VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

BISWAJEET BEHERA (1BM23CS069)

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

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

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

This is to certify that the Lab work entitled **“DATA STRUCTURES”** carried out by **BISWAJEET BEHERA(1BM23CS069)**, who is Bonafide student of **B. M. S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and**
Engineering of the Visvesvaraya Technological University, Belgaum during the year 2024-25. The Lab report has been approved as it satisfies the academic requirements in respect of Data structures Lab - **(23CS3PCDST)** work prescribed for the said degree.

Prof.SELVA
KUMAR S
Assistant Professor
Department of CSE
BMSCE, Bengaluru

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

Index Sheet

Course outcomes:

CO1	Apply the concept of linear and nonlinear data structures.
CO2	Analyze data structure operations for a given problem
CO3	Design and develop solutions using the operations of linear and nonlinear data structure for a given specification.
CO4	Conduct practical experiments for demonstrating the operations of different data structures.

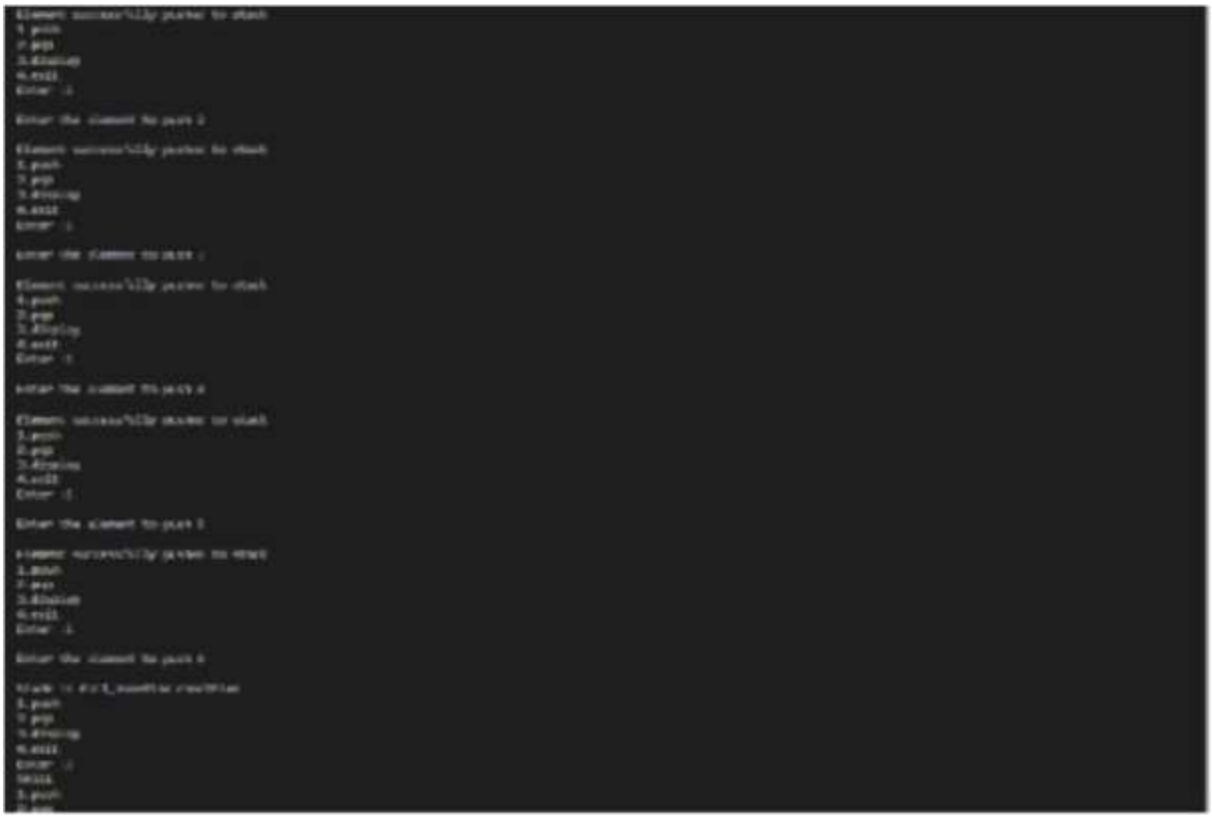
Lab program 1:

Write a program to simulate the working of stack using an array with the following: a) Push b) Pop c) Display The program should print appropriate messages for stack overflow, stack underflow

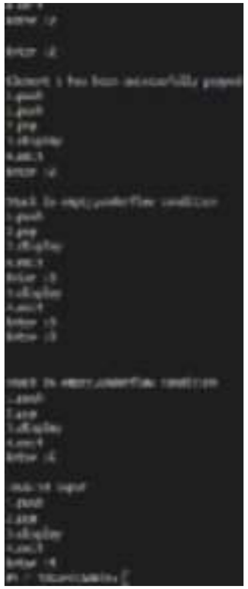
```
#include<stdio.h>

#include<stdlib.h>

#define SIZE 5

int stack[SIZE];

int top=-1;


        void push(int a)

        {

        if(top==SIZE-1)

        {

        printf("\nStack is full,overflow condition");

        }

        else

        {

        top++;

        stack[top]=a;

        printf("\nElement successfully pushed to stack");
        }

        }

        void pop()

        {


          if(top==-1)

          {

          printf("\nStack is empty,underflow condition");

          }

          else

          {
```

```
int ele = stack[top];

printf("\nElement %d has been successfully popped",ele); top--

;
}

}

void display()

{

if(top==-1)

{

printf("\nstack is empty,underflow condition");

}

else

{

for(int i=top;i>-1;i—)

{

printf("%d ",stack[i]);

}

}

}
void main()

{

int c,e; while(1)
{

printf("\n\n1.push\n2.pop\n3.display\n4.exit\nEnter :");

scanf("%d",&c);

switch (c)
{

case 1: printf("\nEnter the element to push "); scanf("%d",&e);
push(e);

break;

case 2: pop(); break;

case 3: display(); break;

case 4: exit(1);

default : printf("\nInvalid

input");

}

}

}
```

```
Enter the element to push [illegible]

Stack is full_overflow condition
1.push
2.pop
3.display
4.exit
Enter :2

Element [illegible] has been successfully popped
1.push
2.pop
3.display
4.exit
Enter :2

Element [illegible] has been successfully popped
1.push
2.pop
3.display
4.exit
Enter :2

Element [illegible] has been successfully popped
1.push
2.pop
3.display
4.exit
Enter :2

Element [illegible] has been successfully popped
1.push
2.pop
3.display
4.exit
Enter :2

Element [illegible] has been successfully popped
1.push
2.pop
3.display
4.exit
Enter :2

Stack is empty_underflow condition
1.push
2.pop
3.display
4.exit
Enter :1

Enter the element to push [illegible]
```



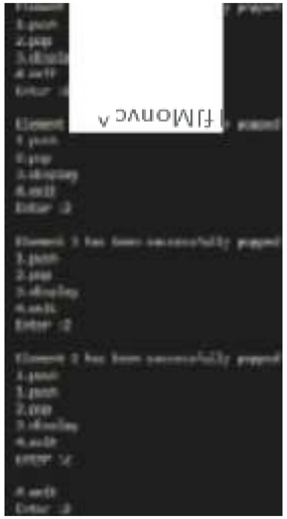
```
Element successfully pushed to stack
1.push
2.pop
3.display
4.exit
Enter :1

Enter the element to push [illegible]

Element successfully pushed to stack
1.push
2.pop
3.display
4.exit
Enter :1

Enter the element to push [illegible]

Element successfully pushed to stack
1.push
2.pop
3.display
4.exit
Enter :1

Enter the element to push [illegible]

Element successfully pushed to stack
1.push
2.pop
3.display
4.exit
Enter :1

Enter the element to push [illegible]

Element successfully pushed to stack
1.push
2.pop
3.display
4.exit
Enter :1

Enter the element to push [illegible]

Stack is full_overflow condition
1.push
2.pop
3.display
4.exit
Enter :[illegible]
[illegible]
1.push
2.pop
```



```
[illegible]
Enter :2

Enter :2

Element 1 has been successfully popped
1.push
1.push
2.pop
3.display
4.exit
Enter :2

Stack is empty,underflow condition
1.push
2.pop
3.display
4.exit
Enter :3
3.display
4.exit
Enter :5
Enter :3


Stack is empty,underflow condition
1.push
2.pop
3.display
4.exit
Enter :6

Invalid input
1.push
2.pop
3.display
4.exit
Enter :4
PS ~ [illegible]
```

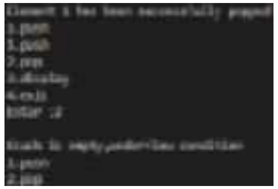
00

TD QJ cr q (D



```
Element        popped
1.push
2.pop
3.displ
4.exit
Enter :

Element        popped
1.push
2.pop
3.display
4.exit
Enter :2

Element 3 has been successfully popped
1.push
2.pop
3.display
4.exit
Enter :2

Element 2 has been successfully popped
1.push
1.push
2.pop
3.display
4.exit
Enter :2

4.exit
Enter :2
```

fJMonvc ^



```
Element 1 has been successfully popped
1.push
1.push
2.pop
3.display
4.exit
Enter :2

Stack is empty,underflow condition
1.push
2.pop
```

Lab program 2

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

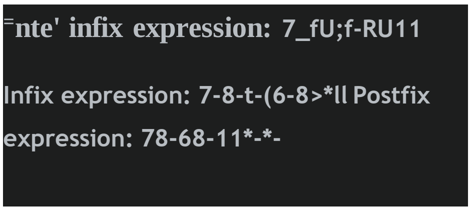
```
#include <stdio.h>
#include <string.h>

int index1 = 0, pos = 0, top = -1, length;
char symbol, temp, infix[20], postfix[20], stack[20];
void infixToPostfix();
void push(char symbol);
char pop();
int pred(char symbol); int main() {
printf("Enter infix expression:\n"); scanf("%s", infix);
infixToPostfix();
printf("\nInfix expression: %s", infix); printf("\nPostfix expression:
%s\n", postfix); return 0;
}
void infixToPostfix() {
length = strlen(infix);
push('#'); // Push an initial dummy character to the stack while
(index1 < length) { symbol = infix[index1]; switch (symbol) { case '(':
push(symbol); break; case ')':
temp = pop();
while (temp != '(') {
postfix[pos++] = temp; temp = pop();
}
break;
```

case	'+
case	'-
case	'*
case	'/
case	

```
while (pred(stack[top]) >= pred(symbol)) { temp
     = pop(); postfix[pos++] = temp;
}
push(symbol);
break;
default:
postfix[pos++] = symbol;
}
index1++;
}
while (top > 0) { temp = pop(); postfix[pos++]
     = temp;
}
postfix[pos] = '\0';
```

```
}

void push(char symbol) {
     top = top + 1;
     stack[top] = symbol;
}

char pop() { char symb;
     symb = stack[top]; top
     = top - 1; return
     symb;
}

int pred(char symbol) { int
     p;
switch (symbol) { case 'A':
     p = 3; break; case
     '*': case '/': p = 2;
     break; case '+': case
     '-': p = 1; break;
     case '(': p = 0;
     break; case '#': p = -
     1; break; default:
p = -1;
}
return p;
}
```

Output:



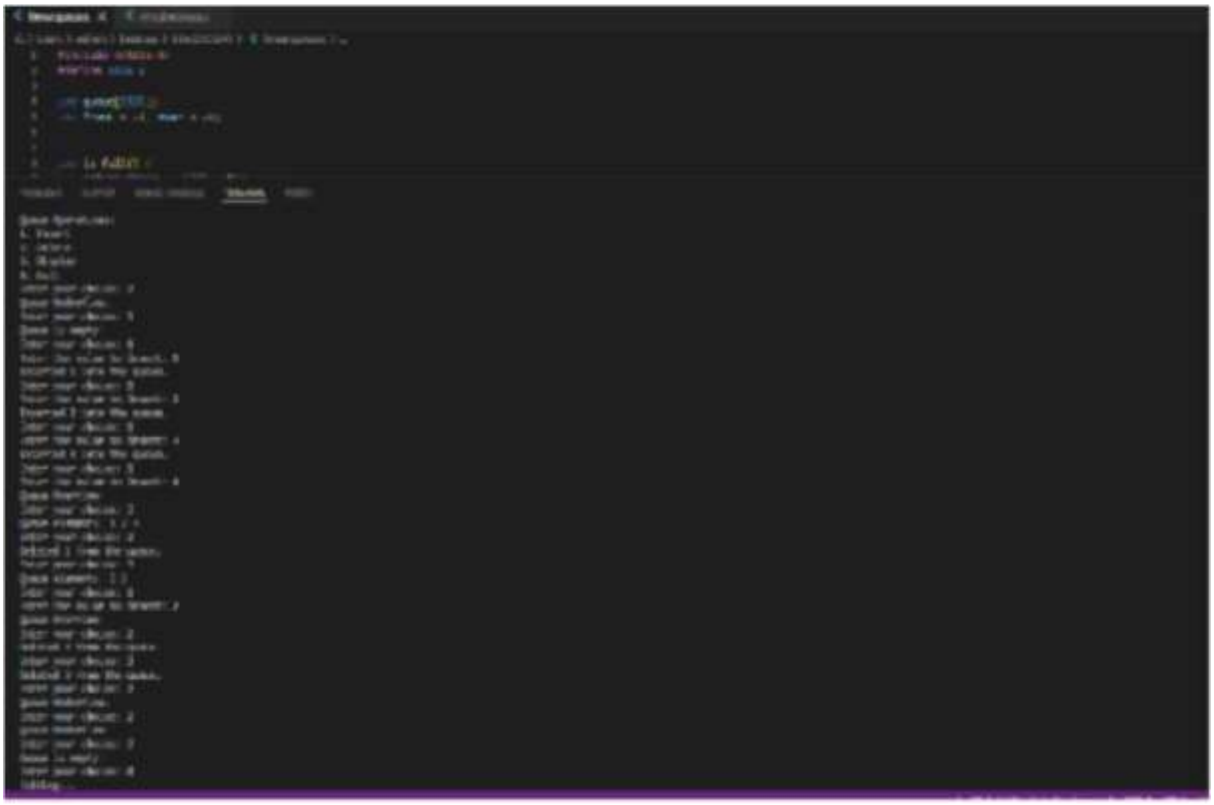
```
=nte' infix expression: 7_fU;f-RU11

Infix expression: 7-8-t-(6-8>*ll Postfix
expression: 78-68-11*-*-
```

Lab Program 3

a) 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 SIZE 3



int queue[SIZE];

int front = -1, rear = -1;




int is_fuN() {

return (rear == SIZE - 1);

}





int is_empty() { return

    (front == -1);
}





void insert(int value) { if (is_fuM()) {

prinK("Queue Overflow\n"); return;
}

                      if (front == -1) front = 0;

queue[++rear] = value;

prinK("Inserted %d into the queue.\n", value);
}

void delete() { if (is_empty()) { printff'Queue Underflow.\n");

return;
}

printff'Deleted %d from the queue.\n", queue[front]);

front++; if (front > rear) { front = -1; rear = -1;
}

}

void display() { if (is_empty()) { printff'Queue is empty!\n");

return;
}

printff'Queue elements: "); for (int i = front; i <= rear; i++) {

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

printf("\n");

}

int main() {
```

```
int choice, value; printf("\nQueue Operations:\n");
printf("1. Insert\n"); printf("2. Delete\n"); printf("3.
Display\n"); printf("4. Exit\n"); while (1) {
printff'Enter your choice: "); scanf("%d", &choice);

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



```
#include <stdio.h>
#define SIZE 3

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


int isFull() {
```

```
Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Queue Underflow.
Enter your choice: 3
Queue is empty
Enter your choice: 1
Enter the value to insert: 1
Inserted 1 into the queue.
Enter your choice: 1
Enter the value to insert: 2
Inserted 2 into the queue.
Enter your choice: 1
Enter the value to insert: 4
Inserted 4 into the queue.
Enter your choice: 1
Enter the value to insert: 6
Queue Overflow
Enter your choice: 3
Queue elements: 1 2 4
Enter your choice: 2
Deleted 1 from the queue.
Enter your choice: 3
Queue elements: 2 4
Enter your choice: 1
Enter the value to insert: 7
Queue Overflow
Enter your choice: 2
Deleted 2 from the queue.
Enter your choice: 2
Deleted 4 from the queue.
Enter your choice: 2
Queue Underflow.
Enter your choice: 2
Queue Underflow
Enter your choice: 3
Queue is empty
Enter your choice: 4
Exiting...
```

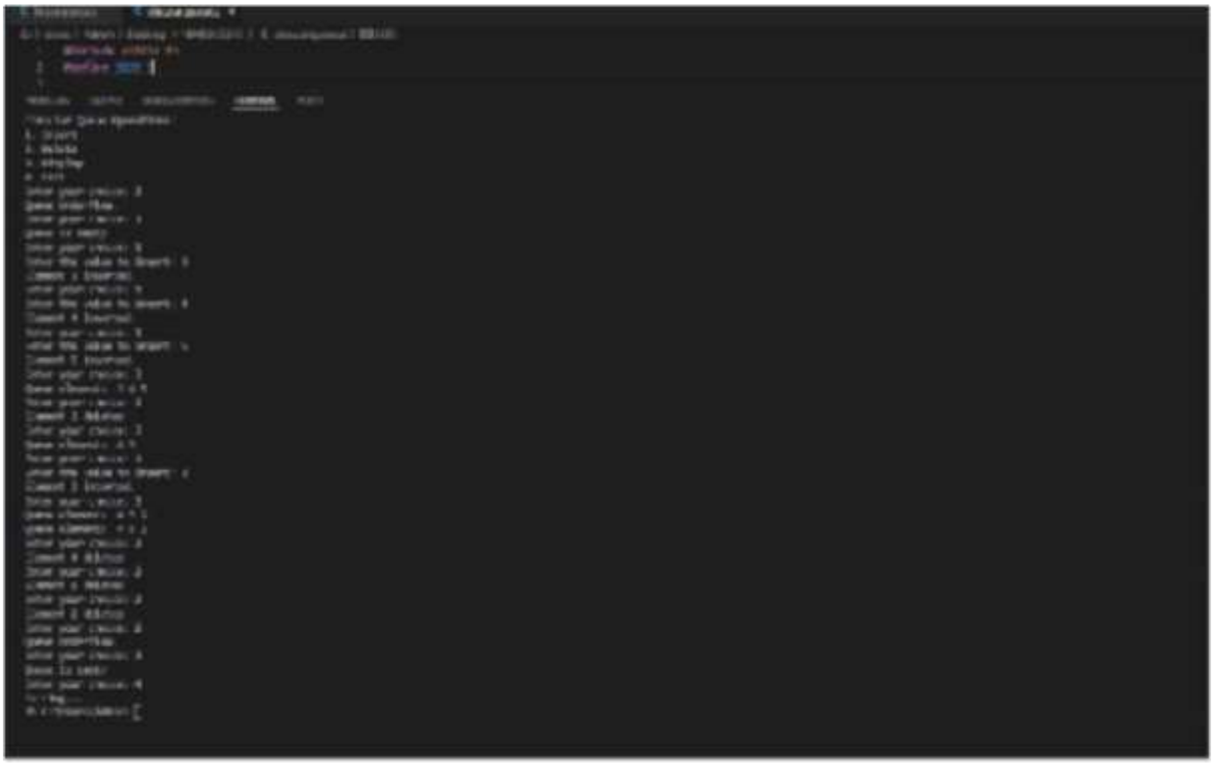
b) WAP to simulate the working of a circular queue of integers using an array. Provide the following operations: Insert, Delete & Display The program should print appropriate messages for queue empty and queue overflow conditions

```
#include <stdio.h>

#define SIZE 3


int queue[SIZE];

int front = -1, rear = -1;




int is_fuN() {

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

}




int is_empty() { return

    (front == -1);
}




   void insert(int n) { if (is_fuM()) {

prinK("Queue Overflow\n"); return;
}

if (is_empty()) front = 0; rear = 0;

else

rear = (rear + 1) % SIZE;
```

```
    queue[rear] = n;

    printff'Element %d inserted.\n", n);

    }




void delete() { if (is_empty()) { printff'Queue

Underflow.\n"); return;
}

printff'Element %d deleted\n", queue[front]); if

(front == rear){ front = -1; rear = -1;
}

else

front = (front + 1) % SIZE;

}




void display() { if (is_empty()) {

printff'Queue is empty\n"); return;
}

printff'Queue elements: "); int i =

front; while (1) {

printf("%d ", queue[i]); if (i == rear)
```

```
break;
i = (i + 1) % SIZE;
}
printf("\n");
}

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

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

```
            printf("Exiting...\n"); return 0; default:

            printff'Invalid choice! Please try again.\n");




        }

    }

}
```



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

```
Circular Queue Operations:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Queue Underflow
Enter your choice: 3
Queue is empty
Enter your choice: 1
Enter the value to insert: 3
Element 3 inserted
Enter your choice: 1
Enter the value to insert: 4
Element 4 inserted
Enter your choice: 1
Enter the value to insert: 5
Element 5 inserted
Enter your choice: 3
Queue elements: 3 4 5
Enter your choice: 2
Element 3 deleted
Enter your choice: 3
Queue elements: 4 5
Enter your choice: 1
Enter the value to insert: 2
Element 2 inserted
Enter your choice: 3
Queue elements: 4 5 2
Queue elements: 4 5 2
Enter your choice: 2
Element 4 deleted
Enter your choice: 2
Element 5 deleted
Enter your choice: 2
Element 2 deleted
Enter your choice: 2
Queue Underflow
Enter your choice: 3
Queue is empty
Enter your choice: 4
Exiting...
PS C:\Users\Admin> 
```

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.

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

Display the contents of the linked list.

```
#include <stdio.h>

#include <stdlib.h>


                                        struct Node { int data;

struct Node* next;
};




// Create a new node

struct Node* create_node(int data) { struct Node* new_node = (struct

Node*)malloc(sizeof(struct Node)); new_node->data = data; new_node->next

= NULL; return new_node;
}


// Insert node at the beginning

void insert_at_beginning(struct Node** head, int data) { struct Node*

new_node = create_node(data); new_node->next = *head;

*head = new_node;
}
```

```
// Insert node at the end

void insert_at_end(struct Node** head, int data) { struct Node*

new_node = create_node(data); if (*head == NULL) {

*head = new_node; return;
}

struct Node* temp = *head; while (temp->next != NULL) { temp = temp-

>next;
}

temp->next = new_node;

}


// Insert node at a specific position

void insert_at_position(stmct Node** head, int data, int position) { if

(position < 0) return; // Invalid position if (position == 0) {

insert_at_beginning(head, data); return;
}

struct Node* new_node = create_node(data);

struct Node* temp = *head;

for (int i = 0; i < position - 1; i++) {

if (temp == NULL) return; // Position out of range temp = temp->next;
}

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

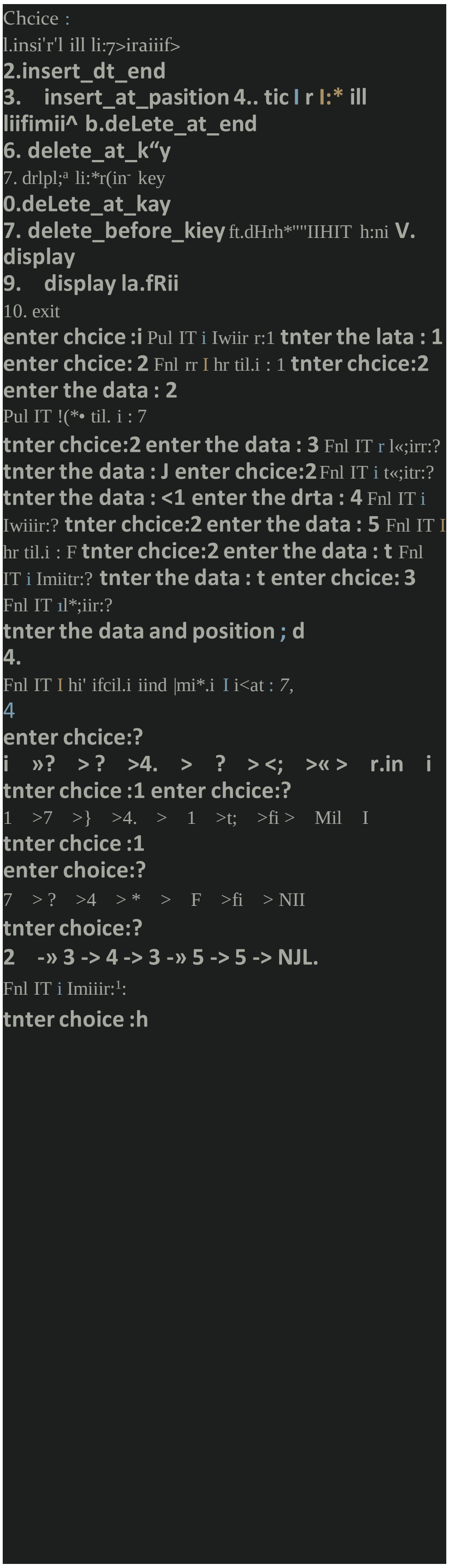
```
// Delete node at the beginning void

delete_at_beginning(struct Node** head) { if (*head !=

NULL) { struct Node* temp = *head;

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

}


// Delete node at the end void delete_at_end(struct

Node** head) { if (*head == NULL) return; if ((*head)->next

== NULL) { free(*head);

*head = NULL; return;
}

struct Node* temp = *head;

while (temp->next && temp->next->next != NULL) { temp =

temp->next;
}

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


// Delete node with a specific key void

delete_at_key(struct Node** head, int key) { if (*head ==

NULL) return; if ((*head)->data == key) {
  struct Node* temp = *head;

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

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

                    if (temp->next == NULL) return;
struct Node* to_delete = temp->next;
                   temp->next = temp->next->next;
free(to_delete);
}


// Delete node before the key
void delete_before_key(struct Node** head, int key) {
if (*head == NULL | | (*head)->next == NULL) return;
  if ((*head)->next->data == key) { struct Node* temp = *head;
*head = (*head)->next;
free(temp);
return;
}

struct Node* temp = *head;
while (temp->next != NULL && temp->next->next != NULL) { if
(temp->next->next->data == key) { struct Node* to_delete =
temp; temp = temp->next; free(to_delete);
```

```
  return;

}

temp = temp->next;

}

}


// Delete node after the key

void delete_after_key(struct Node** head, int key) {

struct Node* temp = *head; while (temp != NULL

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

if (temp != NULL && temp->next != NULL) { struct

Node* to_delete = temp->next; temp->next = temp-

>next->next; free(to_delete);
}

}


// Display the list void display(struct Node* head) {

struct Node* temp = head; while (temp != NULL) {

prinK("%d -> ", temp->data); temp = temp->next;
}

prinK("NULL\n");

}

// Free all nodes to avoid memory leaks
```

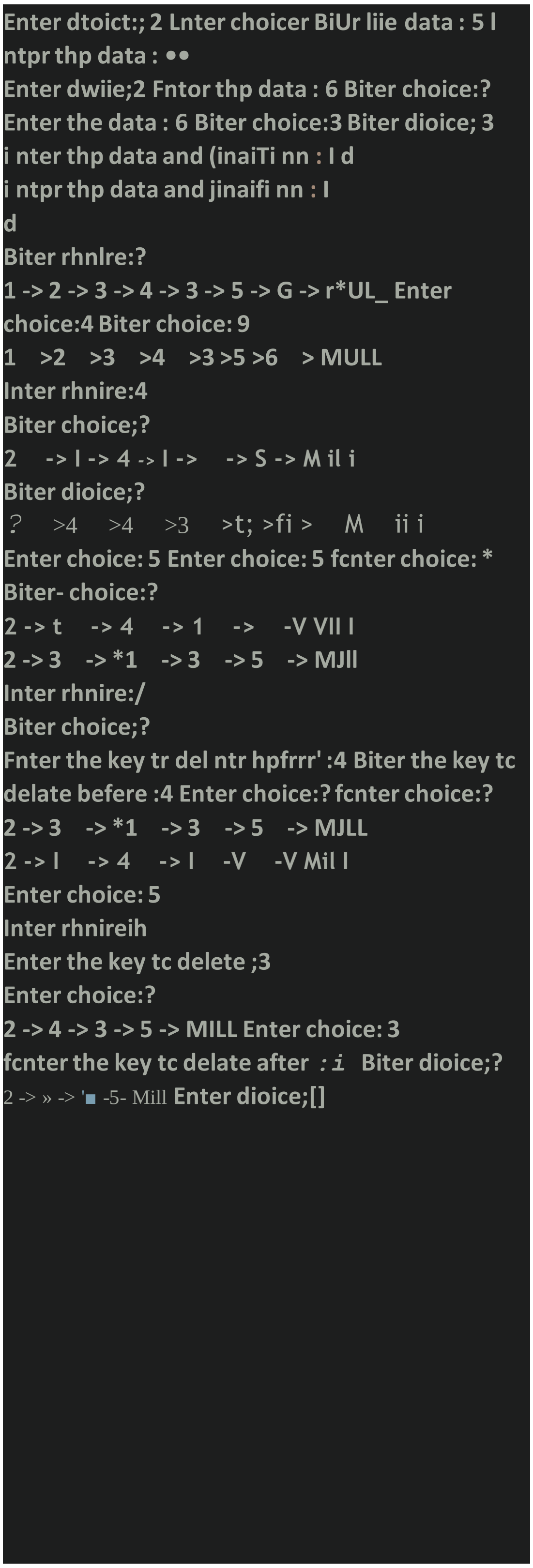
```
void free_list(struct Node** head) { struct Node* temp; while (*head != NULL) { temp = *head;

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

}


int main() {

struct Node* head = NULL; int data, key;

prinK("Choice :
\n1.msert_at_begmning\n2.insert_at_end\n3.msert_at_posiQon\n4.delete_at_begmning\n5.delete
_at_end\n6.delete_at_key\n7.delete_before_key\n8.delete_aner_key\n9.display\n10.exit\n");


int c;

while (1) {

prinK("Enter choice: "); scanf("%d", &c); switch (c) { case 1:

prinK("Enter the data: "); scanf("%d", &data); insert_at_beginning(&head, data); break; case 2:

prinK("Enter the data: "); scanf("%d", &data); insert_at_end(&head, data);
```

```
    break;

case 3:

printff'Enter the data and position: ");

scanf("%d%d", &data, &key);

insert_at_position(&head, data, key); break;

case 4:

delete_at_beginning(&head); break; case 5:

delete_at_end(&head); break; case 6:

printff'Enter the key to delete: "); scanf("%d",

&key); delete_at_key(&head, key); break; case

7:

printff'Enter the key to delete before: ");

scanf("%d", &key); delete_before_key(&head,

key); break; case 8:

printff'Enter the key to delete after: ");

scanf("%d", &key); delete_after_key(&head,

key); break; case 9:

display(head);

break;
```

```
    case 10:

        exit(0);

        default:

        printff'Invalid choice...\n");
  }

  }

  return
  0;
}
```



```
Chcice :
l.insi'r'l ill li:7>iraiiif>
2.insert_dt_end
3.    insert_at_pasition 4.. tic l r l:* ill
liifimii^ b.deLete_at_end
6. delete_at_k“y
7. drlpl;a li:*r(in- key
0.deLete_at_kay
7. delete_before_kiey ft.dHrh*""IIHIT h:ni V.
display
9.    display la.fRii
10. exit
enter chcice :i Pul IT i Iwiir r:1 tnter the lata : 1
enter chcice: 2 Fnl rr I hr til.i : 1 tnter chcice:2
enter the data : 2
Pul IT !(*• til. i : 7
tnter chcice:2 enter the data : 3 Fnl IT r l«;irr:?
tnter the data : J enter chcice:2 Fnl IT i t«;itr:?
tnter the data : <1 enter the drta : 4 Fnl IT i
Iwiiir:? tnter chcice:2 enter the data : 5 Fnl IT I
hr til.i : F tnter chcice:2 enter the data : t Fnl
IT i Imiitr:? tnter the data : t enter chcice: 3
Fnl IT ıl*;iir:?
tnter the data and position ; d
4.
Fnl IT I hi' ifcil.i iind |mi*.i I i<at : 7,
4
enter chcice:?
i    »?    > ?    >4.    >    ?    > <;    >« >    r.in    i
tnter chcice :1 enter chcice:?
1    >7    >}    >4.    >    1    >t;    >fi >    Mil    I
tnter chcice :1
enter choice:?
7    > ?    >4    > *    >    F    >fi    > NII
tnter choice:?
2    -» 3 -> 4 -> 3 -» 5 -> 5 -> NJL.
Fnl IT i Imiiir:1:
tnter choice :h
```



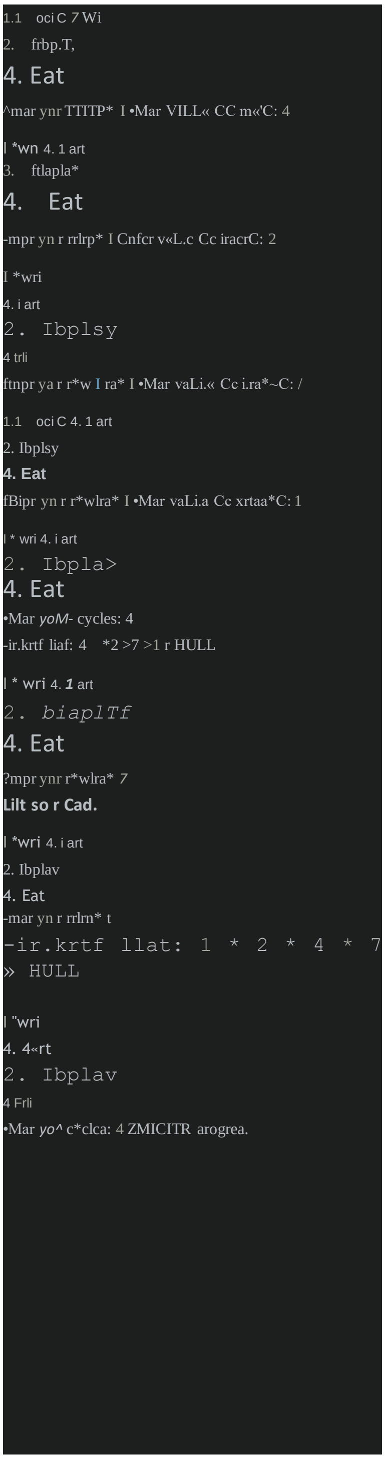
```
Enter dtoict:; 2 Lnter choicer BiUr liie data : 5 l
ntpr thp data : ••
Enter dwiie;2 Fntor thp data : 6 Biter choice:?
Enter the data : 6 Biter choice:3 Biter dioice; 3
i nter thp data and (inaiTi nn : I d
i ntpr thp data and jinaifi nn : I
d
Biter rhnlre:?
1 -> 2 -> 3 -> 4 -> 3 -> 5 -> G -> r*UL_ Enter
choice:4 Biter choice: 9
1    >2    >3    >4    >3 >5 >6    > MULL
Inter rhnire:4
Biter choice;?
2     -> l -> 4 -> l ->     -> S -> M il i
Biter dioice;?
 ?    >4    >4    >3    >t; >fi >    M    ii i
Enter choice: 5 Enter choice: 5 fcnter choice: *
Biter- choice:?
2 -> t    -> 4    -> 1    ->    -V VII I
2 -> 3    -> *1    -> 3    -> 5    -> MJll
Inter rhnire:/
Biter choice;?
Fnter the key tr del ntr hpfrrr' :4 Biter the key tc
delate befere :4 Enter choice:? fcnter choice:?
2 -> 3    -> *1    -> 3    -> 5    -> MJLL
2 -> l    -> 4    -> l    -V    -V Mil I
Enter choice: 5
Inter rhnireih
Enter the key tc delete ;3
Enter choice:?
2 -> 4 -> 3 -> 5 -> MILL Enter choice: 3
fcnter the key tc delate after  : i   Biter dioice;?
2 -> » -> '■ -5- Mill Enter dioice;[]
```

Lab Program 5

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* insertAtEnd(struct Node* head, int value) { struct Node*

  newNode = (struct Node*)malloc(sizeof(struct Node)); newNode->data =

  value; newNode->next = NULL; if (!head) return newNode;


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

  >next = newNode; return head;
}


void printList(struct Node* head) { while (head) {

prinK("%d -> ", head->data); head = head->next;
}

prinK("NULL\n");

}
```

```
struct Node* sortList(struct Node* head) { if (!head | | !head->next) return head;

struct Node* current = head; while (current) {

struct Node* next = current->next; while (next) {

if (current->data > next->data) { int temp = current->data; current->data = next-

>data; next->data = temp;
}

next = next->next;

}

current = current->next;

}

return head;

}


int main() {

struct Node* head = NULL; int choice, value;

do {

prinK("\n1. Insert\n2. Sort\n3. Display\n4. Exit\nEnter your choice: "); scanf("%d",

&choice);

switch (choice) { case 1:
    printff'Enter value to insert: ");

scanf("%d", &value); head =

insertAtEnd(head, value); break; case 2:

    head = sortList(head); printf("List

sorted.\n"); break; case 3:

    prntf("Linked list: "); printList(head);

break; case 4:

    printff'ExiQng program.\n"); break;

default:

    printf("Invalid choice. Try again.\n");
    }


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

    }
```

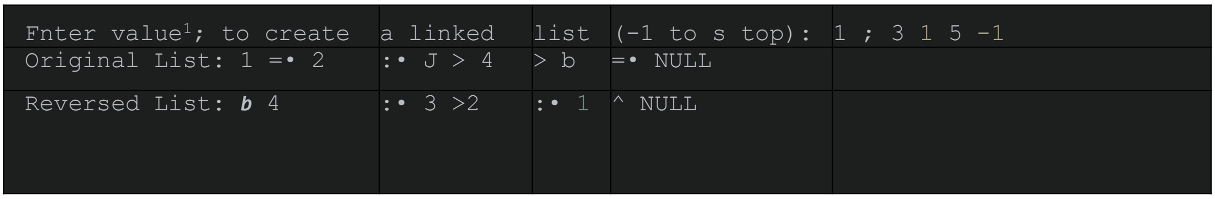


```
1.1   oci C 7 Wi
2.    frbp.T,
4. Eat
^mar ynr TTITP*  I •Mar  VILL« CC m«'C: 4

I *wn 4. 1 art
3.    ftlapla*
4.   Eat
-mpr yn r rrlrp* I Cnfcr v«L.c Cc iracrC: 2

I *wri
4. i art
2. Ibplsy
4 trli
ftnpr ya r r*w I ra* I •Mar vaLi.« Cc i.ra*~C: /

1.1   oci C 4. 1 art
2. Ibplsy
4. Eat
fBipr yn r r*wlra* I •Mar vaLi.a Cc xrtaa*C: 1

I * wri 4. i art
2. Ibpla>
4. Eat
•Mar yoM- cycles: 4
-ir.krtf liaf: 4   *2 >7 >1 r HULL

I * wri 4. 1 art
2. biaplTf
4. Eat
?mpr ynr r*wlra* 7
Lilt so r Cad.

I *wri 4. i art
2. Ibplav
4. Eat
-mar yn r rrlrn* t
-ir.krtf llat: 1 * 2 * 4 * 7
» HULL

I "wri
4. 4«rt
2. Ibplav
4 Frli
•Mar yo^ c*clca: 4 ZMICITR arogrea.
```

```
#include <stdio.h>

#include <stdlib.h>


 struct Node { int data;

struct Node* next;
};




struct Node* insertAtEnd(struct Node* head, int value) { struct Node*

    newNode = (struct Node*)malloc(sizeof(struct Node)); newNode->data =

    value; newNode->next = NULL; if (!head) return newNode;


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

    >next = newNode; return head;
}


void printList(struct Node* head) { while (head) {

prinK("%d -> ", head->data); head = head->next;
}

prinK("NULL\n");

}
```

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

while (current) {
next = current->next; current->next = prev; prev = current;
    current = next;
}
return prev;
}

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

printff'Enter values to create a linked list (-1 to stop): "); do {
scanf("%d", &value);
if (value != -1) head = insertAtEnd(head, value);
} while (value != -1);

printff'Original List: "); printList(head);
head = reverseList(head);

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

```
    printList(head);



    return 0;

}
```

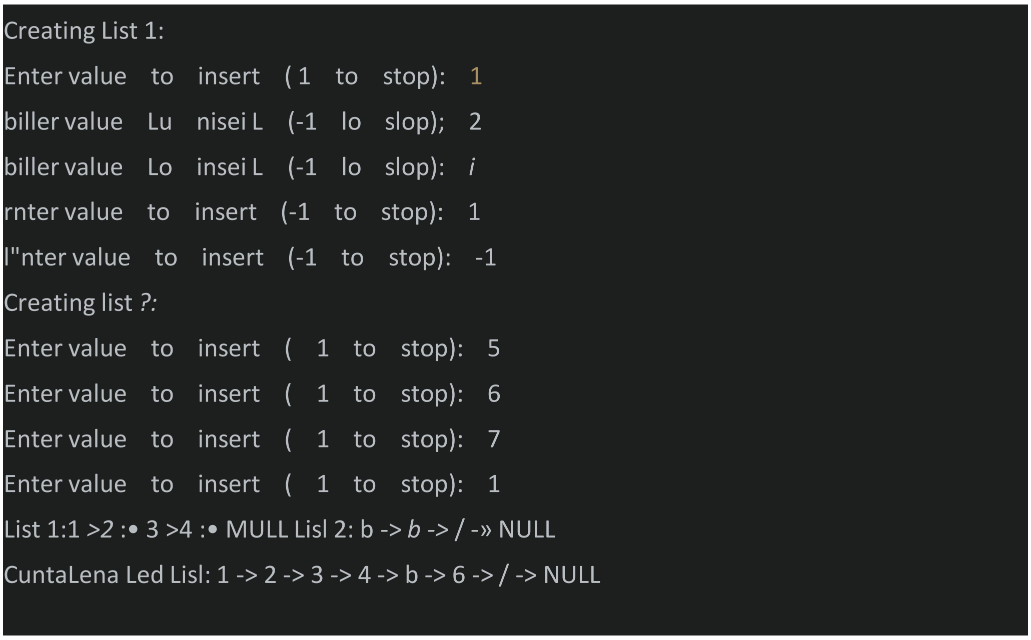


```
Fnter value1; to create a linked list (-1 to s top): 1 ; 3 1 5 -1
Original List: 1 =• 2 :• J > 4 > b =• NULL

Reversed List: b 4 :• 3 >2 :• 1 ^ NULL
```

```
#include <stdio.h>

#include <stdlib.h>


 struct Node { int data;

struct Node* next;
};




struct Node* insertAtEnd(struct Node* head, int value) { struct Node*

    newNode = (struct Node*)malloc(sizeof(struct Node)); newNode->data =

    value; newNode->next = NULL; if (!head) return newNode;


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

    >next = newNode; return head;
}


void printList(struct Node* head) { while (head) {

prinK("%d -> ", head->data); head = head->next;
}

prinK("NULL\n");

}
```

```
struct Node* concatenateLists(struct Node* headl, struct Node* head2) {

    if (!head1) return head2; if (!head2) return headl;


    struct Node* temp = headl; while (temp->next)

    temp = temp->next; temp->next = head2; return

    head1;
    }


    int main() {

    struct Node* list1 = NULL; struct Node* list2 =

    NULL;

    int choice, value;


    printff'CreaQng List 1:\n"); do {

    printff'Enter value to insert (-1 to stop): ");

    scanf("%d", &value);

    if (value != -1) l ist1 = insertAtEnd(list1, value); }

    while (value != -1);


    printff'CreaQng List 2:\n"); do {

    printff'Enter value to insert (-1 to stop): ");

    scanf("%d", &value);

    if (value != -1) list2 = insertAtEnd(list2, value); }

    while (value != -1);
```

```
printff'List 1: ");

printList(listl);


printff'List 2: ");

printList(list2);

listl = concatenateLists(list1, list2);


printff'Concatenated List: ");

printList(listl);

 return 0;

 }
```



```
Creating List 1:
Enter value    to    insert    ( 1    to    stop):    1
biller value    Lu    nisei L    (-1    lo    slop);    2
biller value    Lo    insei L    (-1    lo    slop):    i
rnter value    to    insert    (-1    to    stop):    1
l"nter value    to    insert    (-1    to    stop):    -1
Creating list ?:
Enter value    to    insert    (    1    to    stop):    5
Enter value    to    insert    (    1    to    stop):    6
Enter value    to    insert    (    1    to    stop):    7
Enter value    to    insert    (    1    to    stop):    1
List 1:1 >2 :• 3 >4 :• MULL Lisl 2: b -> b -> / -» NULL
CuntaLena Led Lisl: 1 -> 2 -> 3 -> 4 -> b -> 6 -> / -> NULL
```

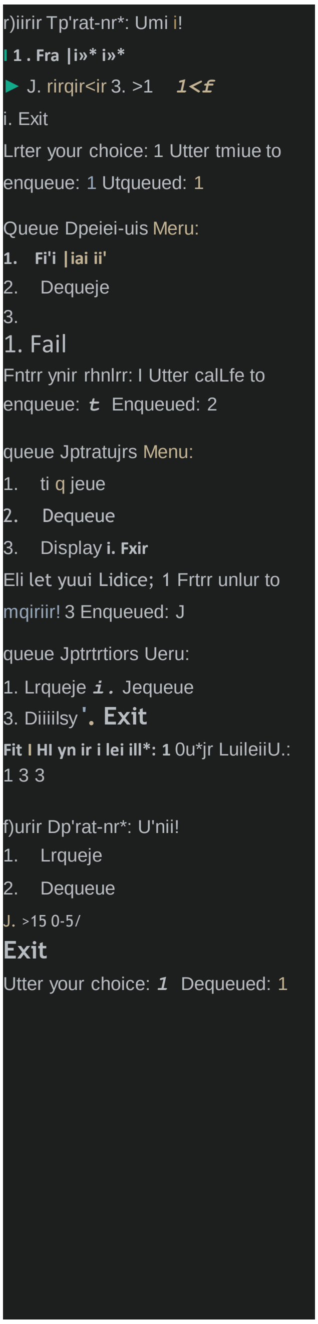
WAP to Implement Single Link List to simulate Stack & Queue Operations.

```
#include <stdio.h>

#include <stdlib.h>

 struct Node { int data;
struct Node* next;

};

struct Queue { struct

    Node* front; struct

    Node* rear;
};

struct Node* createNode(int data) { struct Node* node = (struct

    Node*)malloc(sizeof(struct Node)); node->data = data; node->next = NULL;

    return node;
}

struct Queue* createQueue() {

struct Queue* queue = (struct Queue*)malloc(sizeof(struct Queue)); queue-

    >front = NULL; queue->rear = NULL; return queue;
}

int isEmpty(struct Queue* queue) {
    return queue->front == NULL;

}


void enqueue(struct Queue* queue, int data) { struct

Node* node = createNode(data); if (queue->rear ==

NULL) { queue->front = queue->rear = node; return;
}

queue->rear->next = node; queue->rear = node;
}


int dequeue(struct Queue* queue) { if

(isEmpty(queue)) { prinK("Queue underflow\n");

return NULL;
}

struct Node* temp = queue->front;

int data = temp->data;

queue->front = queue->front->next;

if (queue->front == NULL) queue->rear = NULL;

free(temp);

return data;
}
```

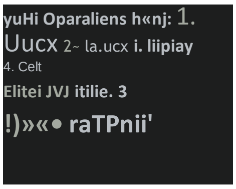
```
void display(struct Queue* queue) { if

(isEmpty(queue)) { prinK("Queue is

empty\n");
```

```
  return;
}
struct Node* temp = queue->front;
printff'Queue contents:\n"); while (temp !=
NULL) { printf("%d ", temp->data); temp =
temp->next;
}
printf("\n");
}

int main() {
struct Queue* queue = createQueue(); int
choice, value;

while (1) {
printf("\nQueue OperaQons Menu:\n");
printf("1. Enqueue\n");
printf("2. Dequeue\n");
printf("3. Display\n");
printf("4. Exit\n");
printff'Enter your choice: ");
scanf("%d", &choice);

switch (choice) { case 1:
printf("Enter value to enqueue: ");
scanf("%d", &value); enqueue(queue, value);
printff'Enqueued: %d\n", value);
```

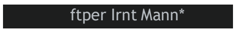
```
    break; case 2:

    value = dequeue(queue); if (value != NULL) {

printf("Dequeued: %d\n", value);
    }

    break; case 3:

    display(queue); break; case 4:

    printff'ExiQng program.\n"); exit(0); default:

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

    }

    return 0;

    }
```



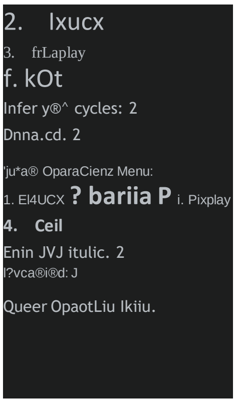
```
r)iirir Tp'rat-nr*: Umi i!
I 1 . Fra |i»* i»*
▶ J. rirqir<ir 3. >1    1<f
i. Exit
Lrter your choice: 1 Utter tmiue to
enqueue: 1 Utqueued: 1

Queue Dpeiei-uis Meru:
1.   Fi'i |iai ii'
2.    Dequeje
3.
1. Fail
Fntrr ynir rhnlrr: I Utter calLfe to
enqueue: t  Enqueued: 2

queue Jptratujrs Menu:
1.    ti q jeue
2.    Dequeue
3.    Display i. Fxir
Eli let yuui Lidice; 1 Frtrr unlur to
mqiriir! 3 Enqueued: J

queue Jptrtrtiors Ueru:
1. Lrqueje i . Jequeue
3. Diiiilsy '. Exit
Fit I HI yn ir i lei ill*: 1 0u*jr LuileiiU.:
1 3 3

f)urir Dp'rat-nr*: U'nii!
1.    Lrqueje
2.    Dequeue
J. >15 0-5/
Exit
Utter your choice: 1  Dequeued: 1
```



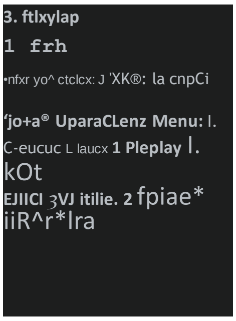
```
yuHi Oparaliens h«nj: 1.
Uucx 2~ la.ucx i. liipiay
4. Celt
Elitei JVJ itilie. 3
!)»«• raTPnii'
```



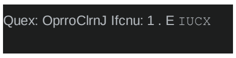
```
ftper Irnt Mann*
```



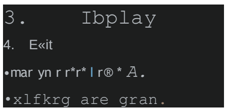
```
2.   Ixucx
3.   frLaplay
f. kOt
Infer y®^ cycles: 2
Dnna.cd. 2
'ju*a® OparaCienz Menu:
1. El4UCX ? bariia P i. Pixplay
4.   Ceil
Enin JVJ itulic. 2
I?vca®i®d: J
Queer OpaotLiu Ikiiu.
```



```
3. ftlxylap
1 frh
•nfxr yo^ ctclcx: J 'XK®: la cnpCi
'jo+a® UparaCLenz Menu: I.
C-eucuc L laucx 1 Pleplay l.
kOt
EJIICI 3VJ itilie. 2 fpiae*
iiR^r*lra
```



```
Quex: OprroClrnJ Ifcnu: 1 . E IUCX
```



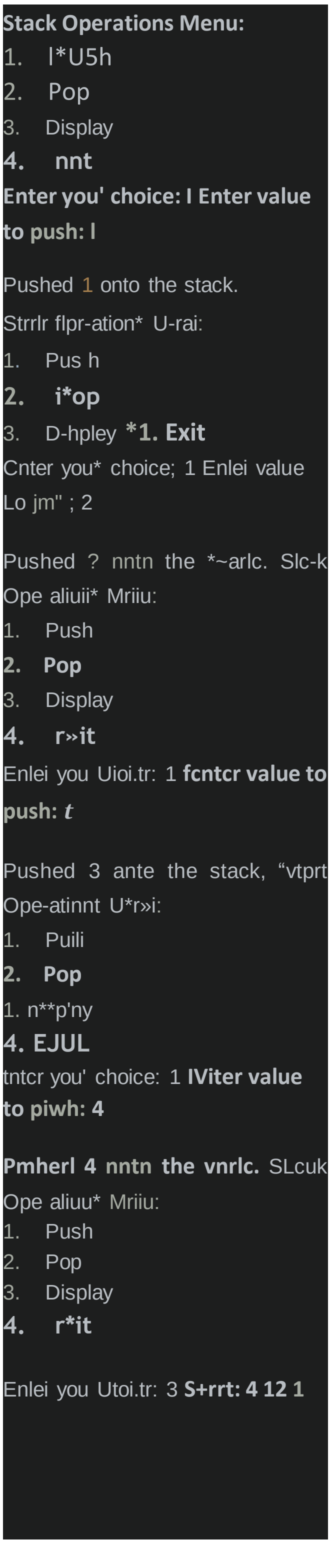
```
3.     Ibplay
4.   E«it
•mar yn r r*r* I r® * A.
•xlfkrg are gran.
```

```
#include <stdio.h>

#include <stdlib.h>


// Define the Node structure

struct Node { int data;

struct Node* next;
};




// FuncQon to create a new node struct Node* createNode(int data) {

struct Node* node = (struct Node*)malloc(sizeof(struct Node)); node-

>data = data; node->next = NULL; return node;
}


// FuncQon to check if the stack is empty int isEmpty(struct Node* top) {

return top == NULL;
}


// FuncQon to push an element onto the stack void push(struct Node**

top, int data) { struct Node* node = createNode(data); node->next =

*top;

*top = node;

prinK("\nPushed %d onto the stack.", data);
}
```

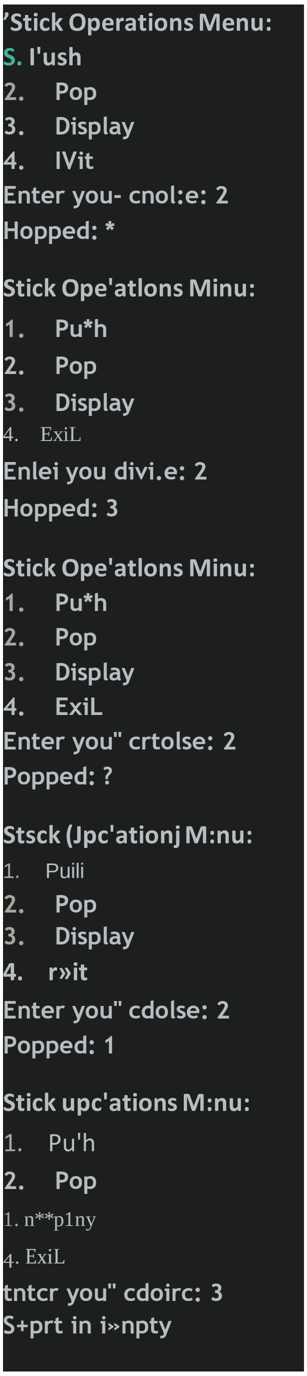
```
// Function to pop an element from the stack int pop(struct

Node** top) { if (isEmpty(*top)) { prinK("Stack underflow\n");

return -1; // Return -1 to indicate the stack is empty
}

struct Node* temp = *top; int data = temp->data;

*top = (*top)->next; free(temp); return data;
}


// Function to display the elements in the stack void

display(struct Node* top) { if (isEmpty(top)) { prinK("Stack is

empty\n"); return;
}

struct Node* temp = top; prinK("\nStack: "); while (temp !=

NULL) { prinK("%d ", temp->data); temp = temp->next;
}

prinK("\n");

}


// Main function with switch-based menu int main() {
```

```
struct Node* stack = NULL; int choice, value;

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

switch (choice) { case 1:
printff'Enter value to push: "); scanf("%d", &value);
push(&stack, value); break; case 2:
value = pop(&stack);
if (value != -1) { // Check for valid pop operation
printff'Popped: %d\n", value);
}

break; case 3:
display(stack); break; case 4:
printf("Exiting program.\n"); exit(0);
```

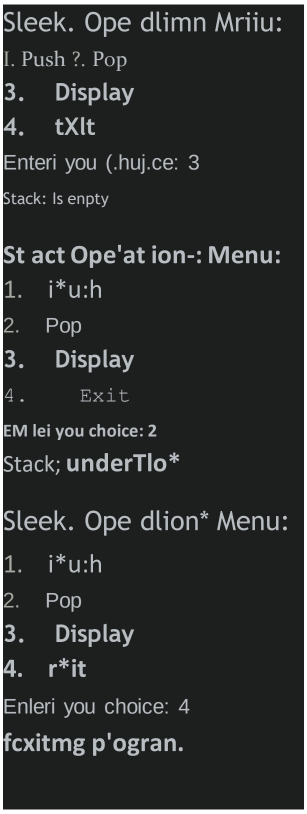
```
            default:
                printff'Invalid choice! Please try again.\n");
        }
    }

    return 0;
}
```



```
Stack Operations Menu:
1.   l*U5h
2.   Pop
3.   Display
4.   nnt
Enter you' choice: I Enter value
to push: l

Pushed 1 onto the stack.
Strrlr flpr-ation* U-rai:
1.   Pus h
2.   i*op
3.   D-hpley *1. Exit
Cnter you* choice; 1 Enlei value
Lo jm" ; 2

Pushed ? nntn the *~arlc. Slc-k
Ope aliuii* Mriiu:
1.   Push
2.   Pop
3.   Display
4.   r»it
Enlei you Uioi.tr: 1 fcntcr value to
push: t

Pushed 3 ante the stack, “vtprt
Ope-atinnt U*r»i:
1.   Puili
2.   Pop
1. n**p'ny
4. EJUL
tntcr you' choice: 1 IViter value
to piwh: 4

Pmherl 4 nntn the vnrlc. SLcuk
Ope aliuu* Mriiu:
1.   Push
2.   Pop
3.   Display
4.   r*it

Enlei you Utoi.tr: 3 S+rrt: 4 12 1
```



```
’Stick Operations Menu:
S. I'ush
2.    Pop
3.    Display
4.    IVit
Enter you- cnol:e: 2
Hopped: *

Stick Ope'atlons Minu:
1.    Pu*h
2.    Pop
3.    Display
4.    ExiL
Enlei you divi.e: 2
Hopped: 3

Stick Ope'atlons Minu:
1.    Pu*h
2.    Pop
3.    Display
4.    ExiL
Enter you" crtolse: 2
Popped: ?

Stsck (Jpc'ationj M:nu:
1.    Puili
2.    Pop
3.    Display
4.   r»it
Enter you" cdolse: 2
Popped: 1

Stick upc'ations M:nu:
1.   Pu'h
2.    Pop
1. n**p1ny
4. ExiL
tntcr you" cdoirc: 3
S+prt in i»npty
```



```
Sleek. Ope dlimn Mriiu:
I. Push ?. Pop
3.   Display
4.   tXlt
Enteri you (.huj.ce: 3
Stack: Is enpty

St act Ope'at ion-: Menu:
1.   i*u:h
2.   Pop
3.   Display
4.      Exit
EM lei you choice: 2
Stack; underTlo*

Sleek. Ope dlion* Menu:
1.   i*u:h
2.   Pop
3.   Display
4.   r*it
Enleri you choice: 4
fcxitmg p'ogran.
```

Lab Program 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

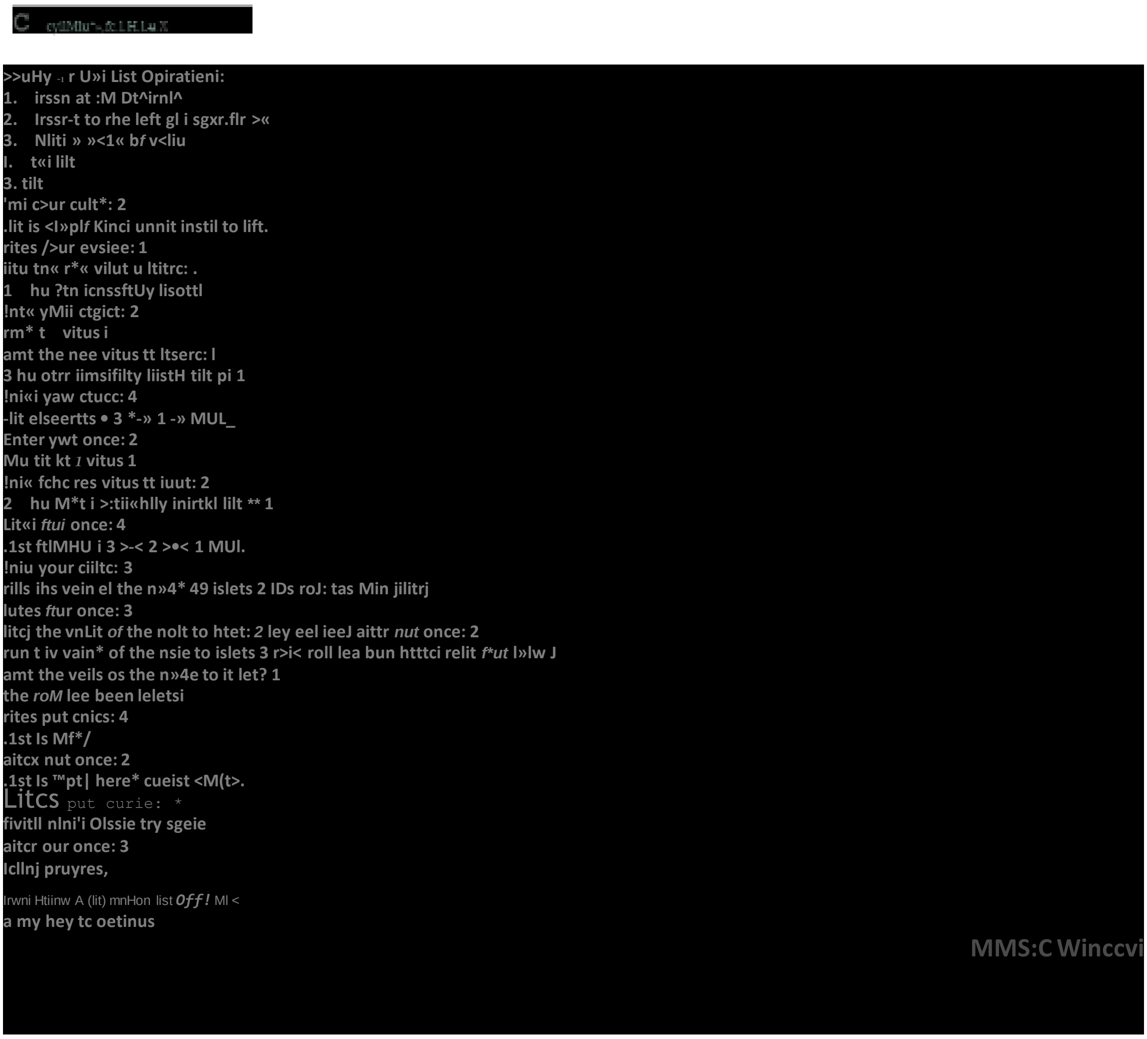
```
#include <stdio.h>

#include <stdlib.h>


typedef struct node { struct node* prev; int data;

struct node* next;

}node;


node* createnode(int data){ node*

    newnode=(node*)malloc(sizeof(node)); newnode-

    >prev=NULL; newnode->next=NULL; newnode-

    >data=data; return newnode;
}


struct node* createDoublyLinkedList() { return NULL;
}

     int isempty(node* head){ return head==NULL;
```

```
}


node* insert_at_beginning(int data, node* head){ node*

newnode=createnode(data); newnode->next=head;

if(head!=NULL) head->prev=newnode; head=newnode;

printf("%d has been successfully inserted.\n",data); return

head;
}


node* insert_tojeft(int data, int key, node* head) { node*

temp = head;


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

temp->next;
}


if (temp == NULL) { printff'Key not found\n"); return head;
}


node* newnode = createnode(data); newnode->next =

temp; newnode->prev = temp->prev;


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

```
    } else {
        head = newnode;
        }

        temp->prev = newnode;


        printf("%d has been successfully inserted left of %d\n", data, key);
        return head;
        }




node* deletenode(int key,node* head){ if(isempty(head)){
printff'List is empty hence cannot delete a node\n "); return
    head;
}

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

if(temp==NULL){ prinK("Key not found\n"); return head;
}

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

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

printff'The node has been deleted\n");

free(temp);

return head;
}


void display(node* head){ node* temp=head;

    if(isempty(head)){ printff'List is empty\n"); return;
}

printff'List elements : "); while(temp->next!=NULL){

    printf("%d <-> ",temp->data); temp=temp->next;
}

printf("%d -> NULL\n",temp->data);

}


int main() {

node* head = createDoublyLinkedList(); int choice,

    value, key;

printf("\nDoubly Linked List OperaQons:\n");

    printf("1. Insert at the beginning\n"); printf("2.

    Insert to the len of a specific node\n"); printf("3.

    Delete a node by value\n"); printf("4. Display the

    list\n");
```

```
printf("5. Exit\n");

while (1) {

    printff'Enter your choice: "); scanf("%d", &choice);


    switch (choice) { case 1:

    printff'Enter the new value to insert: "); scanf("%d", &value);

    head=insert_at_beginning(value,head); break; case 2:

    if(!isempty(head)){ printff'Enter the key value: "); scanf("%d",

    &key);

    printff'Enter the new value to insert: "); scanf("%d", &value);

    head=insert_to_left(value,key,head);
    }

    else

    printff'List is empty hence cannot insert to left.\n"); break;


    case 3:

    if(!isempty(head)){

    printff'Enter the value of the node to delete: "); scanf("%d",

    &value); head=deletenode(value,head);
    }

    else
    prinK("List is empty hence cannot delete.\n");
break;

    case 4:
    display(head);
    break;

    case 5:
    printff'ExiQng program.\n"); exit(0);

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



```
>>uHy -1 r U»i List Opiratieni:
1.   irssn at :M Dt^irnl^
2.   Irssr-t to rhe left gl i sgxr.flr >«
3.   Nliti » »<1« bf v<liu
I.   t«i lilt
3. tilt
'mi c>ur cult*: 2
.lit is <I»plf Kinci unnit instil to lift.
rites />ur evsiee: 1
iitu tn« r*« vilut u ltitrc: .
1    hu ?tn icnssftUy lisottl
!nt« yMii ctgict: 2
rm* t    vitus i
amt the nee vitus tt ltserc: l
3 hu otrr iimsifilty liistH tilt pi 1
!ni«i yaw ctucc: 4
-lit elseertts • 3 *-» 1 -» MUL_
Enter ywt once: 2
Mu tit kt 1 vitus 1
!ni« fchc res vitus tt iuut: 2
2    hu M*t i >:tii«hlly inirtkl lilt ** 1
Lit«i ftui once: 4
.1st ftlMHU i 3 >-< 2 >•< 1 MUl.
!niu your ciiltc: 3
rills ihs vein el the n»4* 49 islets 2 IDs roJ: tas Min jilitrj
lutes ftur once: 3
litcj the vnLit of the nolt to htet: 2 ley eel ieeJ aittr nut once: 2
run t iv vain* of the nsie to islets 3 r>i< roll lea bun htttci relit f*ut l»lw J
amt the veils os the n»4e to it let? 1
the roM lee been leletsi
rites put cnics: 4
.1st Is Mf*/
aitcx nut once: 2
.1st Is ™pt| here* cueist <M(t>.
Litcs put curie: *
fivitll nlni'i Olssie try sgeie
aitcr our once: 3
Icllnj pruyres,

Irwni Htiinw A (lit) mnHon list Off! Ml <
a my hey tc oetinus
```

Lab Program 8 Write a program

a) To construct a binary Search tree.

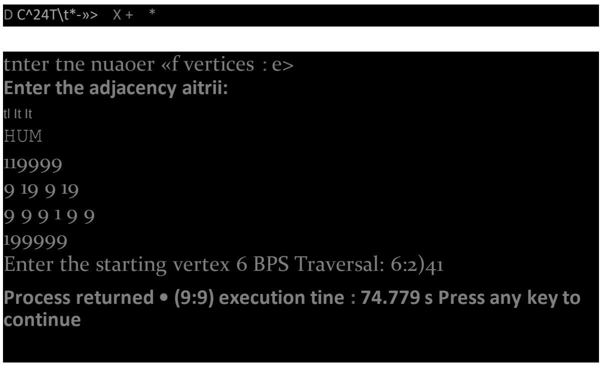
b) To traverse the tree using all the methods i.e., inorder, preorder and post order

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 = newNode->right = NULL; return newNode;
}


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

    createNode(data);
}

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

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

return root;

}


void inorderTraversal(struct Node* root) { if

(root == NULL) { return;
}

inorderTraversal(root->left); printf("%d ",

root->data); inorderTraversal(root->right);
}


void preorderTraversal(struct Node* root) { if

(root == NULL) { return;
}

printf("%d ", root->data);

preorderTraversal(root->left);

preorderTraversal(root->right);
}


void postorderTraversal(struct Node* root) {

if (root == NULL) { return;
}

postorderTraversal(root->left);

postorderTraversal(root->right); prinK("%d ",

root->data);
}
```

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

while (1) {

printf("\nBinary Search Tree OperaQons:\n");
printf("1. Insert a node\n"); printf("2. In-order
traversal\n"); printf("3. Pre-order traversal\n");
printf("4. Post-order traversal\n"); printf("5. Exit\n");
printff'Enter your choice: "); scanf("%d", &choice);

switch (choice) { case 1:
printff'Enter the value to insert: "); scanf("%d",
&data); root = insert(root, data); printff'Node %d
inserted.\n", data); break;

case 2:
prinK("In-order traversal: ");
inorderTraversal(root);
printf('V);
break;

case 3:
```

```
printf("Pre-order traversal: ");
preorderTraversal(root);
prmtffV);
break;

case 4:
printf("Post-order traversal: ");
postorderTraversal(root);
prmtffV);
break;

case 5:
pmtf("Exiting...\n");
exit(0);
default:
printf("Invalid choice, please try again.\n");
}
}

return 0;
}
```



```
tnter tne nuaoer «f vertices  : e>
Enter the adjacency aitrii:
tl It It
HUM
119999
9 19 9 19
9 9 9 1 9 9
199999
Enter the starting vertex 6 BPS Traversal: 6:2)41

Process returned • (9:9) execution tine : 74.779 s Press any key to
continue
```

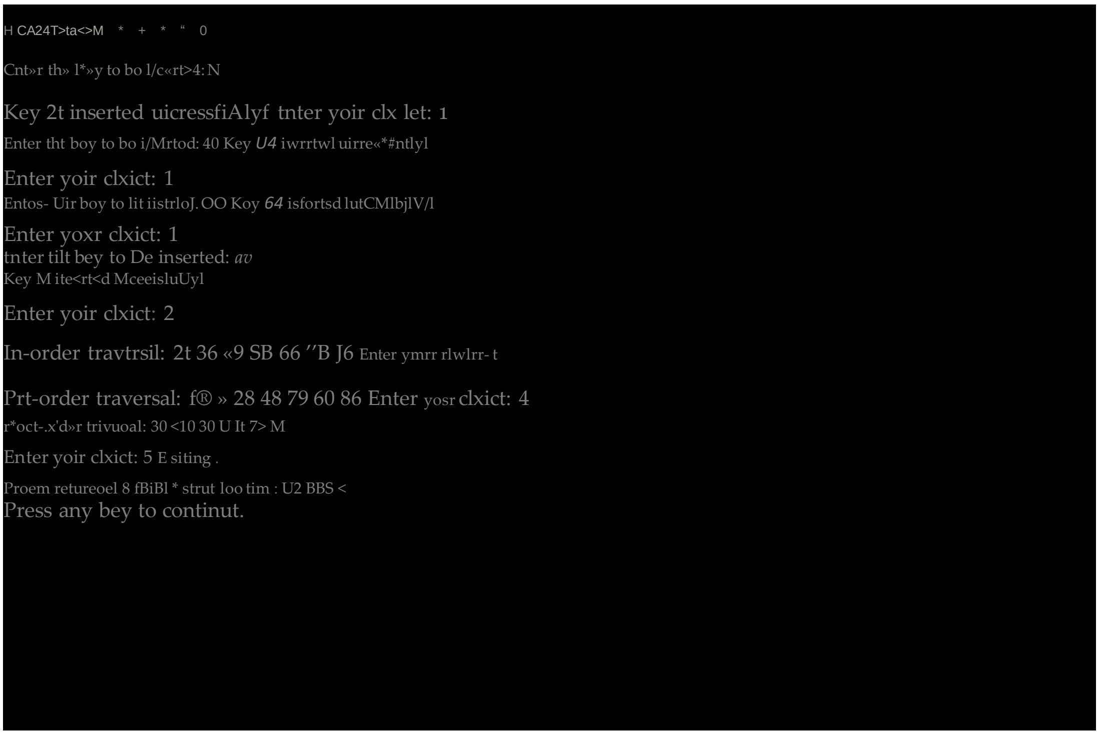
Lab Program 9

Write a program to traverse a graph using BFS method

```
#include <stdio.h>

#include <stdlib.h>


#define MAX 100


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


struct Queue* createQueue() {

struct Queue* q = (struct Queue*)malloc(sizeof(struct Queue)); q->front

= -1; q->rear = -1; return q;
}


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


void enqueue(struct Queue* q, int value) { if (q->rear == MAX - 1) {

prinK("Queue is full\n");

} else {

if (q->front == -1) {
   q->front = 0;
}
q->items[++q->rear] = value;
}
}

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

void bfs(int graph[MAX][MAX], int startVertex, int n) {
   int visited[MAX] = {0}; struct Queue* q =
   createQueue();

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

printff'BFS Traversal: ");
```

```
while (!isEmpty(q)) {

int currentVertex = dequeue(q);

printf("%d ", currentVertex);


  for (int i = 1; i <= n; i++) {

  if (graph[currentVertex][i] == 1 && !visited[i]) {

  visited[i] = 1; enqueue(q, i);

  }

  }

  }

  printf("\n");

  }


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


  printff'Enter the number of vertices : "); scanf("%d",

  &n);


  printff'Enter the adjacency matrix:\n"); for (int i = 1; i

  <= n; i++) { for (int j = 1; j <= n; j++) { scanf("%d",

  &graph[i][j]);
  }

  }
```

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

    bfs(graph, startVertex, n);

    return 0;
}
```



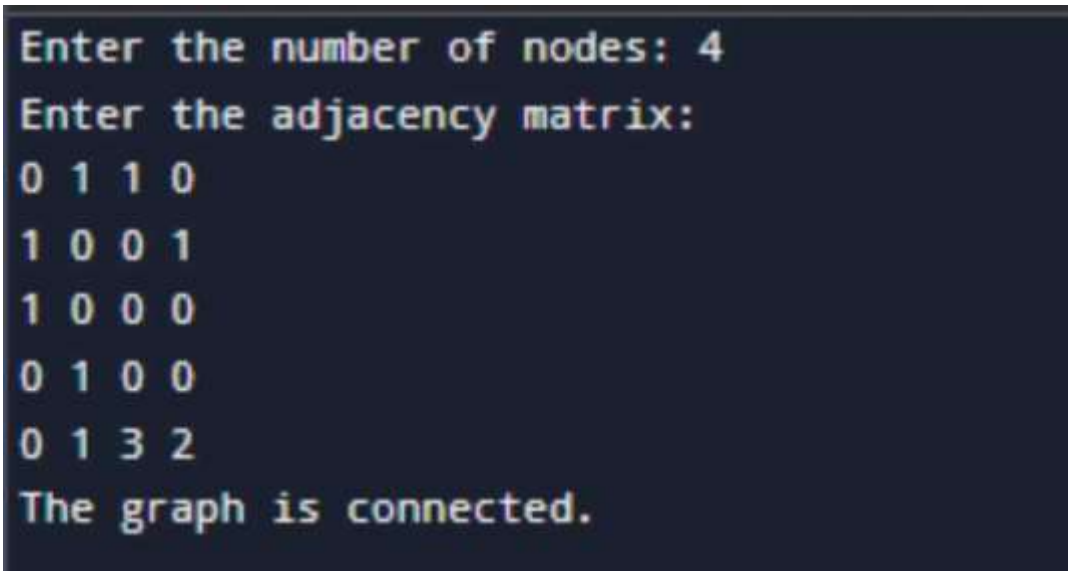
```
H CA24T>ta<>M   *   +   *   “   0
Cnt»r th» l*»y to bo l/c«rt>4: N
Key 2t inserted uicressfiAlyf tnter yoir clx let: 1
Enter tht boy to bo i/Mrtod: 40 Key U4 iwrrtwl uirre«*#ntlyl
Enter yoir clxict: 1
Entos- Uir boy to lit iistrloJ. OO Koy 64 isfortsd lutCMlbjlV/l
Enter yoxr clxict: 1
tnter tilt bey to De inserted: av
Key M ite<rt<d MceeisluUyl
Enter yoir clxict: 2
In-order travtrsil: 2t 36 «9 SB 66 ’’B J6 Enter ymrr rlwlrr- t
Prt-order traversal: f® » 28 48 79 60 86 Enter yosr clxict: 4
r*oct-.x'd»r trivuoal: 30 <10 30 U It 7> M
Enter yoir clxict: 5 E siting .
Proem retureoel 8 fBiBl * strut loo tim : U2 BBS <
Press any bey to continut.
```

Lab Program 9

Write a program to check whether given graph is connected or not using DFS method.

```
#include <stdio.h>


#define MAX NODES 100

int adjacencyMatrix[MAX_NODES][MAX_NODES]; int

visited[MAX_NODES]; int nodes;

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


for (int i = 0; i < nodes; i++) {

if (adjacencyMatrix[vertex][i] == 1 && !visited[i]) { DFS(i);

}

}

}

// Function to check if the graph is connected int

isConnected() {

// Initialize visited array to 0 for (int i = 0; i < nodes; i++) {

visited[i] = 0;
}
// Start DFS from node 0 DFS(0);

// Check if all nodes are visited for (int i = 0; i <
nodes; i++) { if (!visited[i]) {
return 0; // Graph is not connected
}
}
return 1; // Graph is connected
}

int main() {
printff'Enter the number of nodes: ");
scanf("%d", &nodes);

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

// Check connectivity if (isConnected()) {
printf("\nThe graph is connected.\n");
} else {
printf("\nThe graph is not connected.\n");
}
```

```
    return 0;

}
```



```
Enter the number of nodes: 4
Enter the adjacency matrix:
0 1 1 0
1 0 0 1
1 0 0 0
0 1 0 0
0 1 3 2
The graph is connected.
```