {
                temp = temp -> next;
          }
          temp ->next = newNode;
    }
    return head;
}
```

```
Struct Node * deleteFirst( struct Node * head);
    if (head == NULL) return NULL;
    struct Node * ptr = head;
    head = head -> next;
    free (Ptr);
    return head;
}

Struct Node * deleteAtIndex(Struct Node *
                    head, int index) {
    if (head == NULL || index < 0) {
        return head;
    }
    Struct Node *p = head;
    struct Node *q = head -> next;
    if (index == 0) return deleteFirst (head);
    for (int i=0; i < index -1; i++) {
        if (q == NULL) return head;
        p = p -> next;
        q = q -> next;
    }
    p->next = q -> next
    return head;
}
```

```
struct Node* deleteAtLast ( struct Node *head){
  if (head == NULL) return NULL;
  if (head -> next == NULL) {
      free (head);
      return NULL;
  }
  else {
  Struct Node * temp = head;
  struct Node * q = head -> next;

  while (q -> next != NULL){
      P = P-> next;
      q = q -> next;
   }
   p -> next = NULL;
    free (q);
   return head;
}
void display (struct Node * head) {
   struct node * temp = head;
   While (temp != NULL) {
     printf ("%d", temp -> data);
      temp = temp -> next;
   }
   Printf ("\n");
}
```

CODE

```
#include <stdio.h>
#include <stdlib.h>
struct Node {int data; struct Node *next;};
struct Node* head = NULL;
struct Node* create(int data, struct Node *head) {
   struct Node *newnode = (struct Node*)malloc(sizeof(struct Node));
   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;
   }
   return head;
}
struct Node* deletefirst(struct Node *head) {
   if (head == NULL) {printf("LINKED LIST is empty"); return NULL;}
   struct Node *ptr = head; head = head->next; free(ptr); return head;
}
struct Node* deleteatindex(struct Node *head, int index) {
   if (head == NULL || index < 0) {printf("LINKED LIST is empty"); return head;}
   if (index == 0) return deletefirst(head);
   struct Node *p = head, *q = head->next;
   for (int i = 0; i < index - 1; i++) {
      if (q == NULL) {printf("index out of range\n"); return head;}
      p = p->next; q = q->next;
   }
   p->next = q->next; free(q); return head;
}
struct Node* deleteatlast(struct Node *head) {
   if (head == NULL) {printf("LINKED LIST is empty"); return NULL;}
   else if (head->next == NULL) {free(head); return NULL;}
   else {
      struct Node *p = head, *q = head->next;
      while (q->next != NULL) {p = p->next; q = q->next;}
      p->next = NULL; free(q);
   }
   return head;
```

```
}
void display(struct Node *head) {
   struct Node *temp = head;
   while (temp != NULL) {printf("%d ", temp->data); temp = temp->next;}
   printf("NULL\n");
}
int main() {
   int choice, data, idx;
   while (1) {
      printf("SELECT:\n1. CREATE\n2. DELETE AT FIRST\n3. DELETE AT INDEX\n4. DELETE AT
LAST\n5. DISPLAY\n6. EXIT\n");
      scanf("%d", &choice);
      switch (choice) {
         case 1: printf("Enter data to create a new node: "); scanf("%d", &data); head = create(data, head); break;
         case 2: head = deletefirst(head); break;
         case 3: printf("Enter the index to delete: "); scanf("%d", &idx); head = deleteatindex(head, idx); break;
         case 4: head = deleteatlast(head); break;
         case 5: printf("Linked List: "); display(head); break;
         case 6: printf("Exiting.....\n"); exit(0);
         default: printf("Invalid choice!!!!!\n");
      }
   }
   return 0;
}
```

```
SELECT:
1. CREATE
2. DELETE AT FIRST
3. DELETE AT INDEX
4. DELETE AT LAST
5. DISPLAY
6. EXIT
1
Enter data to create a new node: 10

SELECT:
1. CREATE
2. DELETE AT FIRST
3. DELETE AT INDEX
4. DELETE AT LAST
5. DISPLAY
6. EXIT
1
Enter data to create a new node: 20

SELECT:
1. CREATE
2. DELETE AT FIRST
3. DELETE AT INDEX
4. DELETE AT LAST
5. DISPLAY
6. EXIT
5
Linked List: 10 20 NULL

SELECT:
1. CREATE
2. DELETE AT FIRST
3. DELETE AT INDEX
4. DELETE AT LAST
5. DISPLAY
6. EXIT
4

SELECT:
1. CREATE
2. DELETE AT FIRST
3. DELETE AT INDEX
4. DELETE AT LAST
5. DISPLAY
6. EXIT
5
Linked List: 10 NULL

SELECT:
1. CREATE
2. DELETE AT FIRST
3. DELETE AT INDEX
4. DELETE AT LAST
5. DISPLAY
6. EXIT
6
Exiting.......
```

LAB PROGRAM 6a

WAP to Implement Single Link List with following operations: Sortthelinkedlist, Reversethelinkedlist, Concatenation of two linked lists.

6 a) WAP to implement Single Link list of following operations:
Sort the linked list
Reverse the linked list
Concatenation of two linked list.

6 b) WAP to implement Single Link list to stimulate stack & queue operations.

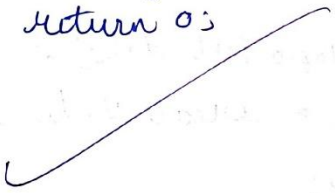
6a

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

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

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

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

```
void reverseList(Node** head) {
    Node *prev = NULL, *current = *head;
    Node *Next = NULL;
    While (current != NULL) {
        next = current -> next;
        current -> next = prev;
        prev = current;
        current = Next;
    }
    *head = prev;
}

Node * concatenateLists (Node * head1, Node*hea
    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;
}

int main () {
    Node *list1 = NULL, * list2 = NULL;
    insertEnd (& list1, 3);
    insertEnd (&list1, 1);
    insertEnd (& list1, 5);
    Printf ("original List 1: ");
    displayList (list1);
    SortList (& list1);
    printf ("Sorted List 1: ");
    displayList (list1);
```

```
// Reverse
reverseList(&list1);
printf("Reversed List 1: ");
displayList(list1);

// Insert
insertEnd(&list2, 10);
insertEnd(&list2, 20);
printf("List 2: ");
displayList(list2);

// Concat

Con
Node* concatenated = concatenateLists(list1, list2);
printf("Concatenated List: ");
displayList(concatenated);
return 0;
}
```



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 insertEnd(struct Node** head, int data) {
   struct Node* newNode = createNode(data);
   if (*head == NULL) {*head = newNode; return;}
   struct Node* temp = *head;
   while (temp->next != NULL) temp = temp->next;
   temp->next = newNode;
}
void printList(struct Node* head) {
   struct Node* temp = head;
   while (temp != NULL) {printf("%d -> ", temp->data); temp = temp->next;}
   printf("NULL\n");
}
void sortList(struct Node* head) {
   struct Node* i = head; struct Node* j; int temp;
   while (i != NULL) {
      j = i->next;
      while (j != NULL) {
         if (i->data > j->data) {temp = i->data; i->data = j->data; j->data = temp;}
         j = j->next;
      }
      i = i->next;
   }
}
void reverseList(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 concatenateLists(struct Node** head1, struct Node* head2) {
   if (*head1 == NULL) {*head1 = head2; return;}
   struct Node* temp = *head1;
```

```
    while (temp->next != NULL) temp = temp->next;
    temp->next = head2;
}
int main() {
    struct Node* list1 = NULL; struct Node* list2 = NULL;
    int choice, data;
    do {
        printf("\nMenu:\n1. Insert element into List 1\n2. Insert element into List 2\n3. Sort List 1\n4. Reverse List 1\n5. Concatenate List 1 and List 2\n6. Print List 1\n7. Print List 2\n8. Exit\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1: printf("Enter data to insert into List 1: "); scanf("%d", &data); insertEnd(&list1, data); break;
            case 2: printf("Enter data to insert into List 2: "); scanf("%d", &data); insertEnd(&list2, data); break;
            case 3: sortList(list1); printf("List 1 sorted.\n"); break;
            case 4: reverseList(&list1); printf("List 1 reversed.\n"); break;
            case 5: concatenateLists(&list1, list2); printf("List 2 concatenated to List 1.\n"); break;
            case 6: printf("List 1: "); printList(list1); break;
            case 7: printf("List 2: "); printList(list2); break;
            case 8: printf("Exiting...\n"); break;
            default: printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 8);
    return 0;
}
```

```
Menu:
1. Insert element into List 1
2. Insert element into List 2
3. Sort List 1
4. Reverse List 1
5. Concatenate List 1 and List 2
6. Print List 1
7. Print List 2
8. Exit
Enter your choice: 1
Enter data to insert into List 1: 5

Menu:
1. Insert element into List 1
2. Insert element into List 2
3. Sort List 1
4. Reverse List 1
5. Concatenate List 1 and List 2
6. Print List 1
7. Print List 2
8. Exit
Enter your choice: 1
Enter data to insert into List 1: 3

Menu:
1. Insert element into List 1
2. Insert element into List 2
3. Sort List 1
4. Reverse List 1
5. Concatenate List 1 and List 2
6. Print List 1
7. Print List 2
8. Exit
Enter your choice: 3
List 1 sorted.

Menu:
1. Insert element into List 1
2. Insert element into List 2
3. Sort List 1
4. Reverse List 1
5. Concatenate List 1 and List 2
6. Print List 1
7. Print List 2
8. Exit
Enter your choice: 6
List 1: 3 -> 5 -> NULL

Menu:
1. Insert element into List 1
2. Insert element into List 2
3. Sort List 1
4. Reverse List 1
5. Concatenate List 1 and List 2
6. Print List 1
7. Print List 2
8. Exit
Enter your choice: 8
Exiting...
```

LAB PROGRAM 6b

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

```
6b)
#include <stdio.h>
#include <stdlib.h>

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

Struct Node* create Node (int data) {
    Struct Node*new Node = (Struct Node *)
                malloc (size of (struct Node));
    new Node -> data = data;
    new Node -> next = NULL;
    return new Node;
}

Void push (strud Node **top, int data) {
    struct Node* new Node = create Node (data);
    new Node ->next = * top;
    *top = new Node;
    printf ("pushed %d in stack \n", data);
}

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

```
void enqueue(struct Node **rear, struct Node
                **front, int data){
    struct Node* newNode = createNode(data);
    if (*rear == NULL){
        *rear = *front = newNode;
    }
    else {
        (*rear)->next = newNode;
        *rear = newNode;
    }
    printf("Enqueued %d\n", data);
}
int dequeue(struct Node **front, struct Node **rear){
    if (*front == NULL){
        printf("Queue is empty\n");
        return -1;
    }
    int data = (*front)->data;
    struct Node *temp = *front;
    *front = (*front)->next;
    if (*front == NULL){
        *rear = NULL; }
    free(temp);
    printf("Dequeued %d from queue\n", data);
    return data;
}
```

```
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");
}
```

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 push(struct Node** top, int data) {
   struct Node* newNode = createNode(data);
   newNode->next = *top; *top = newNode; printf("Pushed %d onto stack\n", data);
}
int pop(struct Node** top) {
   if (*top == NULL) {printf("Stack is empty\n"); return -1;}
   int data = (*top)->data; struct Node* temp = *top; *top = (*top)->next; free(temp);
   printf("Popped %d from stack\n", data); return data;
}
void enqueue(struct Node** rear, struct Node** front, int data) {
   struct Node* newNode = createNode(data);
   if (*rear == NULL) {*rear = *front = newNode;} else {(*rear)->next = newNode; *rear = newNode;}
   printf("Enqueued %d into queue\n", data);
}
int dequeue(struct Node** front, struct Node** rear) {
   if (*front == NULL) {printf("Queue is empty\n"); return -1;}
   int data = (*front)->data; struct Node* temp = *front; *front = (*front)->next;
   if (*front == NULL) {*rear = NULL;} free(temp); printf("Dequeued %d from queue\n", data); return data;
}
void display(struct Node* head) {
   if (head == NULL) {printf("List is empty\n"); return;}
   struct Node* temp = head; while (temp != NULL) {printf("%d -> ", temp->data); temp = temp->next;}
printf("NULL\n");
}
int main() {
   struct Node* stack = NULL; struct Node* front = NULL; struct Node* rear = NULL;
   int choice, data; while (1) {
      printf("\nMenu:\n1. Push to Stack\n2. Pop from Stack\n3. Display Stack\n4. Enqueue to Queue\n5.
Dequeue from Queue\n6. Display Queue\n7. Exit\nEnter your choice: ");
      scanf("%d", &choice);
      switch (choice) {
```

```
            case 1: printf("Enter data to push onto stack: "); scanf("%d", &data); push(&stack, data); break;
            case 2: pop(&stack); break;
            case 3: printf("Stack: "); display(stack); break;
            case 4: printf("Enter data to enqueue into queue: "); scanf("%d", &data); enqueue(&rear, &front, data);
break;
            case 5: dequeue(&front, &rear); break;
            case 6: printf("Queue: "); display(front); break;
            case 7: exit(0);
            default: printf("Invalid choice. Please try again.\n");
        }
    }
    return 0;
}
```

```
Menu:
1. Push to Stack
2. Pop from Stack
3. Display Stack
4. Enqueue to Queue
5. Dequeue from Queue
6. Display Queue
7. Exit
Enter your choice: 1
Enter data to push onto stack: 10
Pushed 10 onto stack

Enter your choice: 4
Enter data to enqueue into queue: 20
Enqueued 20 into queue

Enter your choice: 3
Stack: 10 -> NULL

Enter your choice: 6
Queue: 20 -> NULL

Enter your choice: 2
Popped 10 from stack

Enter your choice: 5
Dequeued 20 from queue

Enter your choice: 7
```

PROGRAM 7

7. WAP to Implement doubly link list with primitive operations

a) Create a doubly linked list.

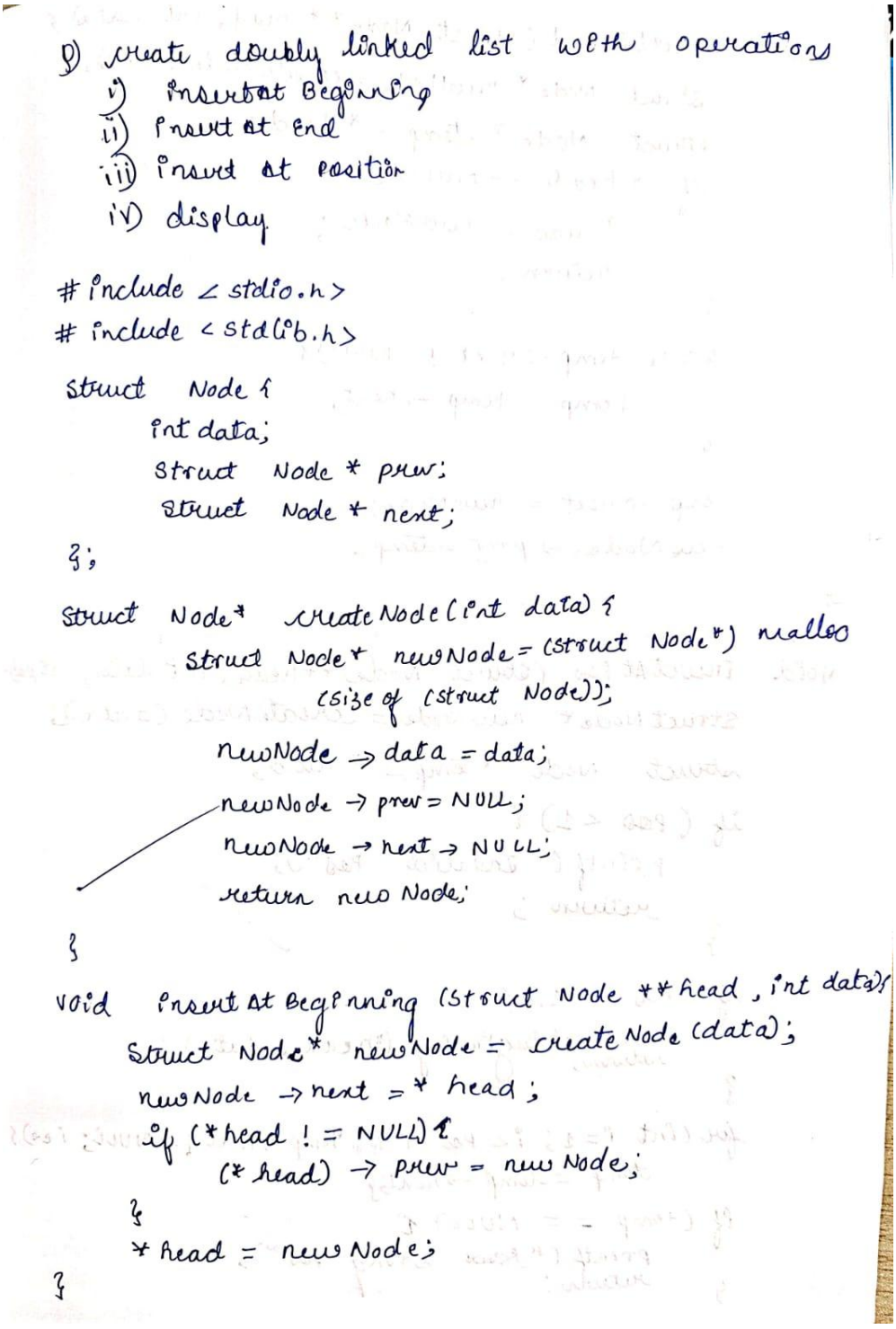
b) Insert a new node at the beginning.

c) Insert the node based on a specific location

d)

Insert a new node at the end.

e) Display the contents of the list



```
7) create doubly linked list with operations
   i) insert at Beginning
   ii) insert at end
   iii) insert at position
   iv) display

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

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

void insertAtPos (struct Node **head, int data, int
    Struct Node * newNode = createNode (data);
    struct Node * temp = * head;
    if ( pos < 1) {
        printf (" Invalid Pos ");
        return ;
    }
    if (pos == 1) {
        Insert at beginning (*head, data);
        return;
    }
    for (int i = 1; i < pos - 1 && temp -> next != NULL; i++)
        temp = temp -> next;}
    If (temp == NULL) {
        printf (" Wrong pos ");
    }  return;
```

```
    newNode -> next = temp -> next;
    newNode -> prev = temp;
    temp
    newNode -> next -> prev = temp, newNode;
    newtemp -> next = newNode;
}

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

```
#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)->prev = newNode;
    }
    newNode->next = *head;
    *head = newNode;
    printf("Inserted %d at the beginning.\n", data);
}

void insertAtLocation(struct Node** head, int data, int location) {
    struct Node* newNode = createNode(data);
    if (location == 1) {
        insertAtBeginning(head, data);
        return;
    }
    struct Node* temp = *head;
    for (int i = 1; i < location - 1 && temp != NULL; i++) {
        temp = temp->next;
    }
    if (temp == NULL) {
```

```
        printf("Location out of range.\n");
        return;
    }
    newNode->next = temp->next;
    if (temp->next != NULL) {
        temp->next->prev = newNode;
    }
    temp->next = newNode;
    newNode->prev = temp;
    printf("Inserted %d at location %d.\n", data, location);
}

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

void display(struct Node* head) {
    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, data, location;
    while (1) {
        printf("\n1. Create a doubly linked list\n2. Insert at beginning\n3. Insert at location\n4. Insert at end\n5. Display\n6. Exit\n");
        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 at beginning: ");
                scanf("%d", &data);
                insertAtBeginning(&head, data);
                break;
            case 3:
                printf("Enter data to insert: ");
                scanf("%d", &data);
                printf("Enter location to insert at: ");
                scanf("%d", &location);
                insertAtLocation(&head, data, location);
                break;
            case 4:
                printf("Enter data to insert at end: ");
                scanf("%d", &data);
                insertAtEnd(&head, data);
                break;
            case 5:
                display(head);
                break;
            case 6:
                exit(0);
            default:
                printf("Invalid choice!\n");
        }
    }
    return 0;
```

}

```
1. Create a doubly linked list
2. Insert at beginning
3. Insert at location
4. Insert at end
5. Display
6. Exit
Enter your choice: 1
Enter data to insert at beginning: 10
Inserted 10 at the beginning.

1. Create a doubly linked list
2. Insert at beginning
3. Insert at location
4. Insert at end
5. Display
6. Exit
Enter your choice: 4
Enter data to insert at end: 20
Inserted 20 at the end.

1. Create a doubly linked list
2. Insert at beginning
3. Insert at location
4. Insert at end
5. Display
6. Exit
Enter your choice: 3
Enter data to insert: 15
Enter location to insert at: 2
Inserted 15 at location 2.

1. Create a doubly linked list
2. Insert at beginning
3. Insert at location
4. Insert at end
5. Display
6. Exit
Enter your choice: 5
10 <-> 15 <-> 20 <-> NULL
```

PROGRAM 8

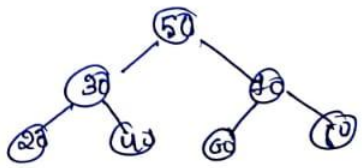
8. Write a program

a) To construct a binary Search tree.

b) To traverse the tree using all the methods i.e., in-order, preorder and post order, display all traversal order

```
⑧ Write a programs
a) To construct a binary search tree
b) to traverse the tree using all the methods
   i.e inorder, preorder, post order display all
   traversal order

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

Struct Node {
      int key;
      Struct Node * left;
      struct Node * right;
};

Struct Node* newNode (int key) {
      Struct Node* = (Struct Node* )malloc (sizeof(
                                  struct Node));
      node -> key = key;
      node -> left = node -> right = NULL;
      return node;
}

struct Node* insert (struct Node* node, int key){
      if (node == NULL) return newNode(key);
      if (key < node -> key) {
         node -> left = insert (node -> left, key);
      else if (key > node-> key) {
         node -> right = insert (node->right, key);
      }
      return node;
}
```

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

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

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

void main() {
    struct Node * root = NULL;
    root = insert(root, 50);
    root = insert(root, 30);
    root = insert(root, 20);
    root = insert(root, 40);
    root = insert(root, 70);
    root = insert(root, 60);
    root = insert(root, 80);
```



```
printf ("In Orde
inorder (root);
printf ("\n");
printf (" Pre order traversal: ");
preorder (root);
printf ("\n");
printf ("Post order traversal: ");
postorder(root);
printf ("\n");
}
```

In order traversal: 20 30 40 50 60 70 80

Pre order traversal: 50 30 20 40 70 60 80

Post order traversal: 20 40 30 60 80 70 50

Code

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

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

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

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

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

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

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

int main() {
    Node* root = NULL;
    root = insert(root, 50);
    insert(root, 30);
    insert(root, 70);
    insert(root, 20);
    insert(root, 40);
    insert(root, 60);
    insert(root, 80);

    printf("Inorder traversal: ");
    inorderTraversal(root);
    printf("\n");

    printf("Preorder traversal: ");
    preorderTraversal(root);
    printf("\n");

    printf("Postorder traversal: ");
    postorderTraversal(root);
    printf("\n");

    return 0;
}
```

Output

Inorder traversal: 20 30 40 50 60 70 80
Preorder traversal: 50 30 20 40 70 60 80
Postorder traversal: 20 40 30 60 80 70 50

PROGRAM 9

9a Write a programme to traverse a graph by bfs
9b write a programme to traverse a graph by dfs

9a) Write a program to traverse a graph using BFS Method.

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

#define MAX_VERTICES 10

struct queue {
    int items[MAX_VERTICES];
    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_VERTICES - 1) {
        printf ("queue is full!\n");
        return;
    }
    if (q->front == -1) {
        q->front = 0;
    }
    q->rear++;
    q->items[q->rear] = value;
}
```

```
void BFS (struct Graph* g, int startVertex) {
    struct Queue q;
    initQueue (&q);
    int visited [MAX_VERTICES] = {0};
    visited [startVertex] = 1;
    enqueue (&q, startVertex);
    printf ("BFS traversal starting from vertex %d
                startVertex);

    While (!isEmpty (&q)) {
        int currentVertex = dequeue (&q);
        printf ("%d ", currentVertex);
        for (int i=0; i < g -> vertices; i++) {
        if (g -> adjMatrix [currentVertex] [i] == 1 &&
                !visited [i]) {
            enqueue (&q, i);
            visited [i] = 1;
        }
    }
    printf ("\n");
}
int main () {
    struct Graph g;
    initGraph (&g, 5);
    addEdge (&g, 0, 1);
    addEdge (&g, 0, 2);
    addEdge (&g, 1, 3);
    addEdge (&g, 1, 4);
    BFS (&g, 0);
    return 0;
}
```

4 b) Write a program to traverse a graph using DFS method.

```
#include <stdio.h>
#define MAX_VERTICES 100
int graph[MAX_VERTICES][MAX_VERTICES];
int visited[MAX_VERTICES];

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

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

int main() {
    int n, e, x, y;
    printf("Enter the number of vertices: ");
    scanf("%d", &n);
    for(int i=0; i<n; i++) {
        for(int j = 0; j<n; j++) {
            graph[i][j]=0;
        }
        visited[i]=0;
    }
    printf("Enter the number of edges: ");
    scanf("%d", &e);
```

```
for(int i=0 ; i <e; i++){
        printf ("Enter edge (x y): ");
        scanf("%d %d", &x, &y);
        graph[x][y] = 1;
        graph[y][x] = 1;
}
printf("DFS Traversal starting from vertex 0: \n");
DFS (0, N);
return 0;
}
```

Enter the number of vertices: 5
Enter the number of edges: 4
Enter edge (x, y): 0 1
Enter edge (x, y): 0 2
Enter edge (x, y): 1 3
Enter edge (x y): 2 4

DFS Traversal starting from vertex 0: 0 1 3 2 4

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

#define MAX 100

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

void enqueue(int value) {
    if (rear == MAX - 1) {
        return;
    }
    if (front == -1) {
        front = 0;
    }
    queue[++rear] = value;
}

int dequeue() {
    if (front == -1 || front > rear) {
        return -1;
    }
    return queue[front++];
}

int isQueueEmpty() {
    return front == -1 || front > rear;
}

void bfs(int graph[MAX][MAX], int n, int start) {
    for (int i = 0; i < n; i++) {
        visited[i] = 0;
    }
    enqueue(start);
    visited[start] = 1;
    while (!isQueueEmpty()) {
        int current = dequeue();
        printf("%d ", current);
```

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

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

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

    printf("Enter the adjacency matrix:\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: ");
    scanf("%d", &start);

    printf("BFS Traversal: ");
    bfs(graph, n, start);

    return 0;
}
```

```
Enter the number of vertices: 5
Enter the adjacency matrix:
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
BFS Traversal: 0 1 2 3 4
```

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

#define MAX 100

int visited[MAX];

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

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

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

    printf("Enter the adjacency matrix:\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: ");
    scanf("%d", &start);

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

    printf("DFS Traversal: ");
    dfs(graph, n, start);

    return 0;
}
```

```
Enter the number of vertices: 5
Enter the adjacency matrix:
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
DFS Traversal: 0 1 2 3 4
```