A

LAB FILE

ON

DATA STRUCTURE USING C

(COURSE CODE: CSIT-124)

Submitted By:

Name: Shourya Solanki

Enrollment No. A2305223569

Class & Section: B.Tech 3CSE-3(Y)

Submitted To:

Ms. Twinkle Tiwari

Assistant Professor

CSE Department

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

AMITY SCHOOL OF ENGINEERING AND TECHNOLOGY

AMITY UNIVERSITY UTTAR PRADESH, NOIDA

SESSION: ODD SEM. (2024-25)

INDEX

Student Name: Shourya Solanki **Enrollment No. A2305223569**

SR. NO.	EXPERIMENT NAME	DATE	PAGE NO.	SIGNATURE
1	Program on traversing & searching an element in the array.			
2	Write a program to insert a new element in the given unsorted array at kth position.			
3	Write a program to delete an element from given sorted array.			
4	Write a program to merge two given sorted arrays.			
5	Write a program to implement Stack using array. Also show overflow and underflow in respective push and pop operations.			
6	Write a program to convert infix to postfix expression.			
7	Write a program to implement Queue using array, which shows insertion and deletion operations.			
8	Write a program to implement Circular Queue using array, which shows insertion and deletion operations.			
9	Write a program to implement Linear Linked List, showing all the operations, like creation, display, insertion, deletion and searching.			
10	Write a program to count the number of times an item is present in a linked list.			
11	Write a program to increment the data part of every node present in a linked list by 10. Display the data both before incriminationand after incrimination.			
12	Write a program to implement Doubly Linked List, showing all the operations, like creation, display, insertion, deletion and searching.			

Faculty Signature

(Ms. Twinkle Tiwari)

INDEX

Student Name: Shourya Solanki **Enrollment No. A2305223569**

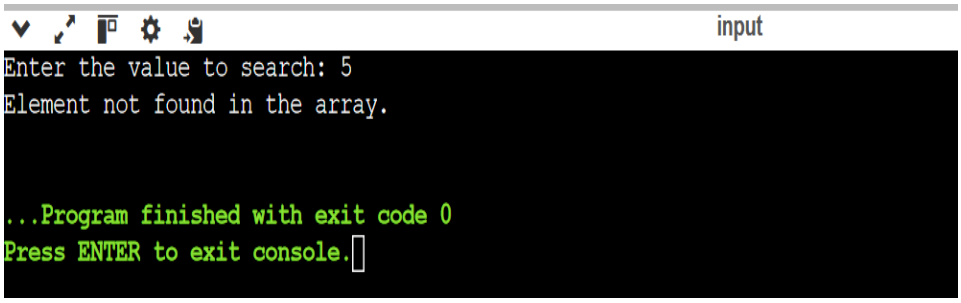
SR. NO.	EXPERIMENT NAME	DATE	PAGE NO.	SIGNATURE
13	Write a program to implement Stack, using Linked List. Implement Push, Pop and display operations.			
14	Write a program to implement Queue, using Linked List. Implement insertion, deletion and display operations.			
15	Write a program to create a Binary Search Tree and display its contents using recursive preorder, postorder and inorder traversal.			
16	Write a program to implement deletion of a node in binary search tree.			
17	Write a program to implement Binary tree and display the contents using non-recursive preorder, postorder and inorder traversal techniques.			
18	Write a program to sort the given array using Bubble Sort. Write a program to sort the given array using Selection Sort.			
19	Write a program to sort the given array using Insertion Sort. Write a program to sort the given array using Quick Sort.			
20	Write a program to sort the given array using Merge Sort.			
21	Write a program to sort the given array using Heap Sort.			
22	Write a program of Graph traversal - Depth first search.			
23	Write a program of Graph traversal - Breadth first search.			

Faculty Signature

(Ms. Twinkle Tiwari)

1) Program on traversing & searching an element in the array.

```
#include <stdio.h>
int main()
{
    int arr[5] = {10, 20, 30, 40, 50};
    int val, found = 0;
    printf("Enter the value to search: "); scanf("%d", &val);

    for (int i = 0; i < 5; i++) { if (arr[i] == val)
    {
    printf("Element found at index: %d\n", i); found = 1;
    }
}
    if (found == 0)
    {
    printf("Element not found in the array.\n");
    }
return 0;
}
```



2) Write a program to insert a new element in the given unsorted array at kth position.

```
#include <stdio.h>
int main()
{
    int arr[10] = {30, 10, 50, 20, 40};
    int n = 5; int val, k;
    printf("Enter the new element to insert: ");
    scanf("%d", &val);
    printf("Enter the position (0 to %d) to insert at: ", n);
    scanf("%d", &k);

    if (k < 0 || k > n)
```

```
    {
        printf("Invalid position!\n"); return 1;
    }
    for (int i = n; i > k; i--)
    {
        arr[i] = arr[i - 1];
    }
    arr[k] = val; n++;
    printf("Array after insertion: ");
    for (int i = 0; i < n; i++)
    {
    printf("%d ", arr[i]);
    }
    printf("\n"); return 0;
}
```

3) Write a program to delete an element from given sorted array.

```
#include <stdio.h>

int main()
{
    int arr[10] = {10, 20, 30, 40, 50};
    int n = 5;
    int val, pos = -1;
    printf("Array before deletion: ");
    for (int i = 0; i < n; i++)
    {
        printf("%d ", arr[i]);
    }
    printf("\n");
    printf("Enter the element to delete: ");
    scanf("%d", &val);
```

```
    for (int i = 0; i < n; i++)
    {
        if (arr[i] == val)
        {
            pos = i;
            break;
        }
    }

    if (pos == -1)
    {
        printf("Element not found in the array.\n");
    }
    else
    {
        for (int i = pos; i < n - 1; i++)
        {
            arr[i] = arr[i + 1];
        }
        n--;
        printf("Array after deletion: ");
        for (int i = 0; i < n; i++)
        {
            printf("%d ", arr[i]);
        }
        printf("\n");
    }
    return 0;
}
```

4) Write a program to merge two given sorted arrays.

```
#include <stdio.h>
int main()
```

```
{
    int arr1[5] = {10, 20, 30, 40, 50};
    int arr2[4] = {15, 25, 35, 45};
    int arr3[9];
    int n1 = 5, n2 = 4, n3 = 0; int i = 0, j = 0;
    while (i < n1 && j < n2)
    {
        if (arr1[i] < arr2[j]) { arr3[n3++] = arr1[i++];
    }
    else
    {
        arr3[n3++] = arr2[j++];
    }
}
    while (i < n1)
    {
        arr3[n3++] = arr1[i++];
    }
    while (j < n2)
    {
        arr3[n3++] = arr2[j++];
    }
    printf("Merged array: "); for (int k = 0; k < n3; k++)
    {
        printf("%d ", arr3[k]);
    } printf("\n");
    return 0;
}
```

5) Write a program to implement Stack using array. Also show overflow and underflow in respective push and pop operations.

```
#include <stdio.h>
#define sz 100
```

```
int stack[sz];
int top = -1;

void display() {
    if (top == -1) {
        printf("Stack is empty!\n");
    } else {
        printf("Current stack: ");
        for (int i = 0; i <= top; i++) {
            printf("%d ", stack[i]);
        }
        printf("\n");
    }
}

int push() {
    int value;
    display();
    if (top == sz - 1) {
        printf("Stack is full!\n");
    } else {
        printf("Enter the value to push: ");
        scanf("%d", &value);
        stack[++top] = value;
        printf("Pushed %d onto the stack.\n", value);
    }
    return 0;
}

int pop() {
    display();
    if (top == -1) {
```

```
        printf("Stack is empty!\n");
    } else {
        printf("Popped %d from the stack.\n", stack[top--]);
    }
    return 0;
}

int menu() {
    int choice;
    printf("Menu:\n");
    printf("1. Push\n");
    printf("2. Pop\n");
    printf("3. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);
    return choice;
}

int main() {
    int ch;
    do {
        ch = menu();
        switch (ch) {
            case 1:
                push();
                 break;
            case 2:
                pop();
                break;
            case 3:
                printf("Done\n");
                break;
```

```
            default:
                printf("Invalid choice. Please enter a valid option.\n");
        }
    } while (ch != 3);

    return 0;
}
```

6) Write a program to convert infix to postfix expression.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct Node {
char data;
struct Node* next;
} Node;
Node* createNode(char data) {
Node* newNode = (Node*)malloc(sizeof(Node));

newNode->data = data; newNode->next = NULL; return newNode;
}
void push(Node** top, char data) { Node* newNode = createNode(data); newNode->next =
*top;
*top = newNode;
}
char pop(Node** top) { if (*top == NULL) {
return '\0';
}
Node* temp = *top; char data = temp->data;
*top = (*top)->next; free(temp);
return data;
}
char peek(Node* top) { if (top != NULL) {
```

```
return top->data;
}
return '\0';
}
int precedence(char op) { if (op == '+' || op == '-') {
return 1;
}
if (op == '*' || op == '/') { return 2;
}
return 0;
}
int isop(char c) {
return (c == '+' || c == '-' || c == '*' || c == '/');
}
int isdig(char c) {
return (c >= '0' && c <= '9');
}
int islet(char c) {
return (c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z');
}
void inpo(const char* in, char* po) {

Node* s = NULL; int poIndex = 0;
int inLength = strlen(in);

for (int i = 0; i < inLength; i++) { char c = in[i];
if (isdig(c) || islet(c)) { po[poIndex++] = c;
} else if (c == '(') { push(&s, c);
} else if (c == ')') {
while (s != NULL && peek(s) != '(') { po[poIndex++] = pop(&s);
}
pop(&s);
```

```
} else if (isop(c)) {
while (s != NULL && precedence(peek(s)) >= precedence(c)) { po[poIndex++] = pop(&s);
}
push(&s, c);
}
}
while (s != NULL) { po[poIndex++] = pop(&s);
}
po[poIndex] = '\0';
}
int main() { char in[100]; char po[100];
printf("Enter infix expression: "); scanf("%s", in);
inpo(in, po);
printf("Postfix expression: %s\n", po); return 0;
}
```

7) Write a program to implement Queue using array, which shows insertion and deletion operations.

```
#include <stdio.h>
#define max 50
int qa[max];

int rear = -1; int front = -1;
void insert()
{
int add_item;
if (rear == max - 1)
printf("Queue Overflow \n");
else
{
if (front == -1) front = 0;
printf("Insert the element in queue: ");
scanf("%d", &add_item);
```

```
rear = rear + 1;
qa[rear] = add_item;
}
}
void delete()
{
if (front == -1 || front > rear)
{
printf("Queue Underflow \n");
}
else
{
printf("Element deleted from queue is: %d\n", qa[front]); front = front + 1;
if (front > rear)
{
front = rear = -1;
}
}
}
void display()
{
int i;
if (front == -1)
{
printf("Queue is empty \n");
}
else
{
printf("Queue is: \n");
for (i = front; i <= rear; i++)

printf("%d ", qa[i]); printf("\n");
```

```
}
}
int main()
{
int choice; while (1)
{
printf("\nMenu:\n");
printf("1. Insert element in queue\n");
printf("2. Delete element from queue\n"); printf("3. Display queue\n");
printf("4. Exit\n"); printf("Enter your choice: ");
scanf("%d", &choice);

switch (choice)
{
case 1:
insert(); break;
case 2:
delete(); break;
case 3:
display(); break;
case 4:
return 0; default:
printf("Invalid choice! Please enter a number between 1 and 4.\n"); break;
}
}
}
```

8) Write a program to implement Circular Queue using array, which shows insertion and deletion operations.

```
#include <stdio.h>
# define max 100
int queue[max];
int front=-1;
```

```
int rear=-1;
void enqueue(int ele)
{
if(front==-1 && rear==-1)
{
front=0; rear=0;
queue[rear]=ele;
}
else if((rear+1)%max==front)
{
printf("Queue is overflow!");
}
else
{
rear=(rear+1)%max;
queue[rear]=ele;
}
}
int dequeue()
{
if((front==-1) && (rear==-1))
{
printf("\nQueue is underflow!");
}
else if(front==rear)
{
printf("\nThe dequeued element is: %d", queue[front]); front=-1;
rear=-1;
}
else
{
```

```
printf("\nThe dequeued element is: %d", queue[front]); front=(front+1)%max;
}
}
void display()
{
int i=front;
if(front==-1 && rear==-1)
{
printf("\n Queue is empty!");
}

else
{
printf("\nElements in a Queue are:"); while(i<=rear)
{
printf("%d,", queue[i]); i=(i+1)%max;
}
}
}
int main()
{
int choice=1,x;

while(choice<4 && choice!=0)
{
printf("\n1.Insert an element");
printf("\n2.Delete an element");
printf("\n3.Display the element");
printf("\nEnter your choice: ");
scanf("%d", &choice);

switch(choice)
```

```
{
case 1:
printf("Enter the element which is to be inserted: ");
scanf("%d", &x);
enqueue(x);
break;
case 2: dequeue();
break;
case 3: display();
break;
}
}
return 0;
}
```

9) Write a program to implement Linear Linked List, showing all the operations, like creation, display, insertion, deletion and searching.

```
#include   <stdio.h>
#include  <stdlib.h>
typedef struct Node
{
int data;
struct Node* next;
}
Node;
void newN(Node** head, int value) {
Node* newNode = (Node*)malloc(sizeof(Node));
newNode->data = value;
newNode->next = *head;
*head = newNode;
}
void dis(Node* head)
{
   if (head == NULL)
```

```
    {
        printf("List is empty!\n");
    }
    else
    {
        Node* temp = head;
        printf("Linked List: ");
        while (temp != NULL)
        {
            printf("%d -> ", temp->data); temp = temp->next;
        }
        printf("NULL\n");
    }
}
void insert(Node** head, int value, int position)
{ Node* newNode = (Node*)malloc(sizeof(Node)); newNode->data = value;
if (position == 1)
{
    newNode->next = *head;
    *head = newNode; return;
}
Node* temp = *head;
for (int i = 1; i < position - 1 && temp != NULL; i++)
{
    temp = temp->next;
}
if (temp == NULL)
{
    printf("Invalid position!\n"); free(newNode);
}
else
{
```

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

}
}
void deln(Node** head, int value) { Node* temp = *head;
Node* prev = NULL;
if (temp != NULL && temp->data == value)
{
    *head = temp->next; free(temp);
return;
}
while (temp != NULL && temp->data != value)
{
    prev = temp;
    temp = temp->next;
}
if (temp == NULL)
{
    printf("Value not found!\n"); return;
}
prev->next = temp->next; free(temp);
}
void search(Node* head, int value)
{
    Node* temp = head;
    int position = 1;
while (temp != NULL)
{
    if (temp->data == value) {
        printf("Element %d found at position %d\n", value, position); return;
}
temp = temp->next; position++;
```

```
}
printf("Element %d not found in the list.\n", value);
}
int main() {
Node* head = NULL; int c, value, position;
do {
    printf("\nMenu:\n");
    printf("1. Create Node\n");
    printf("2. dis List\n");
    printf("3. Insert Node\n");
    printf("4. Delete Node\n");
    printf("5. Search Element\n");
    printf("6. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &c);

switch (c) { case 1:
printf("Enter value to create node: "); scanf("%d", &value);
newN(&head, value); break;
case 2:
dis(head); break;
case 3:
printf("Enter value to insert: "); scanf("%d", &value); printf("Enter position to insert: ");
scanf("%d", &position); insert(&head, value, position); break;
case 4:
printf("Enter value to delete: "); scanf("%d", &value); deln(&head, value);
break; case 5:
printf("Enter value to search: "); scanf("%d", &value); search(head, value);
break; case 6:
break; default:
printf("Invalid choice.\n");
}
} while (c != 6); Node* temp;
```

```
    while (head != NULL) { temp = head;
    head = head->next; free(temp);
    }
    return 0;
    }
```

10) Write a program to count the number of times an item is present in a linked list.

```
#include <stdio.h>
 #include <stdlib.h>
 typedef struct Node {
    int data;
    struct Node* next;
 } Node;
 void newN(Node** head, int value) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = value;
    newNode->next = *head;
    *head = newNode;
 }
 int occ(Node* head, int item) {
    int c = 0;
    Node* temp = head;

    while (temp != NULL) {
        if (temp->data == item) {
            c++;
        }
        temp = temp->next;
    }
    return c;
 }
 void dis(Node* head) {
    if (head == NULL) {
        printf("List is empty!\n");
    } else {
        Node* temp = head;
        printf("Linked List: ");
        while (temp != NULL) {
            printf("%d -> ", temp->data);
            temp = temp->next;
        }
        printf("NULL\n");
```

```
    }
}

int main() {
    Node* head = NULL;
    int value, item;
```

```
    newN(&head, 10);
    newN(&head, 20);
    newN(&head, 30);
    newN(&head, 20);
    newN(&head, 40);
    newN(&head, 20);
    dis(head);
    printf("Enter the item to count in the list: ");
    scanf("%d", &item);
    int c = occ(head, item);
    printf("Item %d appears %d time(s) in the list.\n", item, c);
    Node* temp;
    while (head != NULL) {
        temp = head;
        head = head->next;
        free(temp);
    }
    return 0;
}
```

11) Write a program to increment the data part of every node present in a linked list by 10. Display the data both before and after incrimination.

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node {
    int data;
    struct Node* next;
} Node;
void newN(Node** head, int value) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = value;
    newNode->next = *head;
    *head = newNode;
}
void dis(Node* head) {
    if (head == NULL) {
        printf("List is empty!\n");
    } else {
        Node* temp = head;
        while (temp != NULL) {
            printf("%d -> ", temp->data);
            temp = temp->next;
        }
        printf("NULL\n");
```

```
    }
}
void inc(Node* head) {
    Node* temp = head;
    while (temp != NULL) {
        temp->data += 10;
        temp = temp->next;
    }
}

int main() {
    Node* head = NULL;
    newN(&head, 10);
    newN(&head, 20);
    newN(&head, 30);
    newN(&head, 40);
    printf("Linked List before incrementing: \n");
    dis(head);
    inc(head);
    printf("Linked List after incrementing: \n");
    dis(head);
    Node* temp;
    while (head != NULL) {
        temp = head;
        head = head->next;
        free(temp);
    }
    return 0;
}
```

12) Write a program to implement Doubly Linked List, showing all the operations, like creation, display, insertion, deletion and searching.

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node {
    int data;
    struct Node* next;
    struct Node* prev;
} Node;
void create(Node** head, int value) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = value;
    newNode->next = NULL;
```

```
    newNode->prev = NULL;

    if (*head == NULL) {
        *head = newNode;
    } else {
        Node* temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->prev = temp;
    }
}
void disf(Node* head) {
    if (head == NULL) {
        printf("List is empty!\n");
        return;
    }
    Node* temp = head;
    printf("Doubly Linked List - Forward: ");
    while (temp != NULL) {
        printf("%d <-> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}
void disb(Node* head) {
    if (head == NULL) {
        printf("List is empty!\n");
        return;
    }
    Node* temp = head;
    while (temp->next != NULL) {
        temp = temp->next;
    }

    printf("Doubly Linked List - Backward: ");
    while (temp != NULL) {
        printf("%d <-> ", temp->data);
        temp = temp->prev;
    }
    printf("NULL\n");
}
void insert(Node** head, int value, int position) {
```

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

    if (position == 1) {
        newNode->next = *head;
        newNode->prev = NULL;
        if (*head != NULL) {
            (*head)->prev = newNode;
        }
        *head = newNode;
        return;
    }

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

    if (temp == NULL) {
        printf("Invalid position!\n");
        free(newNode);
        return;
    }

    newNode->next = temp->next;
    if (temp->next != NULL) {
        temp->next->prev = newNode;
    }
    temp->next = newNode;
    newNode->prev = temp;
}
void deln(Node** head, int value) {
    Node* temp = *head;
    while (temp != NULL && temp->data != value) {
        temp = temp->next;
    }

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

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

```
    } else {
        *head = temp->next;
    }

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

    free(temp);
}
void search(Node* head, int value) {
    Node* temp = head;
    int position = 1;
    while (temp != NULL) {
        if (temp->data == value) {
            printf("Element %d found at position %d\n", value, position);
            return;
        }
        temp = temp->next;
        position++;
    }
    printf("Element %d not found in the list.\n", value);
}

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

    do {
        printf("\nMenu:\n");
        printf("1. Create Node\n");
        printf("2. Display List Forward\n");
        printf("3. Display List Backward\n");
        printf("4. Insert Node\n");
        printf("5. Delete Node\n");
        printf("6. Search Element\n");
        printf("7.  Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to create node: ");
                scanf("%d", &value);
```

```
            create(&head, value);
            break;
        case 2:
            disf(head);
            break;
        case 3:
            disb(head);
            break;
        case 4:
            printf("Enter value to insert: ");
            scanf("%d", &value);
            printf("Enter position to insert: ");
            scanf("%d", &position);
            insert(&head, value, position);
            break;
        case 5:
            printf("Enter value to delete: ");
            scanf("%d", &value);
            deln(&head, value);
            break;
        case 6:
            printf("Enter value to search: ");
            scanf("%d", &value);
            search(head, value);
            break;
        case 7:
            break;
        default:
            printf("Invalid choice!\n");
      }
   } while (choice != 7);
   return 0;
}
```

13) Write a program to implement Stack using Linked List. Implement Push, Pop and display operations.

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node {
   int data;
   struct Node* next;
} Node;
void push(Node** top, int value) {
   Node* newNode = (Node*)malloc(sizeof(Node));
```

```
    if (newNode == NULL) {
        printf("Memory allocation failed!\n");
        return;
    }
    newNode->data = value;
    newNode->next = *top;
    *top = newNode;
    printf("Pushed %d onto the stack.\n", value);
}
void pop(Node** top) {
    if (*top == NULL) {
        printf("Stack is empty!\n");
        return;
    }
    Node* temp = *top;
    *top = (*top)->next;
    printf("Popped %d from the stack.\n", temp->data);
    free(temp);
}
void dis(Node* top) {
    if (top == NULL) {
        printf("Stack is empty!\n");
        return;
    }
    printf("Stack elements: ");
    Node* temp = top;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
int main() {
    Node* stack = NULL;
    int choice, value;

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

```
        switch (choice) {
            case 1:
                printf("Enter value to push: ");
                scanf("%d", &value);
                push(&stack, value);
                break;
            case 2:
                pop(&stack);
                break;
            case 3:
                dis(stack);
                break;
            case 4:
                break;
            default:
                printf("Invalid choice.\n");
        }
    } while (choice != 4);
    while (stack != NULL) {
        pop(&stack);
    }
    return 0;
}
```

14) Write a program to implement Queue, using Linked List. Implement Insertion, Deletion and display operations.

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node {
    int data;
    struct Node* next;
} Node;
void enqueue(Node** front, Node** rear, int value) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    if (newNode == NULL) {
        printf("Memory allocation failed!\n");
        return;
    }
    newNode->data = value;
    newNode->next = NULL;

    if (*rear == NULL) {
        *front = newNode;
```

```
        *rear = newNode;
    } else {
        (*rear)->next = newNode;
        *rear = newNode;
    }
    printf("Enqueued %d into the queue.\n", value);
}
void dequeue(Node** front, Node** rear) {
    if (*front == NULL) {
        printf("Queue is empty!\n");
        return;
    }
    Node* temp = *front;
    *front = (*front)->next;
    if (*front == NULL) {
        *rear = NULL;
    }
    printf("Dequeued %d from the queue.\n", temp->data);
    free(temp);
}
void dis(Node* front) {
    if (front == NULL) {
        printf("Queue is empty!\n");
        return;
    }
    printf("Queue elements: ");
    Node* temp = front;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
int main() {
    Node* front = NULL;
    Node* rear = NULL;
    int c, value;
    do {
        printf("\nMenu:\n");
        printf("1. Enqueue\n");
        printf("2. Dequeue\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your Choice: ");
```

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

        switch
            (c)
            {
            cas
            e 1:
                printf("Enter value to enqueue:
                ");scanf("%d", &value);
                enqueue(&front, &rear, value);
                break;
            case 2:
                dequeue(&front,
                &rear);break;
            case 3:
                dis(fr
                ont);
                brea
                k;
            case 4:
                b
            rea
            k;
            def
            ault:
                printf("Invalid Choice!\n");
        }
    } while (c!=4);
    while (front != NULL) {
        dequeue(&front,
        &rear);
    }
    return 0;
}
```

15. Write a program to create a binary search tree and display its contents using recursive preorder, postorder and inorder traversal.

```
#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)
    {
        root = createNode(data);
    }
    else if (data < root->data)
    {
        root->left = insert(root->left, data);
    }
    else
    {
        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);
```

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

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

    while(1)
    {
        printf("Enter 1 to insert a value, 0 to exit: ");
        scanf("%d", &choice);
        if (choice == 0) break;
        printf("Enter value to insert: ");
        scanf("%d", &value);
        root = insert(root, value);
    }

    printf("\nInorder Traversal: ");
    inorderTraversal(root);
    printf("\nPreorder Traversal: ");
    preorderTraversal(root);
    printf("\nPostorder Traversal: ");
    postorderTraversal(root);

    return 0;
}
```

OUTPUT

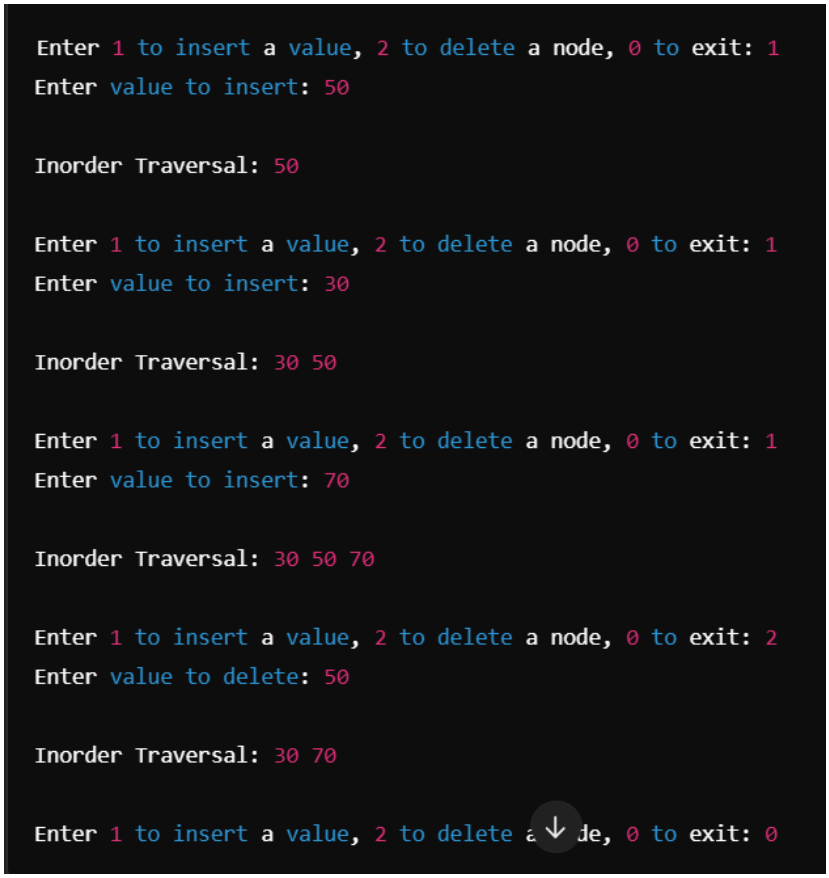
```
Enter 1 to insert a value, 0 to exit: 1
Enter value to insert: 30
Enter 1 to insert a value, 0 to exit: 1
Enter value to insert: 20
Enter 1 to insert a value, 0 to exit: 1
Enter value to insert: 40
Enter 1 to insert a value, 0 to exit: 0

Inorder Traversal: 20 30 40
Preorder Traversal: 30 20 40
Postorder Traversal: 20 40 30
```

16. Write a program to implement deletion of a node in binary search 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)
    {
        root = createNode(data);
    }
    else if (data < root->data)
    {
        root->left = insert(root->left, data);
    }
    else
    {
        root->right = insert(root->right, data);
    }
    return root;
}
```

OUTPUT



17. Write a program to implement Binary tree and display the contents using non-recursive preorder, postorder and inorder traversal techniques.

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

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

struct Stack
{
    int top;
    int capacity;
    struct Node** array;
};

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 Stack* createStack(int capacity)
{
    struct Stack* stack = (struct Stack*)malloc(sizeof(struct Stack));
    stack->top = -1;
    stack->capacity = capacity;
    stack->array = (struct Node**)malloc(stack->capacity * sizeof(struct Node*));
    return stack;
}

int isEmpty(struct Stack* stack)
{
    return stack->top == -1;
}

void push(struct Stack* stack, struct Node* node)
{
    stack->array[++stack->top] = node;
}

struct Node* pop(struct Stack* stack)
{
    return stack->array[stack->top--];
}

struct Node* peek(struct Stack* stack)
{
    return stack->array[stack->top];
}

void inorderTraversal(struct Node* root)
{
```

```
    struct Stack* stack = createStack(100);
    struct Node* current = root;

    while (current != NULL || !isEmpty(stack))
    {
        while (current != NULL)
        {
            push(stack, current);
            current = current->left;
        }
        current = pop(stack);
        printf("%d ", current->data);

        current = current->right;
    }
    free(stack);
}

void preorderTraversal(struct Node* root)
{
    if (root == NULL) return;

    struct Stack* stack = createStack(100);
    push(stack, root);

    while (!isEmpty(stack))
    {
        struct Node* current = pop(stack);
        printf("%d ", current->data);

        if (current->right != NULL) push(stack, current->right);
        if (current->left != NULL) push(stack, current->left);
    }
    free(stack);
}

void postorderTraversal(struct Node* root)
{
    if (root == NULL) return;

    struct Stack* stack1 = createStack(100);
    struct Stack* stack2 = createStack(100);

    push(stack1, root);
    struct Node* current;

    while (!isEmpty(stack1))
    {
        current = pop(stack1);
        push(stack2, current);

        if (current->left != NULL) push(stack1, current->left);
        if (current->right != NULL) push(stack1, current->right);
    }

    while (!isEmpty(stack2))
    {
        current = pop(stack2);
        printf("%d ", current->data);
    }
```

```
    free(stack1);
    free(stack2);
}

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

    root = createNode(10);
    root->left = createNode(20);
    root->right = createNode(30);
    root->left->left = createNode(40);
    root->left->right = createNode(50);
    root->right->left = createNode(60);
    root->right->right = createNode(70);

    printf("Inorder Traversal: ");
    inorderTraversal(root);
    printf("\nPreorder Traversal: ");
    preorderTraversal(root);
    printf("\nPostorder Traversal: ");
    postorderTraversal(root);
    printf("\n");

    return 0;
}
```

OUTPUT