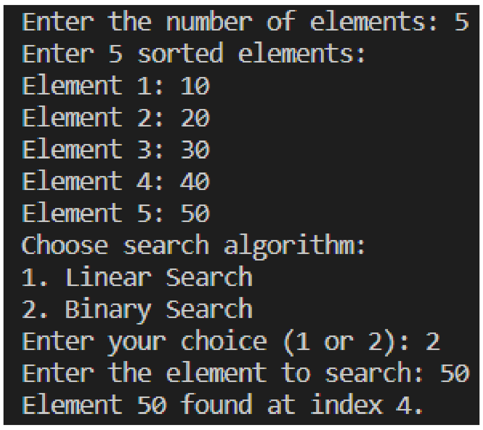
INDEX

Ques 1 : Write a Program to implement Linear and Binary search in an array.

```c
#include <stdio.h>

int linearSearch(int arr[], int n, int key) {
    for (int i = 0; i < n; i++) {
        if (arr[i] == key) {
            return i;
        }
    }
    return -1;
}

int binarySearch(int arr[], int low, int high, int key) {
    while (low <= high) {
        int mid = low + (high - low) / 2;

        if (arr[mid] == key) {
            return mid;
        } else if (arr[mid] < key) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }

    return -1;
}

int main() {
    int n;
    printf("Enter the number of elements: ");
    scanf("%d", &n);

    int arr[n];
    printf("Enter %d sorted elements:\n", n);
    for (int i = 0; i < n; i++) {
        printf("Element %d: ", i + 1);
        scanf("%d", &arr[i]);
    }

    int choice, key, result;

    printf("Choose search algorithm:\n");
    printf("1. Linear Search\n");
    printf("2. Binary Search\n");
    printf("Enter your choice (1 or 2): ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("Enter the element to search: ");
            scanf("%d", &key);
            result = linearSearch(arr, n, key);
            break;

        case 2:
            printf("Enter the element to search: ");
            scanf("%d", &key);
            result = binarySearch(arr, 0, n - 1, key);
            break;

        default:
            printf("Invalid choice.\n");
            return 1;
    }

    if (result != -1) {
        printf("Element %d found at index %d.\n", key, result);
    } else {
        printf("Element not found in the array.\n");
    }

    return 0;
}
```

OUTPUT:



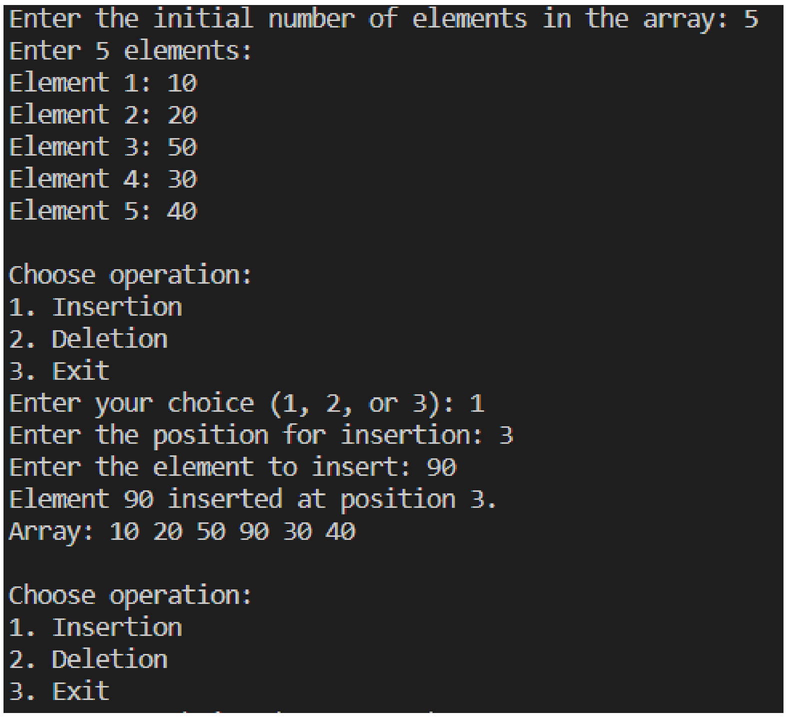
```
Enter the number of elements: 5
Enter 5 sorted elements:
Element 1: 10
Element 2: 20
Element 3: 30
Element 4: 40
Element 5: 50
Choose search algorithm:
1. Linear Search
2. Binary Search
Enter your choice (1 or 2): 2
Enter the element to search: 50
Element 50 found at index 4.
```

Ques 2 : Write a program to implement operations like insertion and deletion in an array.

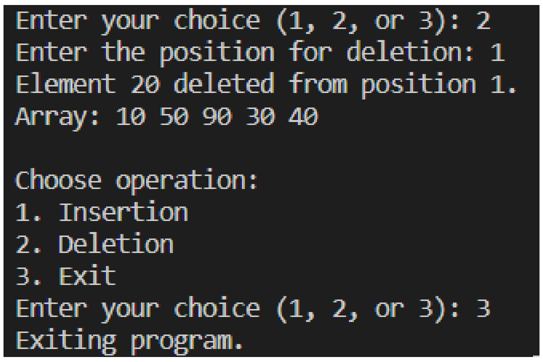
```
#include <stdio.h>

void displayArray(int arr[], int n) {
    printf("Array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

void insertElement(int arr[], int *n, int position, int element) {
    if (position < 0 || position > *n) {
        printf("Invalid position for insertion.\n");
        return;
    }

    for (int i = *n - 1; i >= position; i--) {
        arr[i + 1] = arr[i];
```

```c
    }

    arr[position] = element;
    (*n)++;

    printf("Element %d inserted at position %d.\n", element, position);
}

void deleteElement(int arr[], int *n, int position) {
    if (position < 0 || position >= *n) {
        printf("Invalid position for deletion.\n");
        return;
    }

    int deletedElement = arr[position];

    for (int i = position; i < *n - 1; i++) {
        arr[i] = arr[i + 1];
    }

    (*n)--;

    printf("Element %d deleted from position %d.\n", deletedElement, position);
}

int main() {
    int n;
    printf("Enter the initial number of elements in the array: ");
    scanf("%d", &n);

    int arr[100];

    printf("Enter %d elements:\n", n);
    for (int i = 0; i < n; i++) {
        printf("Element %d: ", i + 1);
        scanf("%d", &arr[i]);
    }

    int choice, position, element;

    do {
        printf("\nChoose operation:\n");
        printf("1. Insertion\n");
        printf("2. Deletion\n");
        printf("3. Exit\n");
        printf("Enter your choice (1, 2, or 3): ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the position for insertion: ");
                scanf("%d", &position);
                printf("Enter the element to insert: ");
                scanf("%d", &element);
                insertElement(arr, &n, position, element);
                displayArray(arr, n);
                break;
            case 2:
                printf("Enter the position for deletion: ");
                scanf("%d", &position);
                deleteElement(arr, &n, position);
                displayArray(arr, n);
                break;

            case 3:
                printf("Exiting program.\n");
                break;

            default:
                printf("Invalid choice. Please enter 1, 2, or 3.\n");
        }
    } while (choice != 3);

    return 0;
}
```

OUTPUT:



```
Enter the initial number of elements in the array: 5
Enter 5 elements:
Element 1: 10
Element 2: 20
Element 3: 50
Element 4: 30
Element 5: 40

Choose operation:
1. Insertion
2. Deletion
3. Exit
Enter your choice (1, 2, or 3): 1
Enter the position for insertion: 3
Enter the element to insert: 90
Element 90 inserted at position 3.
Array: 10 20 50 90 30 40

Choose operation:
1. Insertion
2. Deletion
3. Exit
```

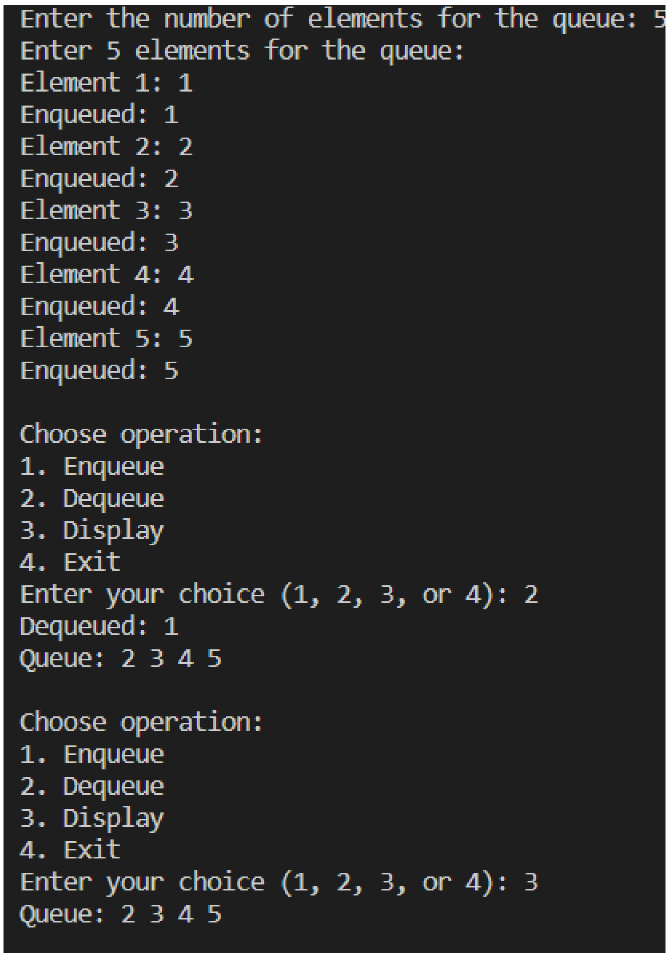


```
Enter your choice (1, 2, or 3): 2
Enter the position for deletion: 1
Element 20 deleted from position 1.
Array: 10 50 90 30 40

Choose operation:
1. Insertion
2. Deletion
3. Exit
Enter your choice (1, 2, or 3): 3
Exiting program.
```

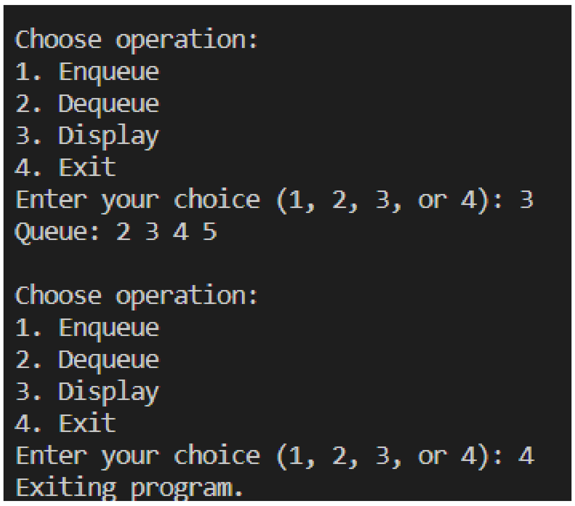
Ques 3 : Write a program to implement queue data structure along with its operations like enqueue, dequeue, display etc. using array.

```c
#include <stdio.h>

#define MAX_SIZE 100

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

void initializeQueue(struct Queue *queue) {
    queue->front = -1;
    queue->rear = -1;
}

int isFull(struct Queue *queue) {
    return (queue->rear == MAX_SIZE - 1);
}

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

void enqueue(struct Queue *queue, int element) {
    if (isFull(queue)) {
        printf("Queue is full. Cannot enqueue.\n");
    } else {
        if (isEmpty(queue)) {
            queue->front = 0;
        }
        queue->rear++;
        queue->items[queue->rear] = element;
        printf("Enqueued: %d\n", element);
    }
}

void dequeue(struct Queue *queue) {
    if (isEmpty(queue)) {
        printf("Queue is empty. Cannot dequeue.\n");
    } else {
        printf("Dequeued: %d\n", queue->items[queue->front]);
        if (queue->front == queue->rear) {
            initializeQueue(queue);
        } else {
            queue->front++;
        }
    }
}

void displayQueue(struct Queue *queue) {
    if (isEmpty(queue)) {
        printf("Queue is empty.\n");
    } else {
        printf("Queue: ");
        for (int i = queue->front; i <= queue->rear; i++) {
            printf("%d ", queue->items[i]);
        }
        printf("\n");
    }
}

int main() {
    struct Queue queue;
    initializeQueue(&queue);

    int n, element;

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

    printf("Enter %d elements for the queue:\n", n);
    for (int i = 0; i < n; i++) {
        printf("Element %d: ", i + 1);
        scanf("%d", &element);
        enqueue(&queue, element);
    }

    int choice;

    do {
        printf("\nChoose operation:\n");
        printf("1. Enqueue\n");
        printf("2. Dequeue\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice (1, 2, 3, or 4): ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the element to enqueue: ");
                scanf("%d", &element);
                enqueue(&queue, element);
                displayQueue(&queue);
                break;

            case 2:
                dequeue(&queue);
                displayQueue(&queue);
                break;

            case 3:
                displayQueue(&queue);
                break;

            case 4:
                printf("Exiting program.\n");
                break;

            default:
                printf("Invalid choice. Please enter 1, 2, 3, or 4.\n");
        }
    } while (choice != 4);

    return 0;
}
```

OUTPUT:



```
Enter the number of elements for the queue: 5
Enter 5 elements for the queue:
Element 1: 1
Enqueued: 1
Element 2: 2
Enqueued: 2
Element 3: 3
Enqueued: 3
Element 4: 4
Enqueued: 4
Element 5: 5
Enqueued: 5

Choose operation:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice (1, 2, 3, or 4): 2
Dequeued: 1
Queue: 2 3 4 5

Choose operation:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice (1, 2, 3, or 4): 3
Queue: 2 3 4 5
```

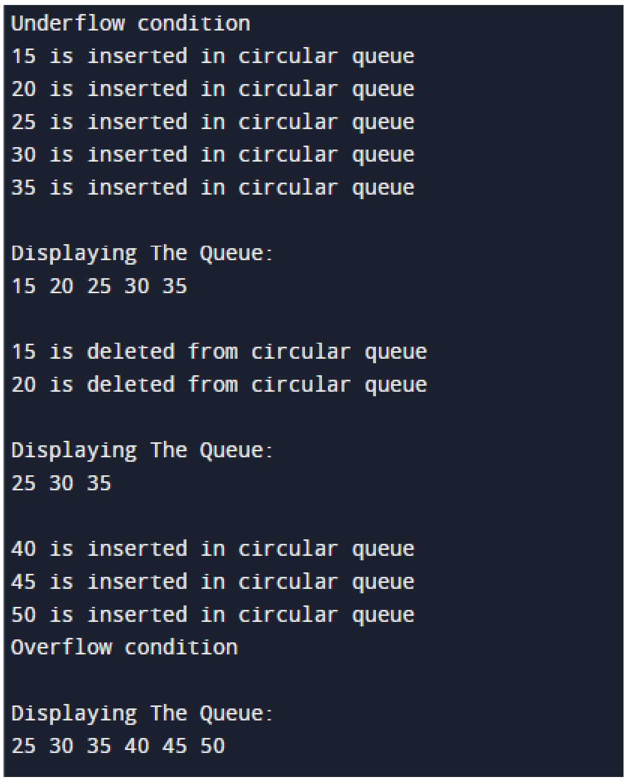


```
Choose operation:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice (1, 2, 3, or 4): 3
Queue: 2 3 4 5

Choose operation:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice (1, 2, 3, or 4): 4
Exiting program.
```

Ques 4 : Write a program to implement Circular Queue using an array.

```
#include<stdio.h>
#define capacity 6

int queue[capacity];
int front = -1, rear = -1;
int checkFull (){
    if ((front == rear + 1) || (front == 0 && rear == capacity - 1)){
        return 1;
    }
    return 0;
}

int checkEmpty (){
    if (front == -1){
      return 1;
    }
    return 0;
}

void enqueue (int value){
    if (checkFull ())
        printf ("Overflow condition\n");
    else{
        if (front == -1)
        front = 0;

        rear = (rear + 1) % capacity;
        queue[rear] = value;
        printf ("%d is inserted in circular queue\n", value);
    }
}

int dequeue (){
    int variable;
    if (checkEmpty ()){
        printf ("Underflow condition\n");
        return -1;
    }
    else{
        variable = queue[front];
        if (front == rear){
          front = rear = -1;
        }
        else{
            front = (front + 1) % capacity;
        }
        printf ("%d is deleted from circular queue\n", variable);
        return 1;
    }
}

void print (){
    int i;
    if (checkEmpty ())
        printf ("Nothing to dequeue\n");
    else{
        printf ("\nDisplaying The Queue: \n");
        for (i = front; i != rear; i = (i + 1) % capacity){
            printf ("%d ", queue[i]);
        }
        printf ("%d \n\n", queue[i]);
    }
}

int main (){
    dequeue ();

    enqueue (15);
    enqueue (20);
    enqueue (25);
    enqueue (30);
    enqueue (35);

    print ();
    dequeue ();
    dequeue ();

    print ();

    enqueue (40);
    enqueue (45);
    enqueue (50);
    enqueue (55);
    print ();

    return 0;
}
```

OUTPUT:

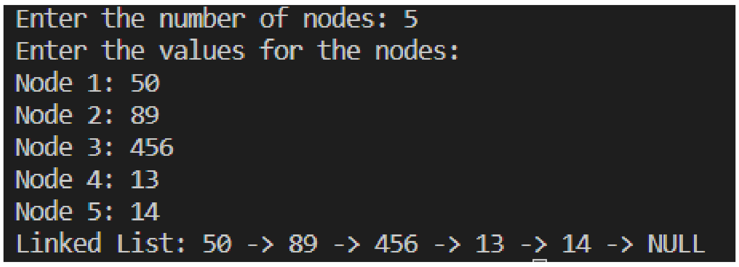


```
Underflow condition
15 is inserted in circular queue
20 is inserted in circular queue
25 is inserted in circular queue
30 is inserted in circular queue
35 is inserted in circular queue

Displaying The Queue:
15 20 25 30 35

15 is deleted from circular queue
20 is deleted from circular queue

Displaying The Queue:
25 30 35

40 is inserted in circular queue
45 is inserted in circular queue
50 is inserted in circular queue
Overflow condition

Displaying The Queue:
25 30 35 40 45 50
```

Ques 5 : Write a program to implement linked list with creation of nodes dynamically.

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

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

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

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

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

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

    int n, value;

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

    if (n <= 0) {
        printf("Invalid number of nodes. Exiting program.\n");
        return 1;
    }

    printf("Enter the values for the nodes:\n");
    for (int i = 1; i <= n; i++) {
        printf("Node %d: ", i);
        scanf("%d", &value);

        struct Node *newNode = createNode(value);

        if (head == NULL) {
            head = newNode;
            temp = head;
        } else {
            temp->next = newNode;
            temp = newNode;
        }
    }

    printf("Linked List: ");
    displayList(head);

    freeList(head);

    return 0;
}
```

OUTPUT:



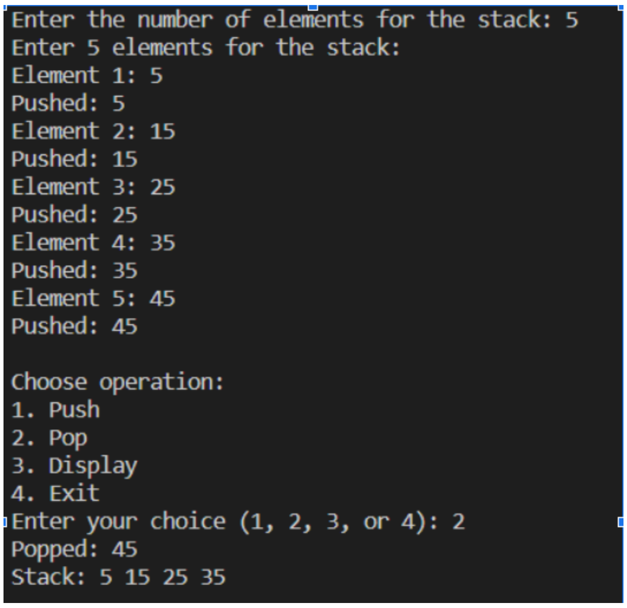
```
Enter the number of nodes: 5
Enter the values for the nodes:
Node 1: 50
Node 2: 89
Node 3: 456
Node 4: 13
Node 5: 14
Linked List: 50 -> 89 -> 456 -> 13 -> 14 -> NULL
```

Ques 6: Write a program to implement stack data structure along with its operations using array.

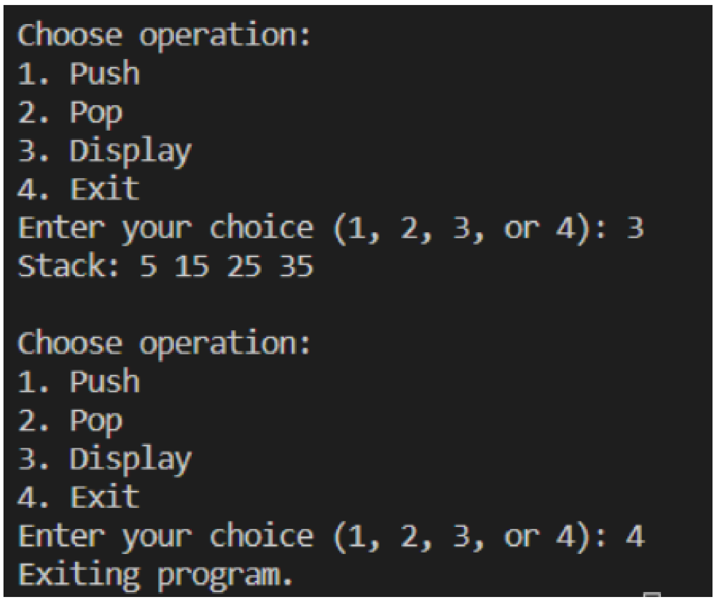
```c
#include <stdio.h>

#define MAX_SIZE 100

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

void initializeStack(struct Stack *stack) {
    stack->top = -1;
}

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

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

```c
void push(struct Stack *stack, int element) {
    if (isFull(stack)) {
        printf("Stack is full. Cannot push.\n");
    } else {
        stack->top++;
        stack->items[stack->top] = element;
        printf("Pushed: %d\n", element);
    }
}

void pop(struct Stack *stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty. Cannot pop.\n");
    } else {
        printf("Popped: %d\n", stack->items[stack->top]);
        stack->top--;
    }
}

void displayStack(struct Stack *stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty.\n");
    } else {
        printf("Stack: ");
        for (int i = 0; i <= stack->top; i++) {
            printf("%d ", stack->items[i]);
        }
        printf("\n");
    }
}

int main() {
    struct Stack stack;
    initializeStack(&stack);

    int n, element;

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

    printf("Enter %d elements for the stack:\n", n);
    for (int i = 0; i < n; i++) {
        printf("Element %d: ", i + 1);
        scanf("%d", &element);
        push(&stack, element);
    }

    int choice;

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

        switch (choice) {
            case 1:
                printf("Enter the element to push: ");
                scanf("%d", &element);
                push(&stack, element);
                displayStack(&stack);
                break;

            case 2:
                pop(&stack);
                displayStack(&stack);
                break;

            case 3:
                displayStack(&stack);
                break;

            case 4:
                printf("Exiting program.\n");
                break;

            default:
                printf("Invalid choice. Please enter 1, 2, 3, or 4.\n");
        }
    } while (choice != 4);

    return 0;
}
```

OUTPUT:



```
Enter the number of elements for the stack: 5
Enter 5 elements for the stack:
Element 1: 5
Pushed: 5
Element 2: 15
Pushed: 15
Element 3: 25
Pushed: 25
Element 4: 35
Pushed: 35
Element 5: 45
Pushed: 45

Choose operation:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice (1, 2, 3, or 4): 2
Popped: 45
Stack: 5 15 25 35
```



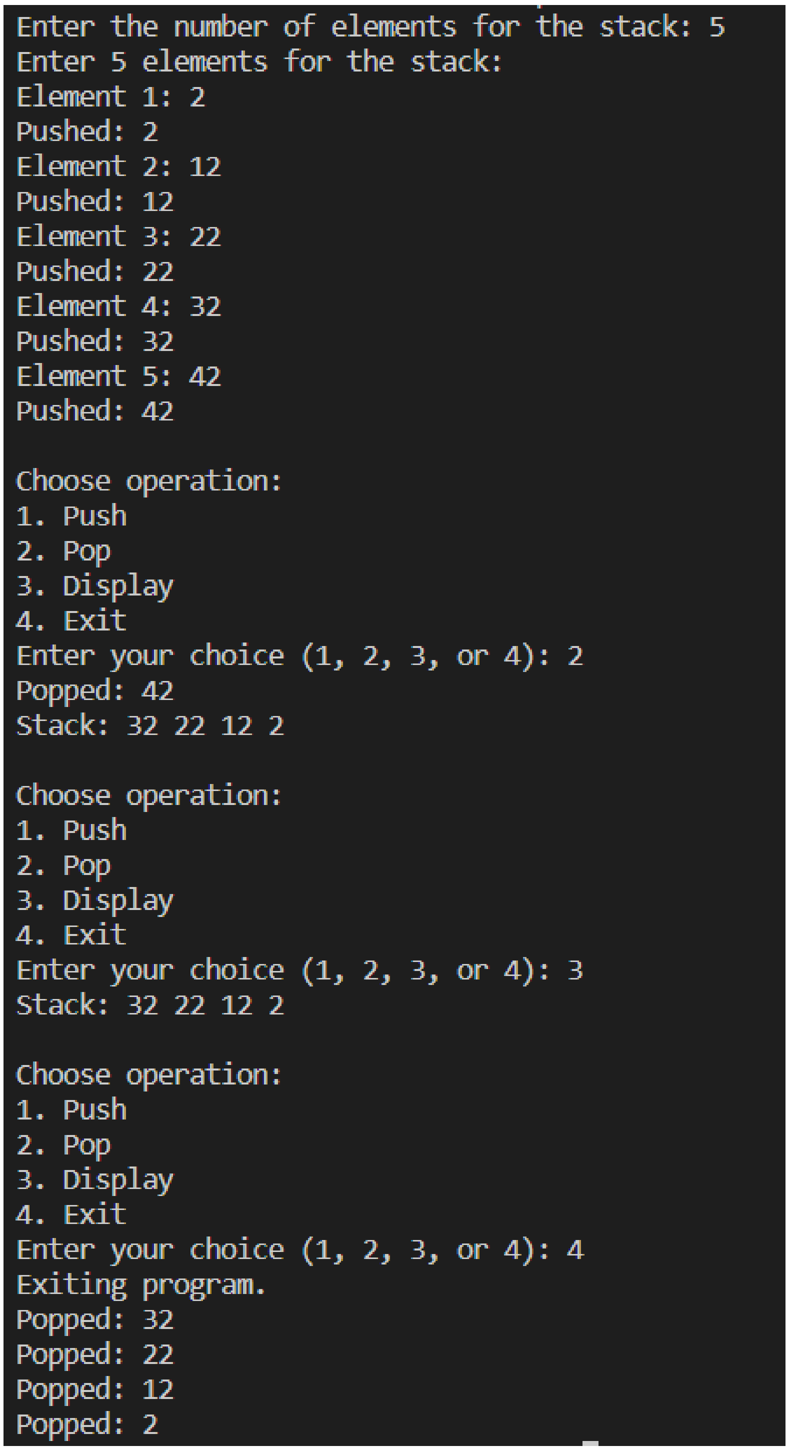
```
Choose operation:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice (1, 2, 3, or 4): 3
Stack: 5 15 25 35

Choose operation:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice (1, 2, 3, or 4): 4
Exiting program.
```

Ques 7: Write a program to implement stack data structure along with its operations using linked list.

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

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

struct Stack {
    struct Node *top;
};

void initializeStack(struct Stack *stack) {
    stack->top = NULL;
}

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

void push(struct Stack *stack, int element) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed.\n");
        exit(EXIT_FAILURE);
    }
    newNode->data = element;
    newNode->next = stack->top;
    stack->top = newNode;
    printf("Pushed: %d\n", element);
}

void pop(struct Stack *stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty. Cannot pop.\n");
    } else {
        struct Node *temp = stack->top;
        printf("Popped: %d\n", temp->data);
        stack->top = temp->next;
        free(temp);
    }
}

void displayStack(struct Stack *stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty.\n");
    } else {
        struct Node *current = stack->top;
        printf("Stack: ");
        while (current != NULL) {
            printf("%d ", current->data);
            current = current->next;
        }
```

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

int main() {
    struct Stack stack;
    initializeStack(&stack);

    int n, element;

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

    printf("Enter %d elements for the stack:\n", n);
    for (int i = 0; i < n; i++) {
        printf("Element %d: ", i + 1);
        scanf("%d", &element);
        push(&stack, element);
    }

    int choice;

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

        switch (choice) {
            case 1:
                printf("Enter the element to push: ");
                scanf("%d", &element);
                push(&stack, element);
                displayStack(&stack);
                break;

            case 2:
                pop(&stack);
                displayStack(&stack);
                break;

            case 3:
                displayStack(&stack);
                break;

            case 4:
                printf("Exiting program.\n");
                break;

            default:
                printf("Invalid choice. Please enter 1, 2, 3, or 4.\n");
        }
    } while (choice != 4);
    while (!isEmpty(&stack)) {
        pop(&stack);
    }

    return 0;
}
```

OUTPUT:



```
Enter the number of elements for the stack: 5
Enter 5 elements for the stack:
Element 1: 2
Pushed: 2
Element 2: 12
Pushed: 12
Element 3: 22
Pushed: 22
Element 4: 32
Pushed: 32
Element 5: 42
Pushed: 42

Choose operation:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice (1, 2, 3, or 4): 2
Popped: 42
Stack: 32 22 12 2

Choose operation:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice (1, 2, 3, or 4): 3
Stack: 32 22 12 2

Choose operation:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice (1, 2, 3, or 4): 4
Exiting program.
Popped: 32
Popped: 22
Popped: 12
Popped: 2
```

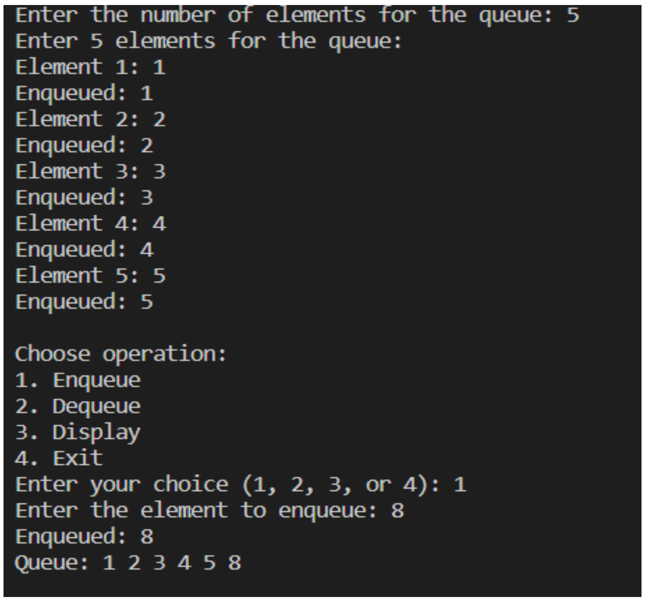
Ques 8 : Write a program to implement queue data structure along with its operations like enqueue, dequeue, display etc. using linked list.

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

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

struct Queue {
    struct Node *front;
    struct Node *rear;
};

void initializeQueue(struct Queue *queue) {
    queue->front = NULL;
    queue->rear = NULL;
}

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

void enqueue(struct Queue *queue, int element) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    newNode->data = element;
    newNode->next = NULL;

    if (isEmpty(queue)) {
        queue->front = newNode;
        queue->rear = newNode;
    } else {
        queue->rear->next = newNode;
        queue->rear = newNode;
    }

    printf("Enqueued: %d\n", element);
}

void dequeue(struct Queue *queue) {
    if (isEmpty(queue)) {
        printf("Queue is empty. Cannot dequeue.\n");
    } else {
        struct Node *temp = queue->front;
        printf("Dequeued: %d\n", temp->data);

        queue->front = temp->next;
        free(temp);

        if (queue->front == NULL) {
            queue->rear = NULL;
        }
    }
}
```

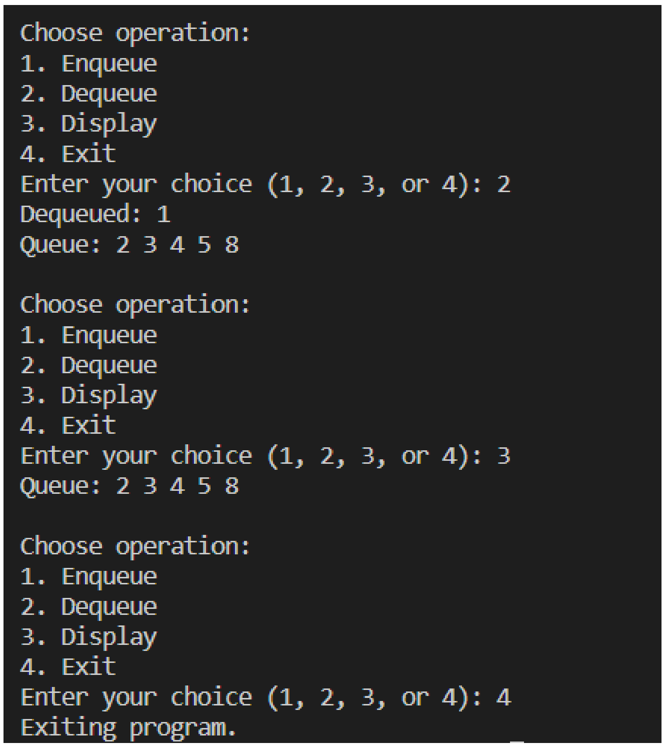
```
void displayQueue(struct Queue *queue) {
    if (isEmpty(queue)) {
        printf("Queue is empty.\n");
    } else {
        struct Node *current = queue->front;
        printf("Queue: ");
        while (current != NULL) {
            printf("%d ", current->data);
            current = current->next;
        }
        printf("\n");
    }
}

int main() {
    struct Queue queue;
    initializeQueue(&queue);

    int n, element;

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

    printf("Enter %d elements for the queue:\n", n);
    for (int i = 0; i < n; i++) {
        printf("Element %d: ", i + 1);
        scanf("%d", &element);
        enqueue(&queue, element);
    }

    int choice;

    do {
        printf("\nChoose operation:\n");
        printf("1. Enqueue\n");
        printf("2. Dequeue\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice (1, 2, 3, or 4): ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the element to enqueue: ");
                scanf("%d", &element);
                enqueue(&queue, element);
                displayQueue(&queue);
                break;

            case 2:
                dequeue(&queue);
                displayQueue(&queue);
                break;

            case 3:
                displayQueue(&queue);
```

```c
                break;

            case 4:
                printf("Exiting program.\n");
                break;

            default:
                printf("Invalid choice. Please enter 1, 2, 3, or 4.\n");
        }
    } while (choice != 4);

    return 0;
}
```

OUTPUT:



```
Enter the number of elements for the queue: 5
Enter 5 elements for the queue:
Element 1: 1
Enqueued: 1
Element 2: 2
Enqueued: 2
Element 3: 3
Enqueued: 3
Element 4: 4
Enqueued: 4
Element 5: 5
Enqueued: 5

Choose operation:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice (1, 2, 3, or 4): 1
Enter the element to enqueue: 8
Enqueued: 8
Queue: 1 2 3 4 5 8
```



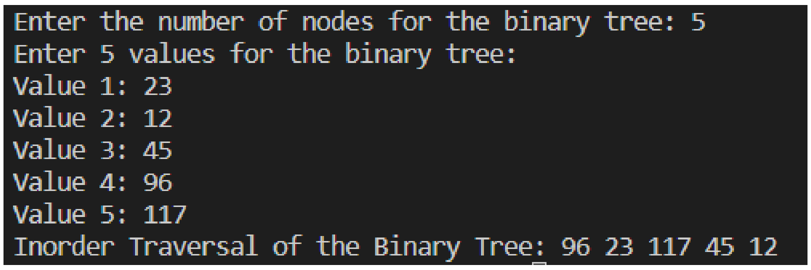
```
Choose operation:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice (1, 2, 3, or 4): 2
Dequeued: 1
Queue: 2 3 4 5 8

Choose operation:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice (1, 2, 3, or 4): 3
Queue: 2 3 4 5 8

Choose operation:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice (1, 2, 3, or 4): 4
Exiting program.
```

Ques 9: Write a program to implement Simple Binary Tree.

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

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

struct Node *createNode(int value) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed.\n");
        exit(EXIT_FAILURE);
    }
    newNode->data = value;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

void insert(struct Node **root, int value) {
    if (*root == NULL) {
        *root = createNode(value);
    } else {
        // Insert at random direction, either left or right
        if (rand() % 2 == 0) {
            insert(&((*root)->left), value);
        } else {
            insert(&((*root)->right), value);
        }
    }
}

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

void freeTree(struct Node *root) {
    if (root != NULL) {
        freeTree(root->left);
        freeTree(root->right);
        free(root);
    }
}

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

    printf("Enter the number of nodes for the binary tree: ");
```

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

    printf("Enter %d values for the binary tree:\n", n);
    for (int i = 0; i < n; i++) {
        printf("Value %d: ", i + 1);
        scanf("%d", &value);
        insert(&root, value);
    }

    printf("Inorder Traversal of the Binary Tree: ");
    inorderTraversal(root);
    printf("\n");

    freeTree(root);

    return 0;
}
```

OUTPUT:

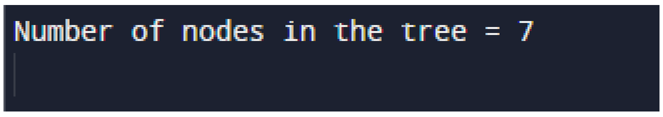


```
Enter the number of nodes for the binary tree: 5
Enter 5 values for the binary tree:
Value 1: 23
Value 2: 12
Value 3: 45
Value 4: 96
Value 5: 117
Inorder Traversal of the Binary Tree: 96 23 117 45 12
```

Ques 10: Write a program to count number of nodes present in given binary tree.

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

struct node{
    int info;
    struct node *left, *right;
};

struct node *createnode(int key){
    struct node *newnode = (struct node*)malloc(sizeof(struct node));
    newnode->info = key;
    newnode->left = NULL;
    newnode->right = NULL;
    return(newnode);
}


int countnode(struct node *root){
    if(root == NULL){
        return 0;
    }
    return 1+countnode(root->left)+countnode(root->right);
}

int main(){
    struct node *root = createnode(25);
    root->left = createnode(27);
    root->right = createnode(19);
    root->left->left = createnode(17);
    root->left->right = createnode(91);
    root->right->left = createnode(13);
    root->right->right = createnode(55);
    /* Sample Tree
     *                25
     *             /    \
     *            27     19
     *           / \     / \
     *         17   91   13 55
     */
    printf("Number of nodes in the tree = %d ",countnode(root));
    printf("\n");

}
```

OUTPUT:



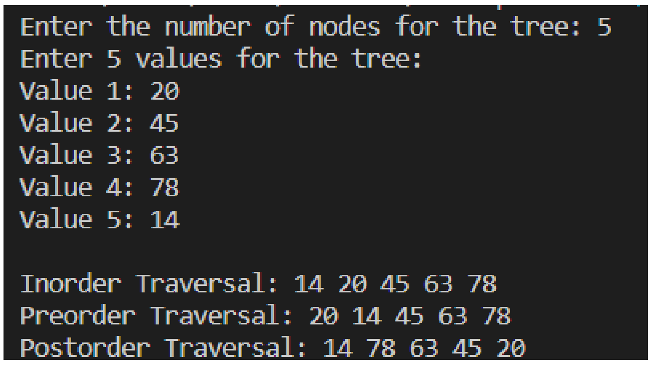
```
Number of nodes in the tree = 7
```

Ques 11: Write a program to implement tree traversals like inorder, preorder and postorder traversals.

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

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

struct Node *createNode(int value) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed.\n");
        exit(EXIT_FAILURE);
    }
    newNode->data = value;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

struct Node *insert(struct Node *root, int value) {
    if (root == NULL) {
        return createNode(value);
    }

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

```c
        root->right = insert(root->right, value);
    }

    return root;
}

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

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

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

void freeTree(struct Node *root) {
    if (root != NULL) {
        freeTree(root->left);
        freeTree(root->right);
        free(root);
    }
}

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

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

    printf("Enter %d values for the tree:\n", n);
    for (int i = 0; i < n; i++) {
        printf("Value %d: ", i + 1);
        scanf("%d", &value);
        root = insert(root, value);
    }

    printf("\nInorder Traversal: ");
    inorderTraversal(root);

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

    freeTree(root);

    return 0;
}
```

OUTPUT:



```
Enter the number of nodes for the tree: 5
Enter 5 values for the tree:
Value 1: 20
Value 2: 45
Value 3: 63
Value 4: 78
Value 5: 14

Inorder Traversal: 14 20 45 63 78
Preorder Traversal: 20 14 45 63 78
Postorder Traversal: 14 78 63 45 20
```

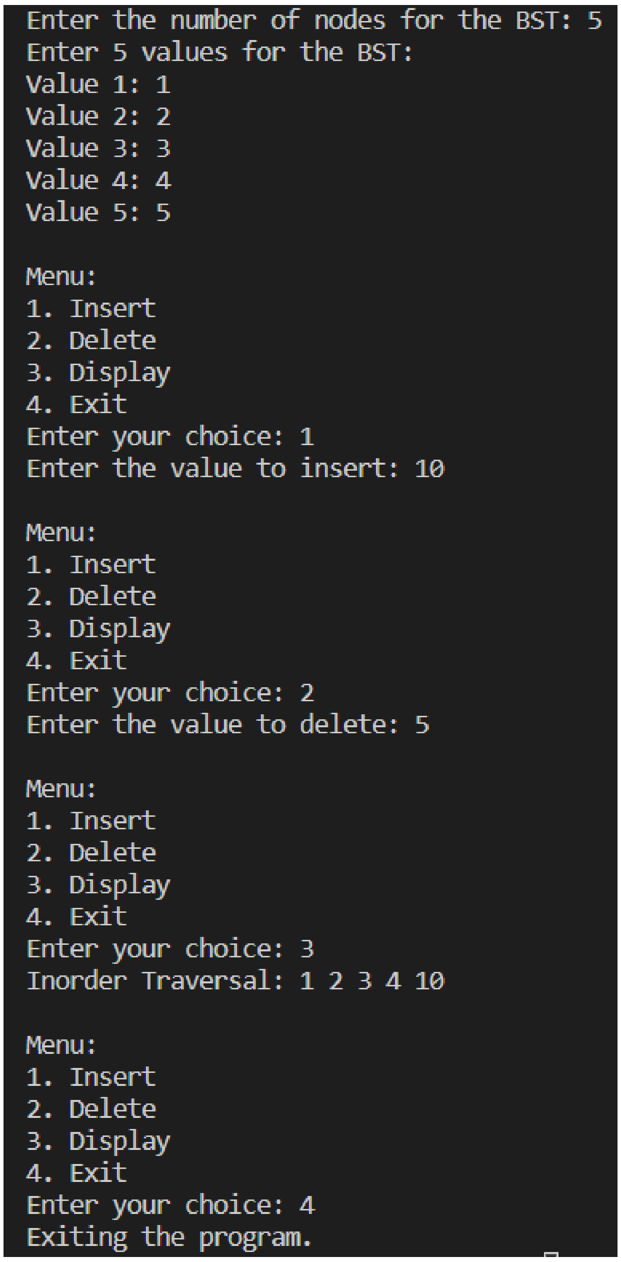
Ques 12: Write a program to implement Binary Search Tree along with its operations like insertion, deletion, and display.

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

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

struct Node *createNode(int value) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed.\n");
        exit(EXIT_FAILURE);
    }
    newNode->data = value;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

struct Node *insert(struct Node *root, int value) {
    if (root == NULL) {
        return createNode(value);
    }

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

```c
    }

    return root;
}

struct Node *findMin(struct Node *node) {
    while (node->left != NULL) {
        node = node->left;
    }
    return node;
}

struct Node *deleteNode(struct Node *root, int value) {
    if (root == NULL) {
        return root;
    }

    if (value < root->data) {
        root->left = deleteNode(root->left, value);
    } else if (value > root->data) {
        root->right = deleteNode(root->right, value);
    } else {
        // Node with one child or no child
        if (root->left == NULL) {
            struct Node *temp = root->right;
            free(root);
            return temp;
        } else if (root->right == NULL) {
            struct Node *temp = root->left;
            free(root);
            return temp;
        }

        // Node with two children
        struct Node *temp = findMin(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }
    return root;
}

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

void freeTree(struct Node *root) {
    if (root != NULL) {
        freeTree(root->left);
        freeTree(root->right);
        free(root);
    }
}
```

```c
void displayMenu() {
    printf("\nMenu:\n");
    printf("1. Insert\n");
    printf("2. Delete\n");
    printf("3. Display\n");
    printf("4. Exit\n");
}

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

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

    printf("Enter %d values for the BST:\n", n);
    for (int i = 0; i < n; i++) {
        printf("Value %d: ", i + 1);
        scanf("%d", &value);
        root = insert(root, value);
    }

    do {
        displayMenu();
        printf("Enter your choice: ");
        scanf("%d", &choice);

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

            case 2:
                printf("Enter the value to delete: ");
                scanf("%d", &value);
                root = deleteNode(root, value);
                break;

            case 3:
                printf("Inorder Traversal: ");
                inorderTraversal(root);
                printf("\n");
                break;

            case 4:
                printf("Exiting the program.\n");
                break;

            default:
                printf("Invalid choice. Please enter a valid option.\n");
        }

    } while (choice != 4);

    freeTree(root);
    return 0;
}
```

OUTPUT:

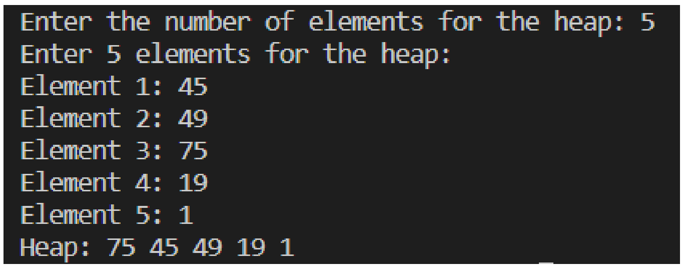


```
Enter the number of nodes for the BST: 5
Enter 5 values for the BST:
Value 1: 1
Value 2: 2
Value 3: 3
Value 4: 4
Value 5: 5

Menu:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter the value to insert: 10

Menu:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 2
Enter the value to delete: 5

Menu:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
Inorder Traversal: 1 2 3 4 10

Menu:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 4
Exiting the program.
```

Ques 13: Write a program to implement heap data structure using array.

```c
#include <stdio.h>

#define MAX_HEAP_SIZE 100

struct Heap {
    int arr[MAX_HEAP_SIZE];
    int size;
};

void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

void heapify(struct Heap *heap, int index) {
    int largest = index;
    int leftChild = 2 * index + 1;
    int rightChild = 2 * index + 2;

    if (leftChild < heap->size && heap->arr[leftChild] > heap->arr[largest]) {
        largest = leftChild;
    }

    if (rightChild < heap->size && heap->arr[rightChild] > heap->arr[largest]) {
        largest = rightChild;
    }

    if (largest != index) {
        swap(&heap->arr[index], &heap->arr[largest]);
        heapify(heap, largest);
    }
}

void insert(struct Heap *heap, int value) {
    if (heap->size == MAX_HEAP_SIZE) {
        printf("Heap overflow: Cannot insert more elements.\n");
        return;
    }

    heap->arr[heap->size] = value;
    int currentIndex = heap->size;
    int parentIndex = (currentIndex - 1) / 2;

    while (currentIndex > 0 && heap->arr[currentIndex] > heap->arr[parentIndex]) {
        swap(&heap->arr[currentIndex], &heap->arr[parentIndex]);
        currentIndex = parentIndex;
        parentIndex = (currentIndex - 1) / 2;
    }

    heap->size++;
}

void displayHeap(struct Heap *heap) {
    printf("Heap: ");
    for (int i = 0; i < heap->size; i++) {
        printf("%d ", heap->arr[i]);
    }
    printf("\n");
}

int main() {
    struct Heap heap;
    heap.size = 0;

    int n, value;
    printf("Enter the number of elements for the heap: ");
    scanf("%d", &n);

    printf("Enter %d elements for the heap:\n", n);
    for (int i = 0; i < n; i++) {
        printf("Element %d: ", i + 1);
        scanf("%d", &value);
        insert(&heap, value);
    }

    displayHeap(&heap);

    return 0;
}
```

OUTPUT:



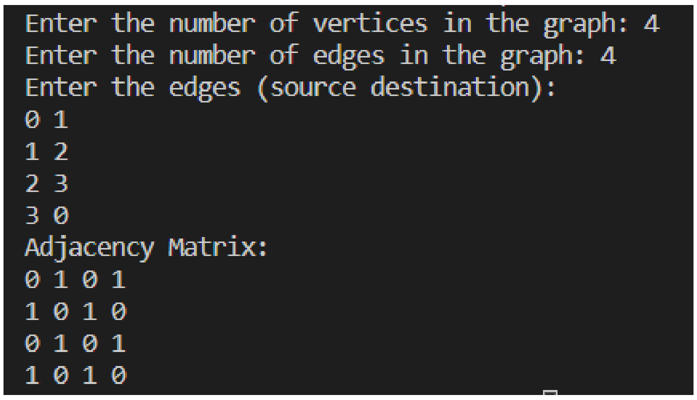
```
Enter the number of elements for the heap: 5
Enter 5 elements for the heap:
Element 1: 45
Element 2: 49
Element 3: 75
Element 4: 19
Element 5: 1
Heap: 75 45 49 19 1
```

Ques 14: Write a program to implement graph using adjacency matrix.

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

#define MAX_VERTICES 100

struct Graph {
    int vertices;
    int adjacencyMatrix[MAX_VERTICES][MAX_VERTICES];
};

void initializeGraph(struct Graph *graph, int vertices) {
    graph->vertices = vertices;

    for (int i = 0; i < vertices; i++) {
        for (int j = 0; j < vertices; j++) {
            graph->adjacencyMatrix[i][j] = 0;
        }
    }
}

void addEdge(struct Graph *graph, int source, int destination) {
    if (source >= 0 && source < graph->vertices && destination >= 0 && destination
< graph->vertices) {
        graph->adjacencyMatrix[source][destination] = 1;
        graph->adjacencyMatrix[destination][source] = 1;
    } else {
        printf("Invalid edge: (%d, %d)\n", source, destination);
```

```c
    }
}

void displayGraph(struct Graph *graph) {
    printf("Adjacency Matrix:\n");
    for (int i = 0; i < graph->vertices; i++) {
        for (int j = 0; j < graph->vertices; j++) {
            printf("%d ", graph->adjacencyMatrix[i][j]);
        }
        printf("\n");
    }
}

int main() {
    struct Graph graph;
    int vertices, edges, source, destination;

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

    initializeGraph(&graph, vertices);

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

    printf("Enter the edges (source destination):\n");
    for (int i = 0; i < edges; i++) {
        scanf("%d %d", &source, &destination);
        addEdge(&graph, source, destination);
    }

    displayGraph(&graph);
    return 0;
}
```

OUTPUT:

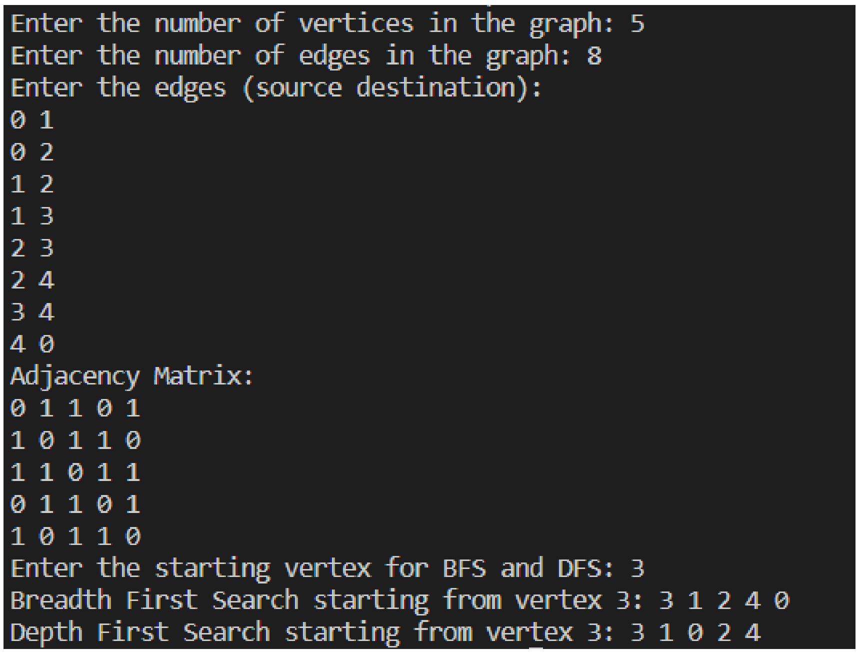


```
Enter the number of vertices in the graph: 4
Enter the number of edges in the graph: 4
Enter the edges (source destination):
0 1
1 2
2 3
3 0
Adjacency Matrix:
0 1 0 1
1 0 1 0
0 1 0 1
1 0 1 0
```

Ques 15: Write a program to implement Breadth First Search and Depth First Search using graph.

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

#define MAX_VERTICES 100
#define QUEUE_SIZE 100

struct Graph {
    int vertices;
    int adjacencyMatrix[MAX_VERTICES][MAX_VERTICES];
};

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

void initializeGraph(struct Graph *graph, int vertices) {
    graph->vertices = vertices;

    for (int i = 0; i < vertices; i++) {
        for (int j = 0; j < vertices; j++) {
            graph->adjacencyMatrix[i][j] = 0;
        }
    }
}

void addEdge(struct Graph *graph, int source, int destination) {
    if (source >= 0 && source < graph->vertices && destination >= 0 && destination
< graph->vertices) {
        graph->adjacencyMatrix[source][destination] = 1;
        graph->adjacencyMatrix[destination][source] = 1;
    } else {
        printf("Invalid edge: (%d, %d)\n", source, destination);
    }
}

void displayGraph(struct Graph *graph) {
    printf("Adjacency Matrix:\n");
    for (int i = 0; i < graph->vertices; i++) {
        for (int j = 0; j < graph->vertices; j++) {
            printf("%d ", graph->adjacencyMatrix[i][j]);
        }
        printf("\n");
    }
}

void enqueue(struct Queue *queue, int item) {
    if (queue->rear == QUEUE_SIZE - 1) {
        printf("Queue overflow\n");
        return;
    }

    if (queue->front == -1) {
        queue->front = 0;
    }

    queue->rear++;
    queue->items[queue->rear] = item;
}

int dequeue(struct Queue *queue) {
    if (queue->front == -1) {
        printf("Queue underflow\n");
        return -1;
    }

    int item = queue->items[queue->front];

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

    return item;
}

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

void breadthFirstSearch(struct Graph *graph, int startVertex) {
    struct Queue queue;
    queue.front = queue.rear = -1;

    int visited[MAX_VERTICES] = {0};
    printf("Breadth First Search starting from vertex %d: ", startVertex);

    visited[startVertex] = 1;
    printf("%d ", startVertex);
    enqueue(&queue, startVertex);

    while (!isEmpty(&queue)) {
        int currentVertex = dequeue(&queue);

        for (int i = 0; i < graph->vertices; i++) {
            if (graph->adjacencyMatrix[currentVertex][i] == 1 && !visited[i]) {
                visited[i] = 1;
                printf("%d ", i);
                enqueue(&queue, i);
            }
        }
    }

    printf("\n");
}

void depthFirstSearchUtil(struct Graph *graph, int vertex, int visited[]) {
    visited[vertex] = 1;
    printf("%d ", vertex);

    for (int i = 0; i < graph->vertices; i++) {
        if (graph->adjacencyMatrix[vertex][i] == 1 && !visited[i]) {
            depthFirstSearchUtil(graph, i, visited);
        }
    }
}

void depthFirstSearch(struct Graph *graph, int startVertex) {
    int visited[MAX_VERTICES] = {0};
    printf("Depth First Search starting from vertex %d: ", startVertex);

    depthFirstSearchUtil(graph, startVertex, visited);

    printf("\n");
}

int main() {
    struct Graph graph;
    int vertices, edges, source, destination, startVertex;

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

    initializeGraph(&graph, vertices);

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

    printf("Enter the edges (source destination):\n");
    for (int i = 0; i < edges; i++) {
        scanf("%d %d", &source, &destination);
        addEdge(&graph, source, destination);
    }

    displayGraph(&graph);

    printf("Enter the starting vertex for BFS and DFS: ");
    scanf("%d", &startVertex);

    breadthFirstSearch(&graph, startVertex);
    depthFirstSearch(&graph, startVertex);

    return 0;
}
```

OUTPUT:



```
Enter the number of vertices in the graph: 5
Enter the number of edges in the graph: 8
Enter the edges (source destination):
0 1
0 2
1 2
1 3
2 3
2 4
3 4
4 0
Adjacency Matrix:
0 1 1 0 1
1 0 1 1 0
1 1 0 1 1
0 1 1 0 1
1 0 1 1 0
Enter the starting vertex for BFS and DFS: 3
Breadth First Search starting from vertex 3: 3 1 2 4 0
Depth First Search starting from vertex 3: 3 1 0 2 4
```

THANK YOU