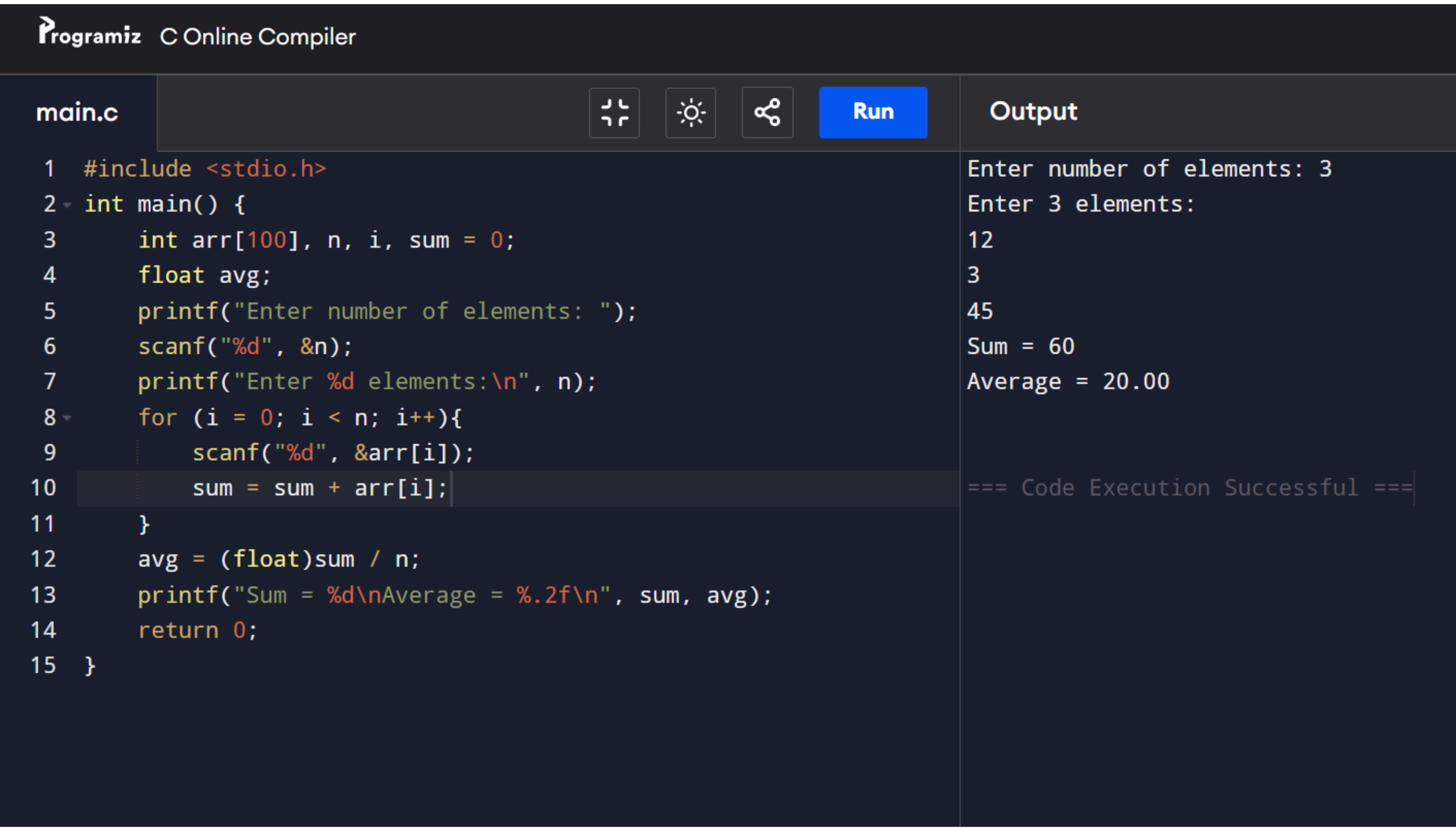
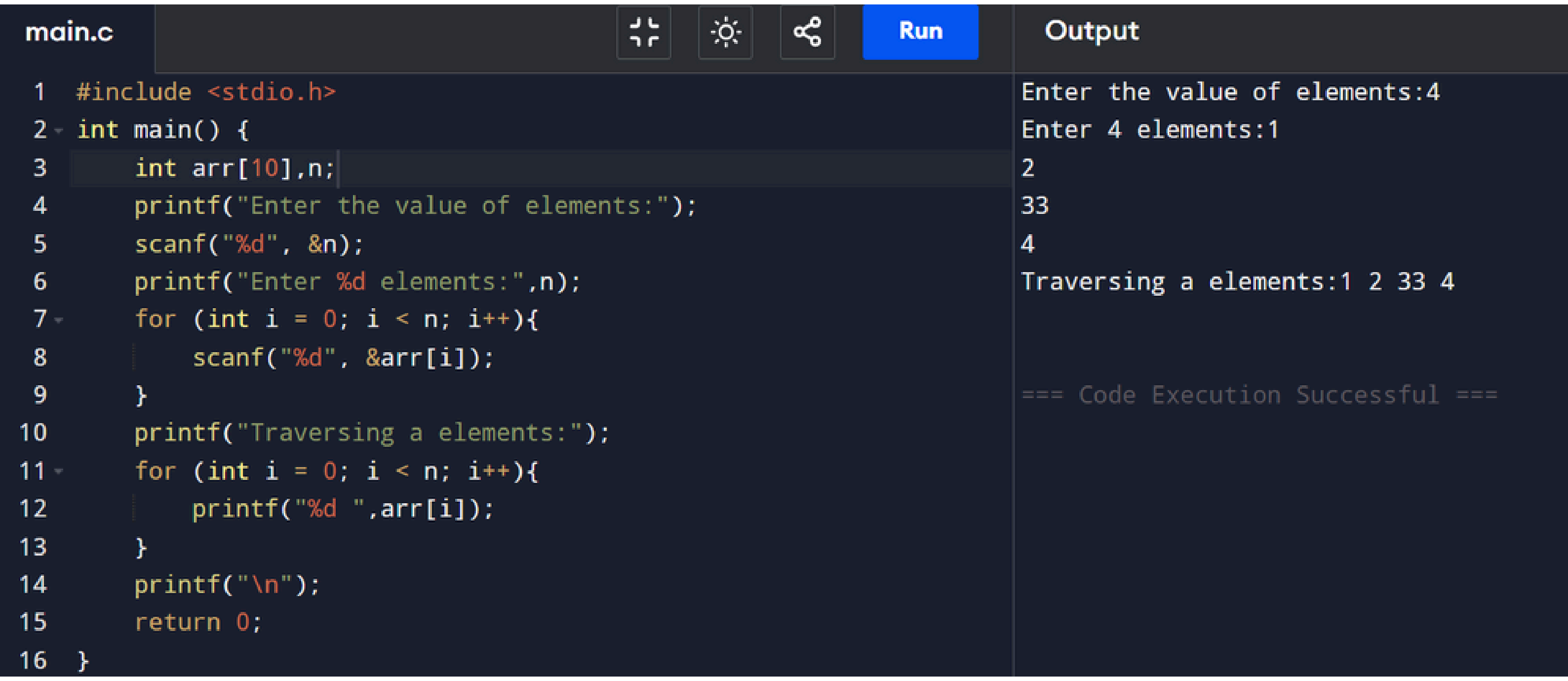
1. ARRAY AND SUM AND AVERAGE



```c
#include <stdio.h>
int main() {
    int arr[100], n, i, sum = 0;
    float avg;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    printf("Enter %d elements:\n", n);
    for (i = 0; i < n; i++){
        scanf("%d", &arr[i]);
        sum = sum + arr[i];
    }
    avg = (float)sum / n;
    printf("Sum = %d\nAverage = %.2f\n", sum, avg);
    return 0;
}
```

```
Enter number of elements: 3
Enter 3 elements:
12
3
45
Sum = 60
Average = 20.00


=== Code Execution Successful ===
```

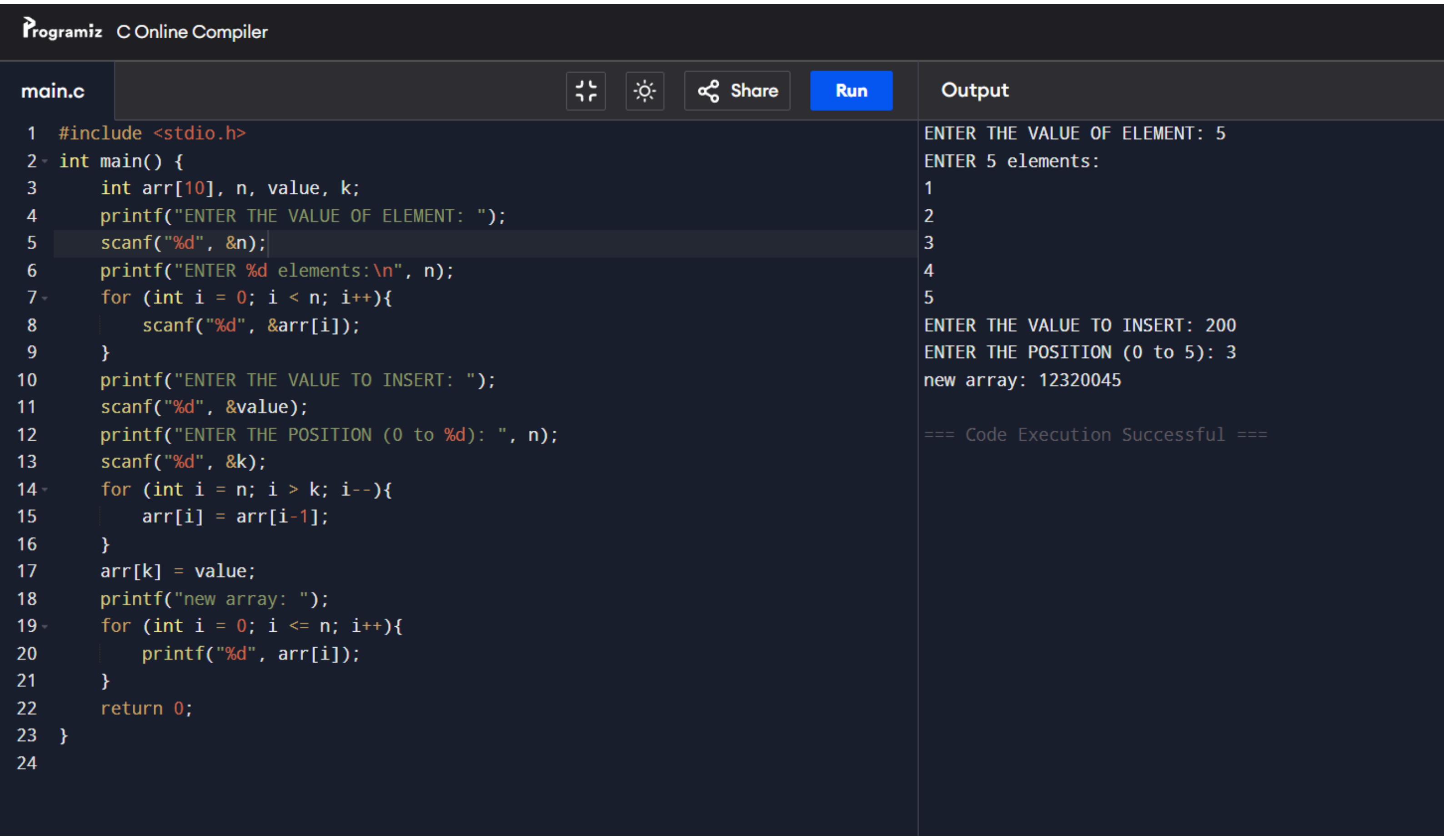
2.TRAVERSING IN ARRAY



```c
#include <stdio.h>
int main() {
    int arr[10],n;
    printf("Enter the value of elements:");
    scanf("%d", &n);
    printf("Enter %d elements:",n);
    for (int i = 0; i < n; i++){
        scanf("%d", &arr[i]);
    }
    printf("Traversing a elements:");
    for (int i = 0; i < n; i++){
        printf("%d ",arr[i]);
    }
    printf("\n");
    return 0;
}
```

```
Enter the value of elements:4
Enter 4 elements:1
2
33
4
Traversing a elements:1 2 33 4


=== Code Execution Successful ===
```

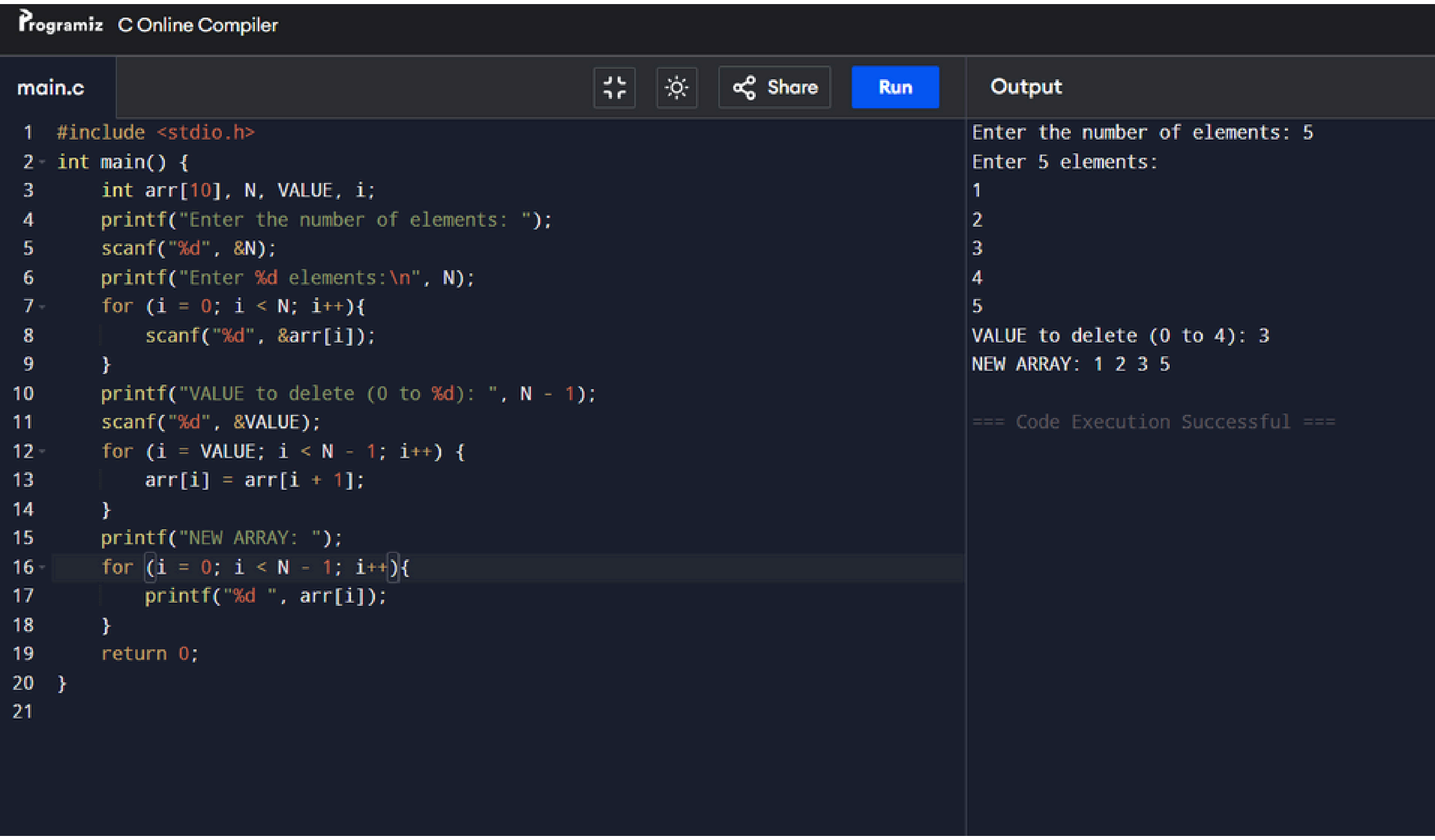
3.INSERTION IN ARRAY



```c
#include <stdio.h>
int main() {
    int arr[10], n, value, k;
    printf("ENTER THE VALUE OF ELEMENT: ");
    scanf("%d", &n);
    printf("ENTER %d elements:\n", n);
    for (int i = 0; i < n; i++){
        scanf("%d", &arr[i]);
    }
    printf("ENTER THE VALUE TO INSERT: ");
    scanf("%d", &value);
    printf("ENTER THE POSITION (0 to %d): ", n);
    scanf("%d", &k);
    for (int i = n; i > k; i--){
        arr[i] = arr[i-1];
    }
    arr[k] = value;
    printf("new array: ");
    for (int i = 0; i <= n; i++){
        printf("%d", arr[i]);
    }
    return 0;
}
```

```
ENTER THE VALUE OF ELEMENT: 5
ENTER 5 elements:
1
2
3
4
5
ENTER THE VALUE TO INSERT: 200
ENTER THE POSITION (0 to 5): 3
new array: 12320045

=== Code Execution Successful ===
```

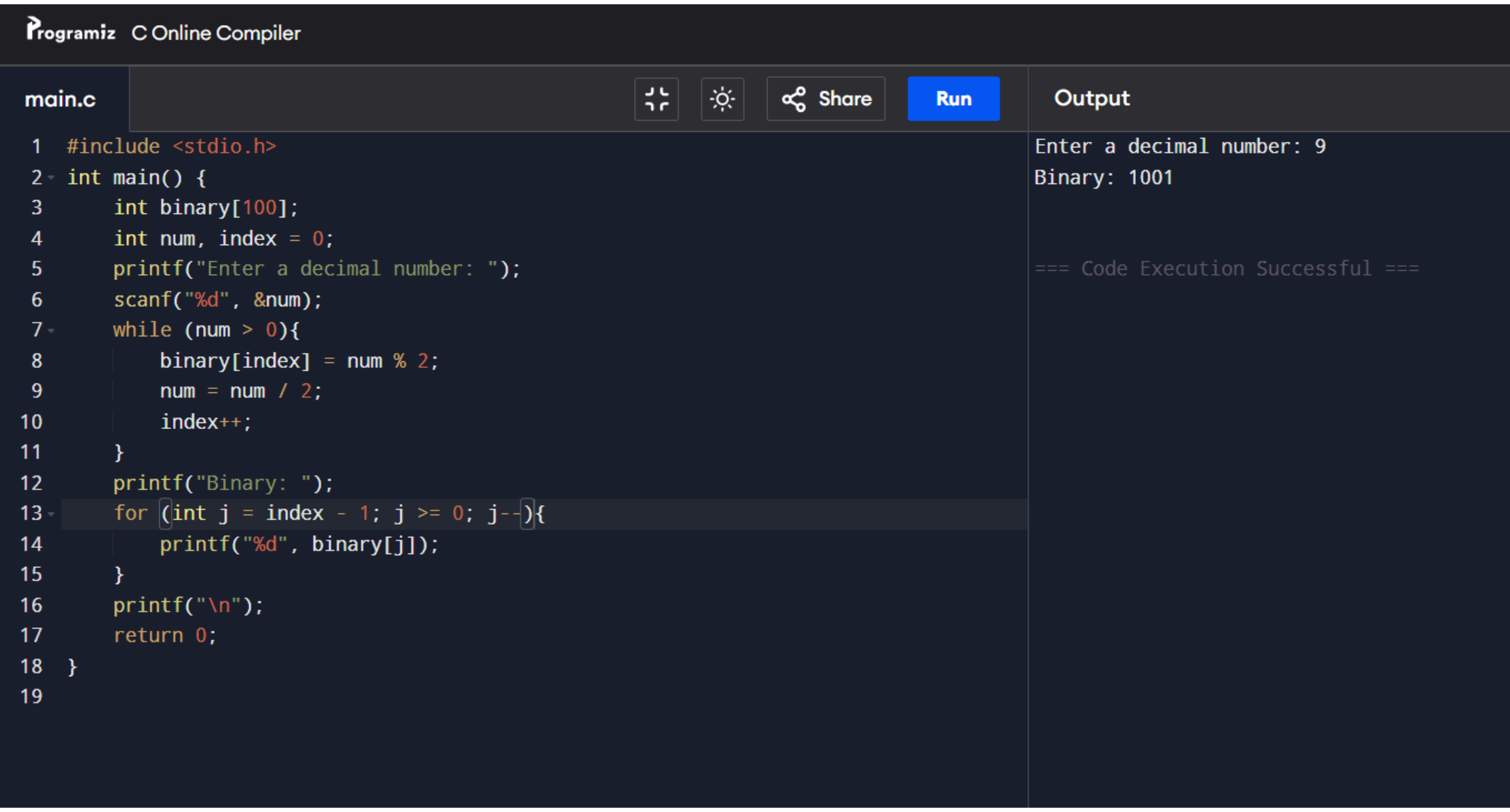
4.DELETION IN ARRAY



```c
#include <stdio.h>
int main() {
    int arr[10], N, VALUE, i;
    printf("Enter the number of elements: ");
    scanf("%d", &N);
    printf("Enter %d elements:\n", N);
    for (i = 0; i < N; i++){
        scanf("%d", &arr[i]);
    }
    printf("VALUE to delete (0 to %d): ", N - 1);
    scanf("%d", &VALUE);
    for (i = VALUE; i < N - 1; i++) {
        arr[i] = arr[i + 1];
    }
    printf("NEW ARRAY: ");
    for (i = 0; i < N - 1; i++){
        printf("%d ", arr[i]);
    }
    return 0;
}
```

```
Enter the number of elements: 5
Enter 5 elements:
1
2
3
4
5
VALUE to delete (0 to 4): 3
NEW ARRAY: 1 2 3 5

=== Code Execution Successful ===
```

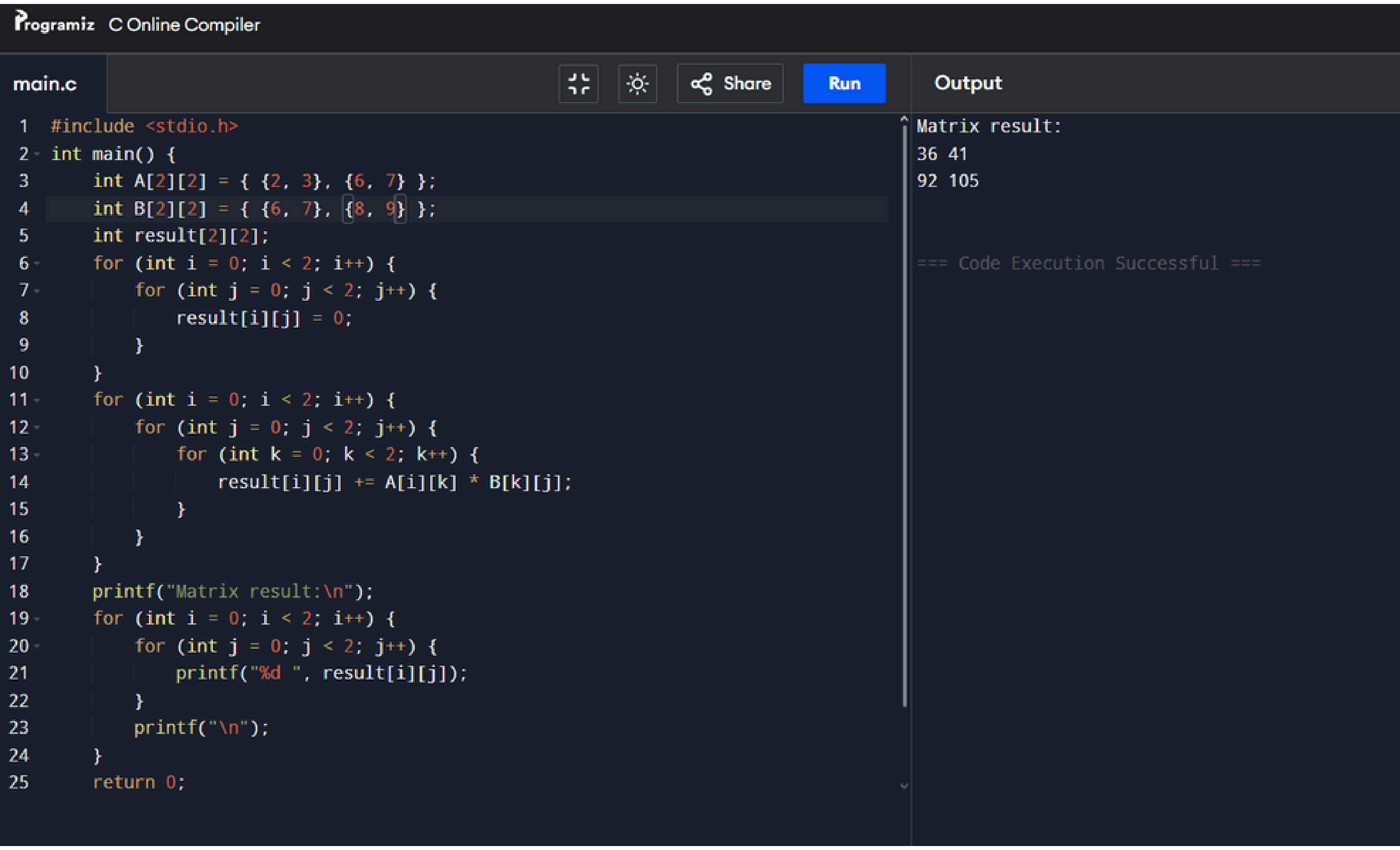
5.DECIMAL TO BINARY



```c
#include <stdio.h>
int main() {
    int binary[100];
    int num, index = 0;
    printf("Enter a decimal number: ");
    scanf("%d", &num);
    while (num > 0){
        binary[index] = num % 2;
        num = num / 2;
        index++;
    }
    printf("Binary: ");
    for (int j = index - 1; j >= 0; j--){
        printf("%d", binary[j]);
    }
    printf("\n");
    return 0;
}
```

```
Enter a decimal number: 9
Binary: 1001


=== Code Execution Successful ===
```

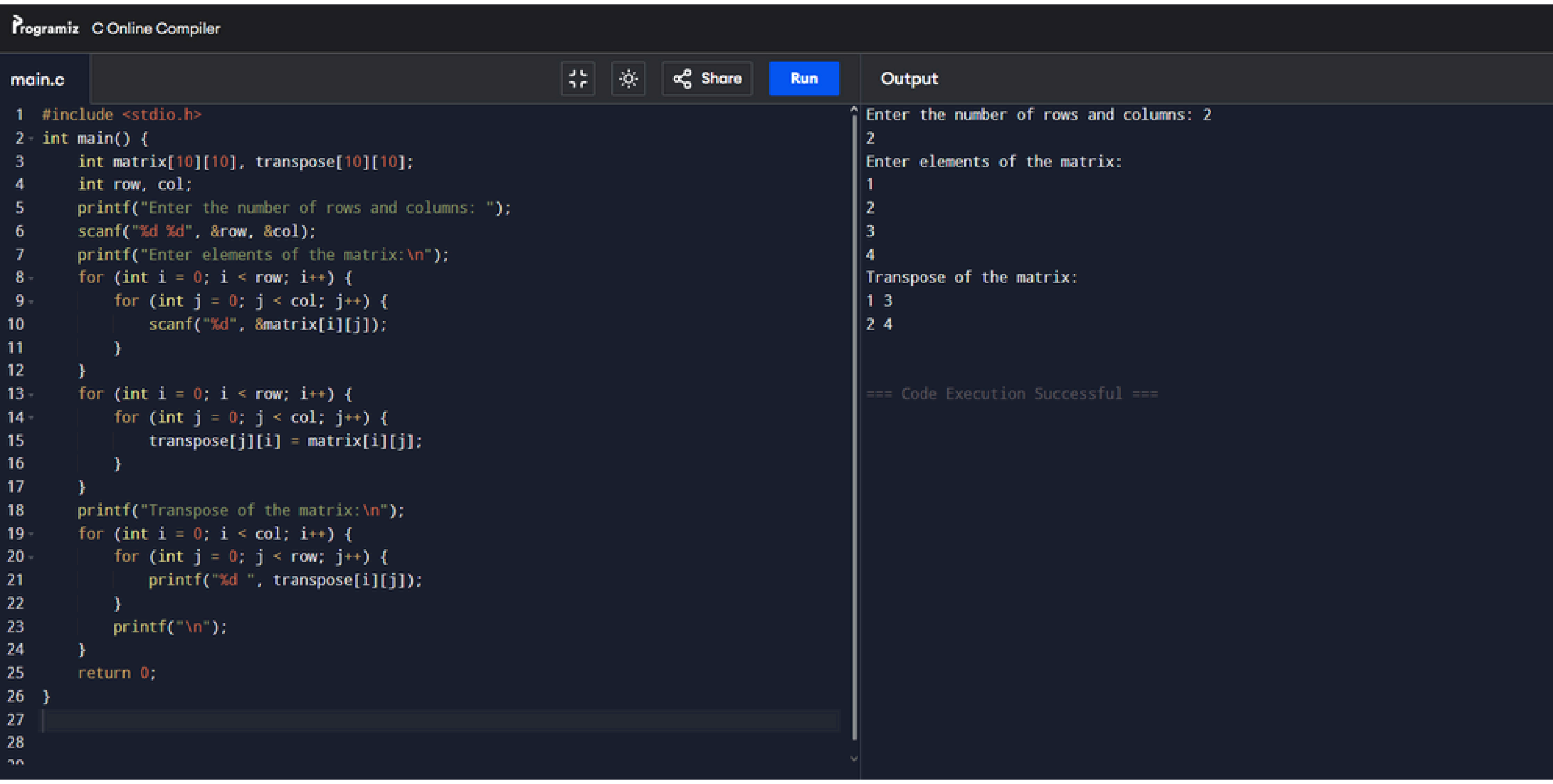
6.MULTIPLICATION IN ARRAY 2D



```c
#include <stdio.h>
int main() {
    int A[2][2] = { {2, 3}, {6, 7} };
    int B[2][2] = { {6, 7}, {8, 9} };
    int result[2][2];
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2; j++) {
            result[i][j] = 0;
        }
    }
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2; j++) {
            for (int k = 0; k < 2; k++) {
                result[i][j] += A[i][k] * B[k][j];
            }
        }
    }
    printf("Matrix result:\n");
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2; j++) {
            printf("%d ", result[i][j]);
        }
        printf("\n");
    }
    return 0;
```

```
Matrix result:
36 41
92 105


=== Code Execution Successful ===
```

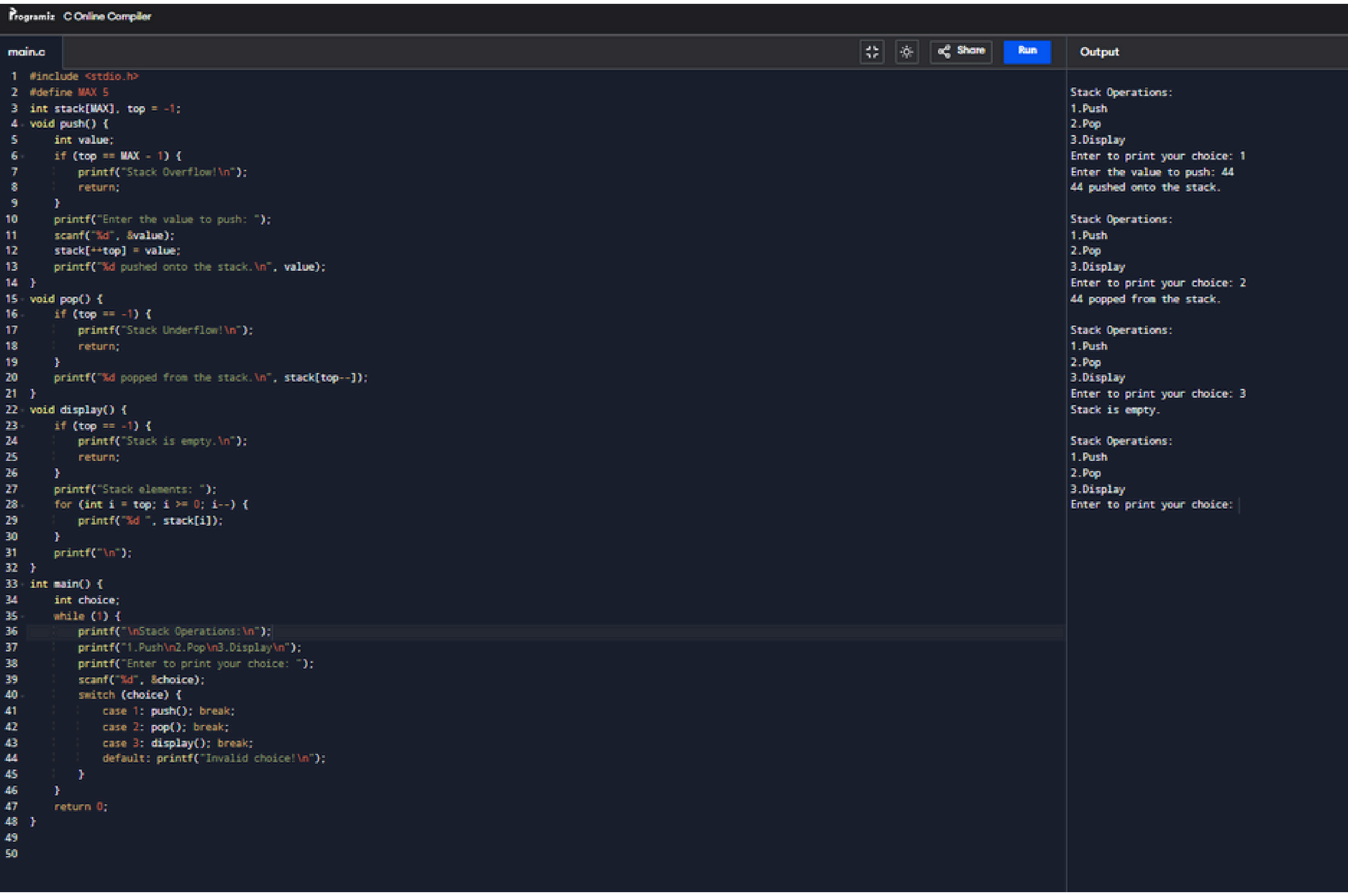
7.TRANSPOSING IN ARRAY



```c
#include <stdio.h>
int main() {
    int matrix[10][10], transpose[10][10];
    int row, col;
    printf("Enter the number of rows and columns: ");
    scanf("%d %d", &row, &col);
    printf("Enter elements of the matrix:\n");
    for (int i = 0; i < row; i++) {
        for (int j = 0; j < col; j++) {
            scanf("%d", &matrix[i][j]);
        }
    }
    for (int i = 0; i < row; i++) {
        for (int j = 0; j < col; j++) {
            transpose[j][i] = matrix[i][j];
        }
    }
    printf("Transpose of the matrix:\n");
    for (int i = 0; i < col; i++) {
        for (int j = 0; j < row; j++) {
            printf("%d ", transpose[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

```
Enter the number of rows and columns: 2
2
Enter elements of the matrix:
1
2
3
4
Transpose of the matrix:
1 3
2 4


=== Code Execution Successful ===
```

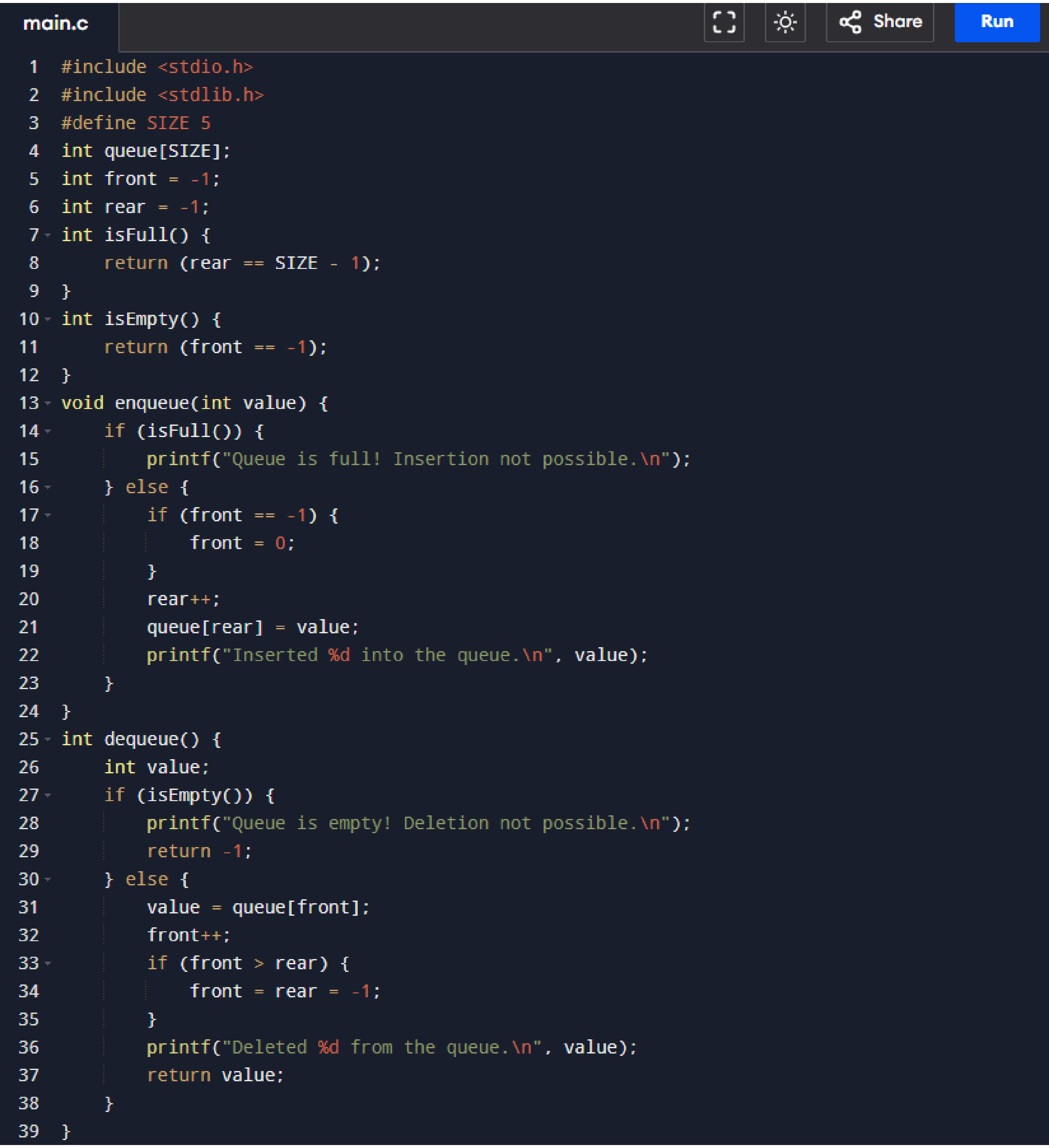
8.STACK OPERATIONS-(PUSH /POP/DISPLAY)



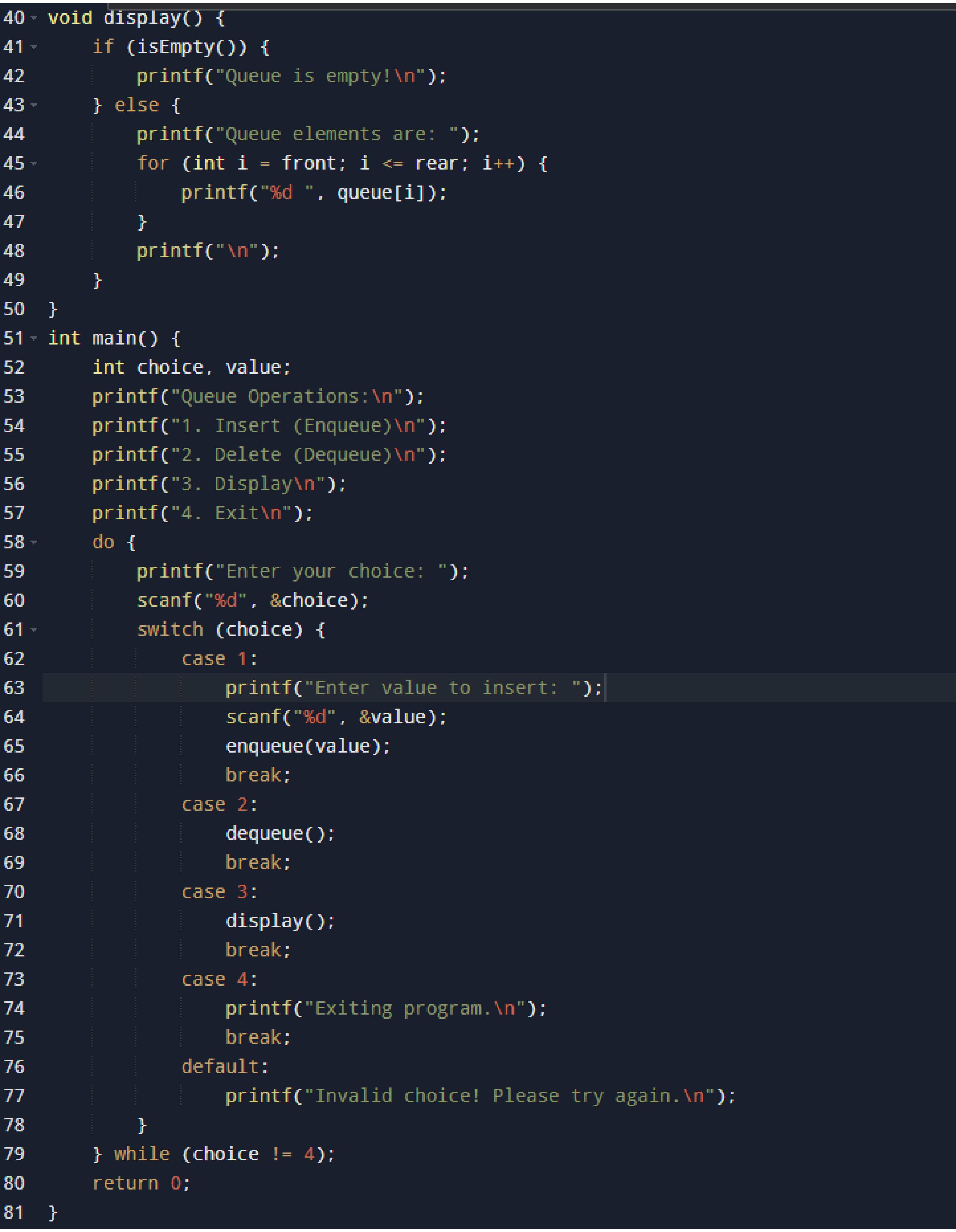
```c
#include <stdio.h>
#define MAX 5
int stack[MAX], top = -1;
void push() {
    int value;
    if (top == MAX - 1) {
        printf("Stack Overflow!\n");
        return;
    }
    printf("Enter the value to push: ");
    scanf("%d", &value);
    stack[++top] = value;
    printf("%d pushed onto the stack.\n", value);
}
void pop() {
    if (top == -1) {
        printf("Stack Underflow!\n");
        return;
    }
    printf("%d popped from the stack.\n", stack[top--]);
}
void display() {
    if (top == -1) {
        printf("Stack is empty.\n");
        return;
    }
    printf("Stack elements: ");
    for (int i = top; i >= 0; i--) {
        printf("%d ", stack[i]);
    }
    printf("\n");
}
int main() {
    int choice;
    while (1) {
        printf("\nStack Operations:\n");
        printf("1.Push\n2.Pop\n3.Display\n");
        printf("Enter to print your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1: push(); break;
            case 2: pop(); break;
            case 3: display(); break;
            default: printf("Invalid choice!\n");
        }
    }
    return 0;
}
```

```
Stack Operations:
1.Push
2.Pop
3.Display
Enter to print your choice: 1
Enter the value to push: 44
44 pushed onto the stack.

Stack Operations:
1.Push
2.Pop
3.Display
Enter to print your choice: 2
44 popped from the stack.

Stack Operations:
1.Push
2.Pop
3.Display
Enter to print your choice: 3
Stack is empty.

Stack Operations:
1.Push
2.Pop
3.Display
Enter to print your choice:
```

9.QUEUE OPERATION-(INSERTION,DELETION,DISPLAY)

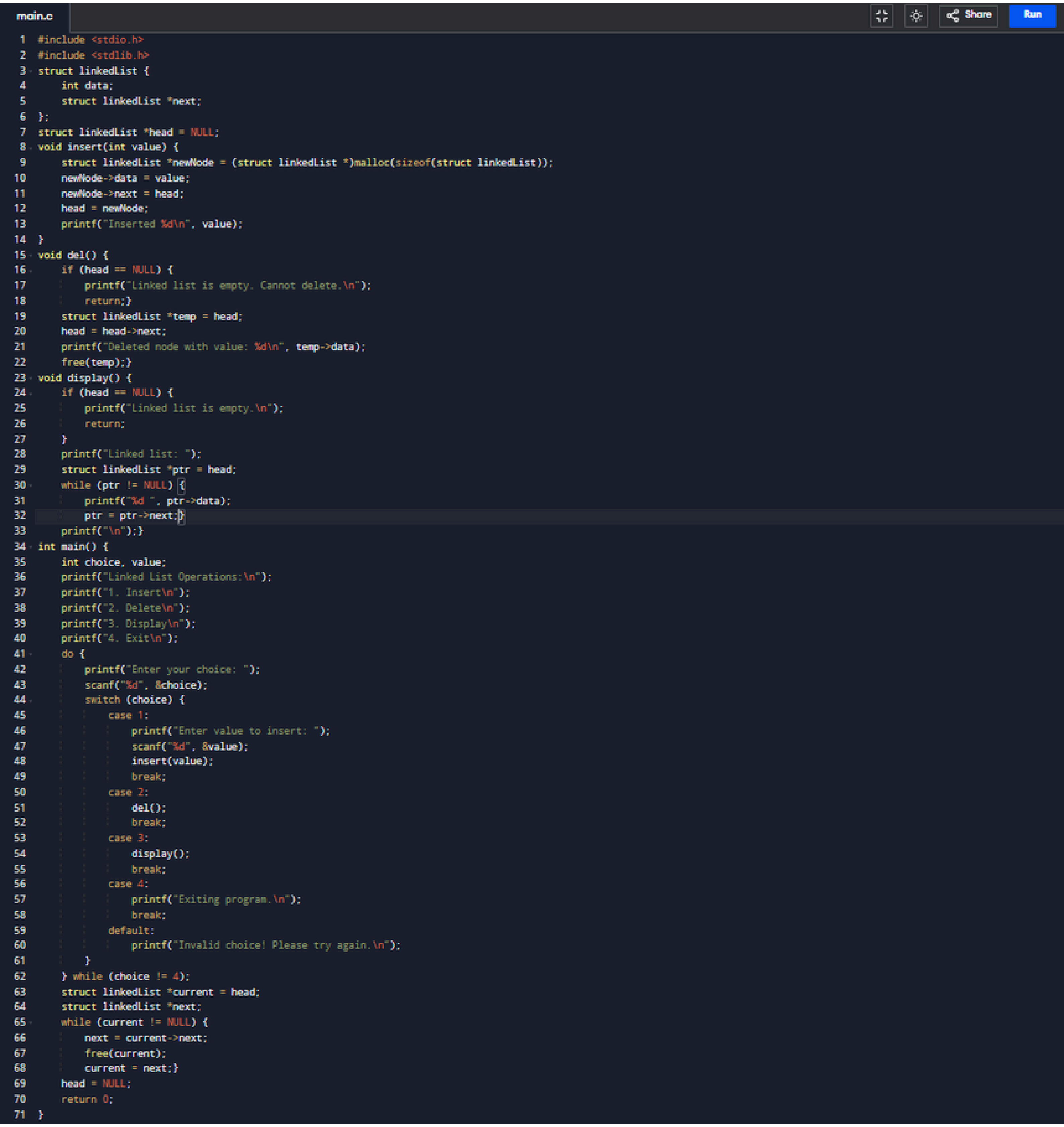


```c
#include <stdio.h>
#include <stdlib.h>
#define SIZE 5
int queue[SIZE];
int front = -1;
int rear = -1;
int isFull() {
    return (rear == SIZE - 1);
}
int isEmpty() {
    return (front == -1);
}
void enqueue(int value) {
    if (isFull()) {
        printf("Queue is full! Insertion not possible.\n");
    } else {
        if (front == -1) {
            front = 0;
        }
        rear++;
        queue[rear] = value;
        printf("Inserted %d into the queue.\n", value);
    }
}
int dequeue() {
    int value;
    if (isEmpty()) {
        printf("Queue is empty! Deletion not possible.\n");
        return -1;
    } else {
        value = queue[front];
        front++;
        if (front > rear) {
            front = rear = -1;
        }
        printf("Deleted %d from the queue.\n", value);
        return value;
    }
}
```



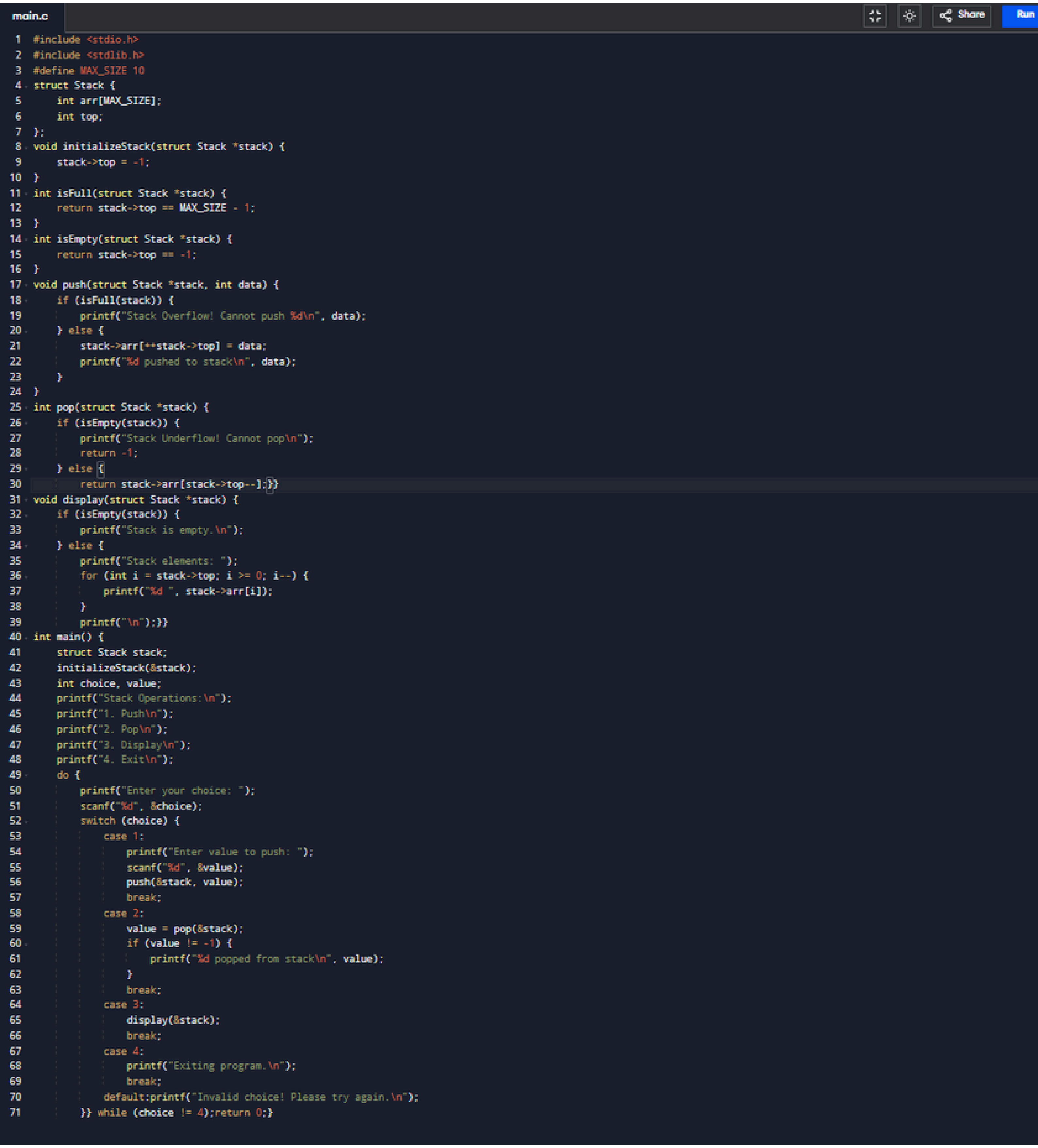
```c
void display() {
    if (isEmpty()) {
        printf("Queue is empty!\n");
    } else {
        printf("Queue elements are: ");
        for (int i = front; i <= rear; i++) {
            printf("%d ", queue[i]);
        }
        printf("\n");
    }
}
int main() {
    int choice, value;
    printf("Queue Operations:\n");
    printf("1. Insert (Enqueue)\n");
    printf("2. Delete (Dequeue)\n");
    printf("3. Display\n");
    printf("4. Exit\n");
    do {
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                enqueue(value);
                break;
            case 2:
                dequeue();
                break;
            case 3:
                display();
                break;
            case 4:
                printf("Exiting program.\n");
                break;
            default:
                printf("Invalid choice! Please try again.\n");
        }
    } while (choice != 4);
    return 0;
}
```

10.LINKED LIST OPERATION-(INSERTION,DELETION,DISPLAY)



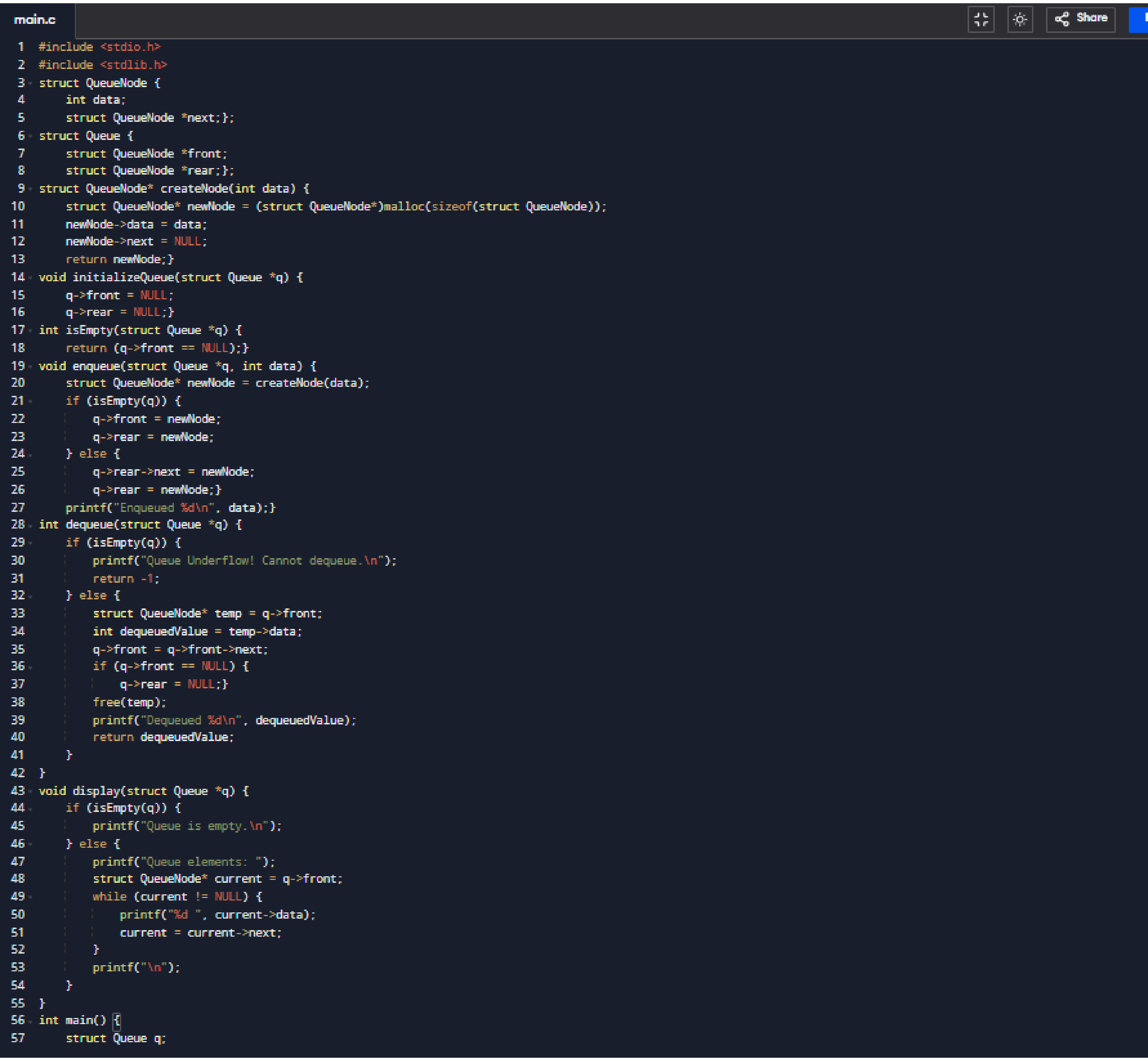
```c
#include <stdio.h>
#include <stdlib.h>
struct linkedList {
    int data;
    struct linkedList *next;
};
struct linkedList *head = NULL;
void insert(int value) {
    struct linkedList *newNode = (struct linkedList *)malloc(sizeof(struct linkedList));
    newNode->data = value;
    newNode->next = head;
    head = newNode;
    printf("Inserted %d\n", value);
}
void del() {
    if (head == NULL) {
        printf("Linked list is empty. Cannot delete.\n");
        return;}
    struct linkedList *temp = head;
    head = head->next;
    printf("Deleted node with value: %d\n", temp->data);
    free(temp);}
void display() {
    if (head == NULL) {
        printf("Linked list is empty.\n");
        return;
    }
    printf("Linked list: ");
    struct linkedList *ptr = head;
    while (ptr != NULL) {
        printf("%d ", ptr->data);
        ptr = ptr->next;}
    printf("\n");}
int main() {
    int choice, value;
    printf("Linked List Operations:\n");
    printf("1. Insert\n");
    printf("2. Delete\n");
    printf("3. Display\n");
    printf("4. Exit\n");
    do {
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                insert(value);
                break;
            case 2:
                del();
                break;
            case 3:
                display();
                break;
            case 4:
                printf("Exiting program.\n");
                break;
            default:
                printf("Invalid choice! Please try again.\n");
        }
    } while (choice != 4);
    struct linkedList *current = head;
    struct linkedList *next;
    while (current != NULL) {
        next = current->next;
        free(current);
        current = next;}
    head = NULL;
    return 0;
}
```

11.stack through linked list

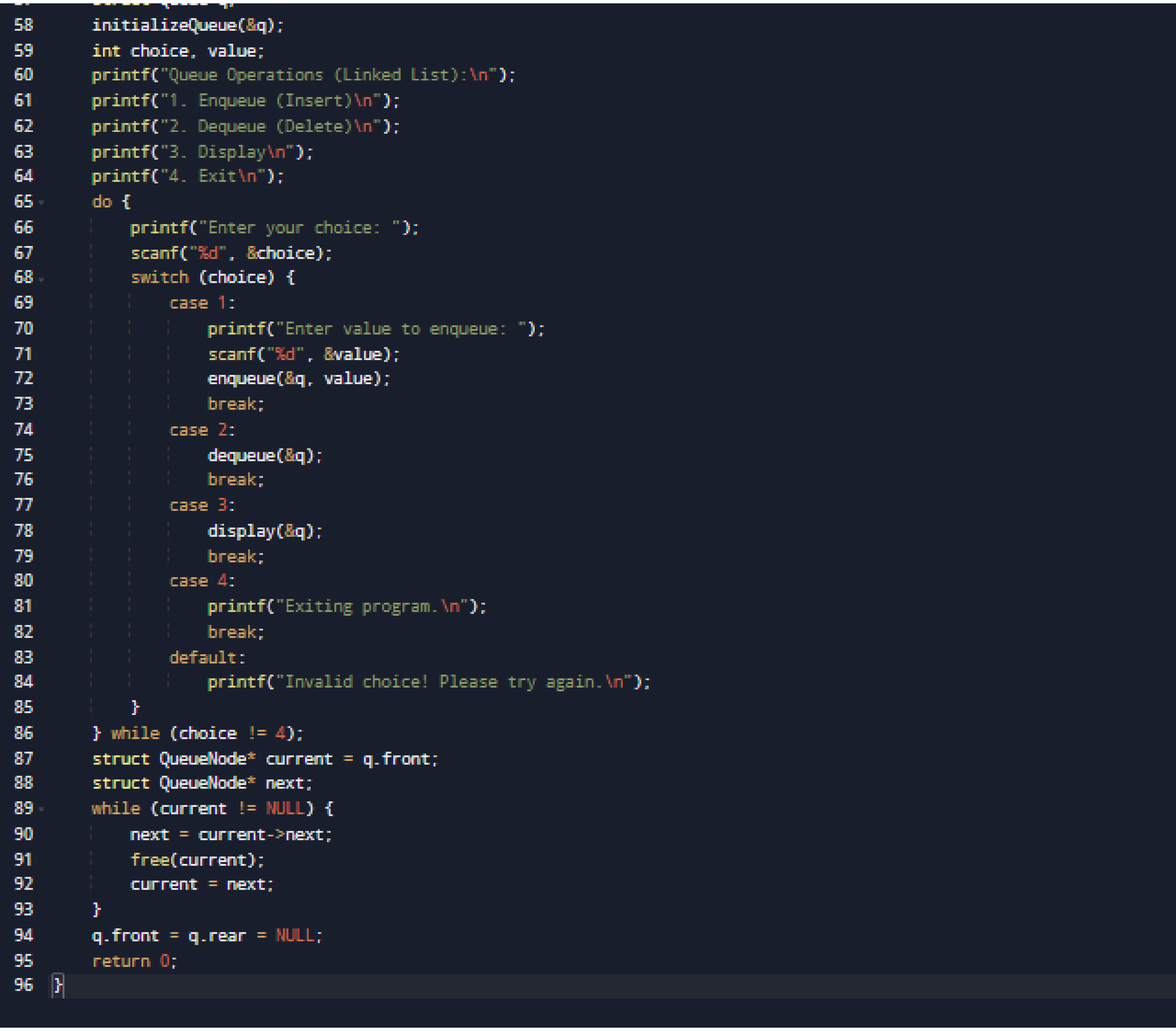


```c
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 10
struct Stack {
    int arr[MAX_SIZE];
    int top;
};
void initializeStack(struct Stack *stack) {
    stack->top = -1;
}
int isFull(struct Stack *stack) {
    return stack->top == MAX_SIZE - 1;
}
int isEmpty(struct Stack *stack) {
    return stack->top == -1;
}
void push(struct Stack *stack, int data) {
    if (isFull(stack)) {
        printf("Stack Overflow! Cannot push %d\n", data);
    } else {
        stack->arr[++stack->top] = data;
        printf("%d pushed to stack\n", data);
    }
}
int pop(struct Stack *stack) {
    if (isEmpty(stack)) {
        printf("Stack Underflow! Cannot pop\n");
        return -1;
    } else {
        return stack->arr[stack->top--];}}
void display(struct Stack *stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty.\n");
    } else {
        printf("Stack elements: ");
        for (int i = stack->top; i >= 0; i--) {
            printf("%d ", stack->arr[i]);
        }
        printf("\n");}}
int main() {
    struct Stack stack;
    initializeStack(&stack);
    int choice, value;
    printf("Stack Operations:\n");
    printf("1. Push\n");
    printf("2. Pop\n");
    printf("3. Display\n");
    printf("4. Exit\n");
    do {
        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:
                value = pop(&stack);
                if (value != -1) {
                    printf("%d popped from stack\n", value);
                }
                break;
            case 3:
                display(&stack);
                break;
            case 4:
                printf("Exiting program.\n");
                break;
            default:printf("Invalid choice! Please try again.\n");
        }} while (choice != 4);return 0;}
```

12.QUEUE THROUGH LINK LIST

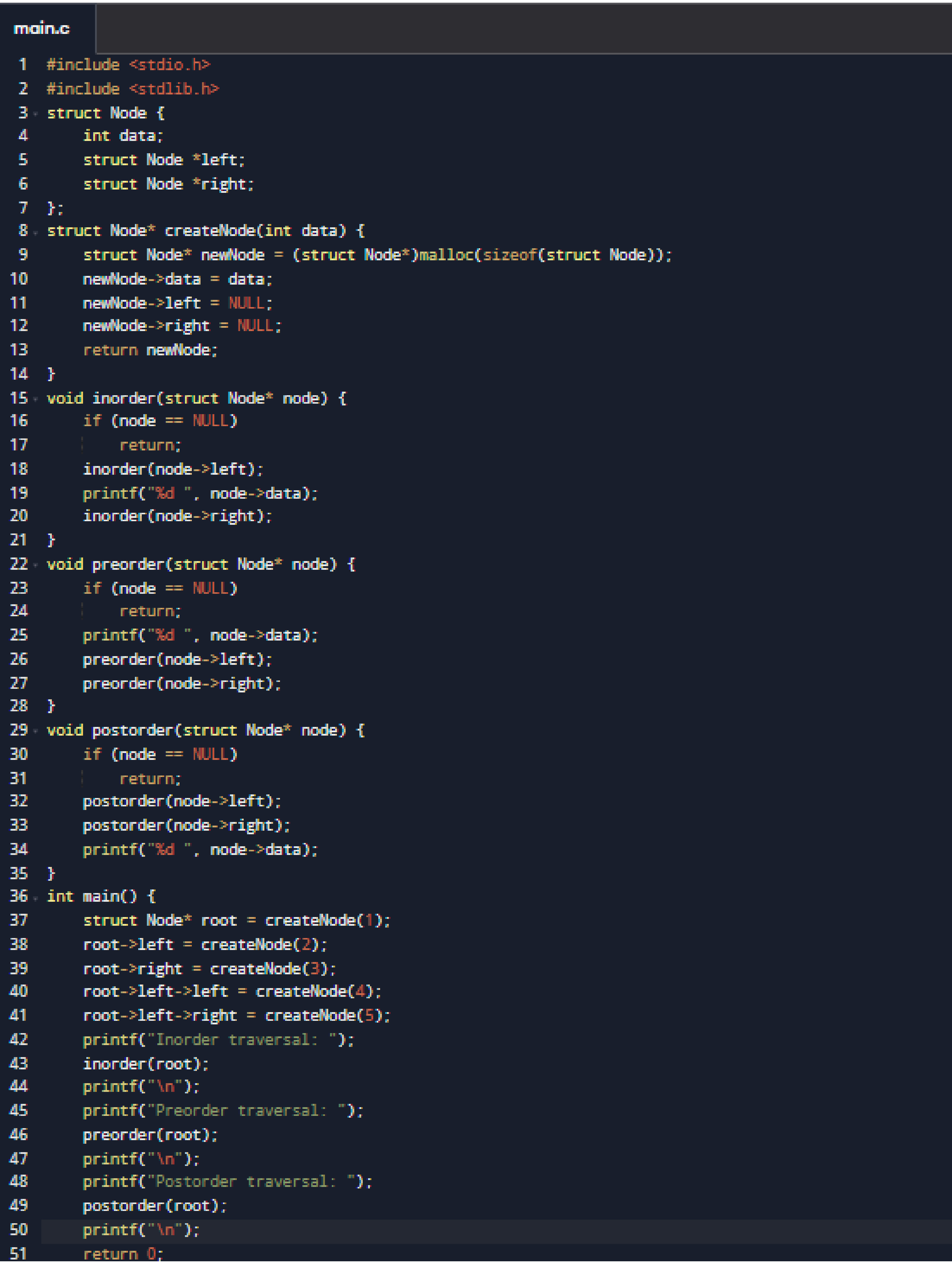


```c
#include <stdio.h>
#include <stdlib.h>
struct QueueNode {
    int data;
    struct QueueNode *next;};
struct Queue {
    struct QueueNode *front;
    struct QueueNode *rear;};
struct QueueNode* createNode(int data) {
    struct QueueNode* newNode = (struct QueueNode*)malloc(sizeof(struct QueueNode));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;}
void initializeQueue(struct Queue *q) {
    q->front = NULL;
    q->rear = NULL;}
int isEmpty(struct Queue *q) {
    return (q->front == NULL);}
void enqueue(struct Queue *q, int data) {
    struct QueueNode* newNode = createNode(data);
    if (isEmpty(q)) {
        q->front = newNode;
        q->rear = newNode;
    } else {
        q->rear->next = newNode;
        q->rear = newNode;}
    printf("Enqueued %d\n", data);}
int dequeue(struct Queue *q) {
    if (isEmpty(q)) {
        printf("Queue Underflow! Cannot dequeue.\n");
        return -1;
    } else {
        struct QueueNode* temp = q->front;
        int dequeuedValue = temp->data;
        q->front = q->front->next;
        if (q->front == NULL) {
            q->rear = NULL;}
        free(temp);
        printf("Dequeued %d\n", dequeuedValue);
        return dequeuedValue;
    }
}
void display(struct Queue *q) {
    if (isEmpty(q)) {
        printf("Queue is empty.\n");
    } else {
        printf("Queue elements: ");
        struct QueueNode* current = q->front;
        while (current != NULL) {
            printf("%d ", current->data);
            current = current->next;
        }
        printf("\n");
    }
}
int main() {
    struct Queue q;
```



```c
    struct Queue q;
    initializeQueue(&q);
    int choice, value;
    printf("Queue Operations (Linked List):\n");
    printf("1. Enqueue (Insert)\n");
    printf("2. Dequeue (Delete)\n");
    printf("3. Display\n");
    printf("4. Exit\n");
    do {
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter value to enqueue: ");
                scanf("%d", &value);
                enqueue(&q, value);
                break;
            case 2:
                dequeue(&q);
                break;
            case 3:
                display(&q);
                break;
            case 4:
                printf("Exiting program.\n");
                break;
            default:
                printf("Invalid choice! Please try again.\n");
        }
    } while (choice != 4);
    struct QueueNode* current = q.front;
    struct QueueNode* next;
    while (current != NULL) {
        next = current->next;
        free(current);
        current = next;
    }
    q.front = q.rear = NULL;
    return 0;
}
```

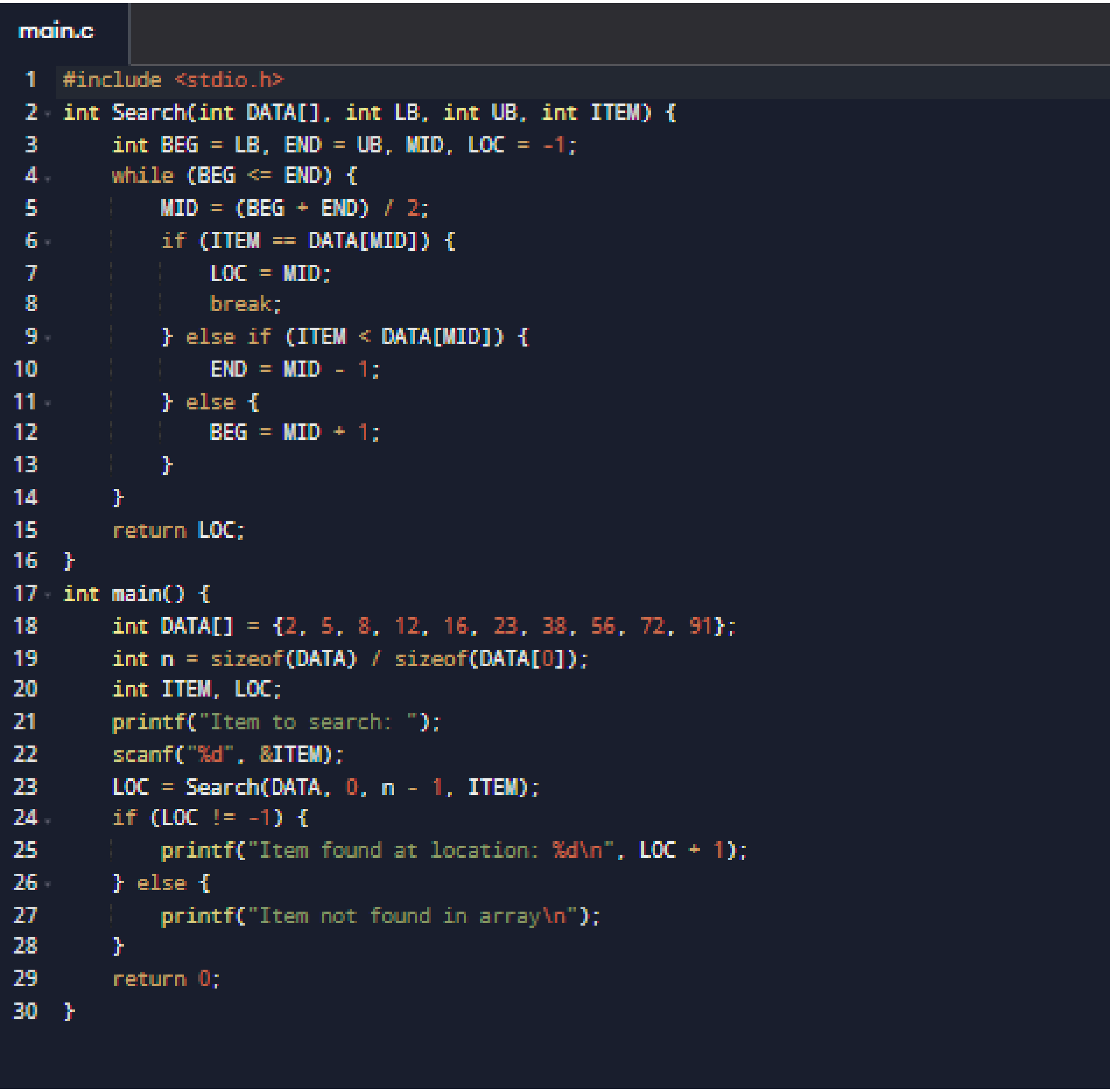
13.TREE TRAVERSAL



```c
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node *left;
    struct Node *right;
};
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}
void inorder(struct Node* node) {
    if (node == NULL)
        return;
    inorder(node->left);
    printf("%d ", node->data);
    inorder(node->right);
}
void preorder(struct Node* node) {
    if (node == NULL)
        return;
    printf("%d ", node->data);
    preorder(node->left);
    preorder(node->right);
}
void postorder(struct Node* node) {
    if (node == NULL)
        return;
    postorder(node->left);
    postorder(node->right);
    printf("%d ", node->data);
}
int main() {
    struct Node* root = createNode(1);
    root->left = createNode(2);
    root->right = createNode(3);
    root->left->left = createNode(4);
    root->left->right = createNode(5);
    printf("Inorder traversal: ");
    inorder(root);
    printf("\n");
    printf("Preorder traversal: ");
    preorder(root);
    printf("\n");
    printf("Postorder traversal: ");
    postorder(root);
    printf("\n");
    return 0;
```

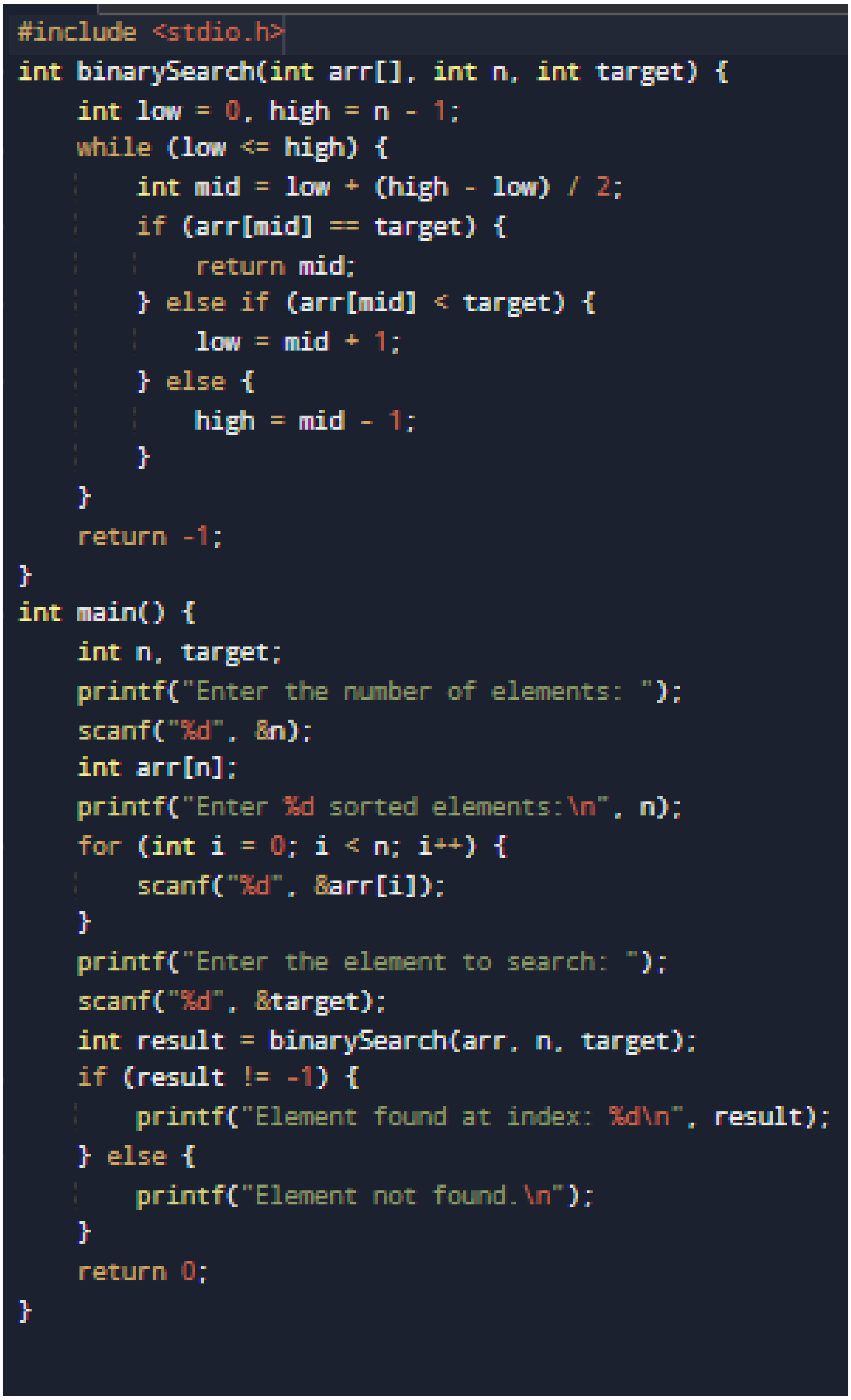
```
}
```

14.TREE SERCHING



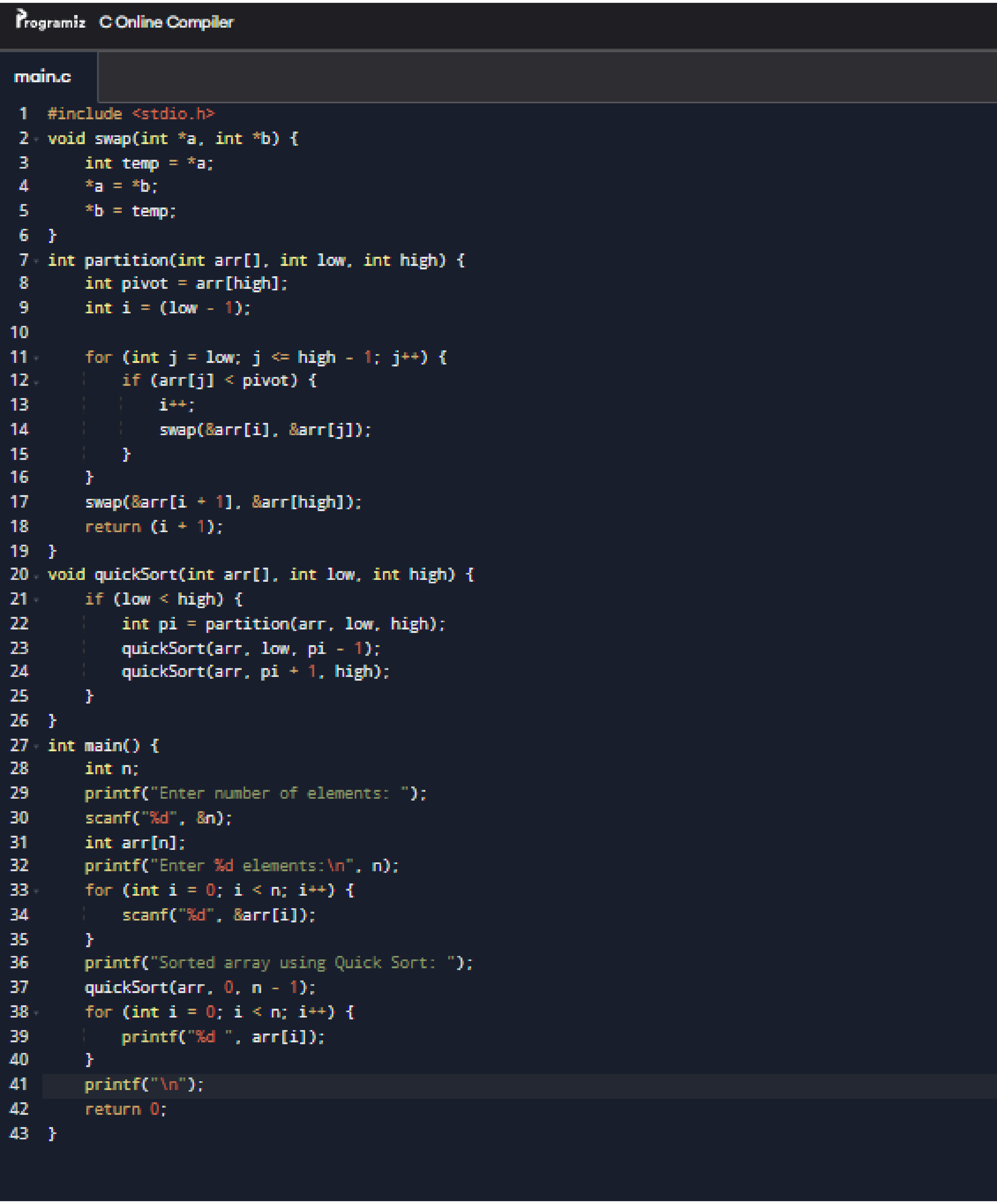
```c
#include <stdio.h>
int Search(int DATA[], int LB, int UB, int ITEM) {
    int BEG = LB, END = UB, MID, LOC = -1;
    while (BEG <= END) {
        MID = (BEG + END) / 2;
        if (ITEM == DATA[MID]) {
            LOC = MID;
            break;
        } else if (ITEM < DATA[MID]) {
            END = MID - 1;
        } else {
            BEG = MID + 1;
        }
    }
    return LOC;
}
int main() {
    int DATA[] = {2, 5, 8, 12, 16, 23, 38, 56, 72, 91};
    int n = sizeof(DATA) / sizeof(DATA[0]);
    int ITEM, LOC;
    printf("Item to search: ");
    scanf("%d", &ITEM);
    LOC = Search(DATA, 0, n - 1, ITEM);
    if (LOC != -1) {
        printf("Item found at location: %d\n", LOC + 1);
    } else {
        printf("Item not found in array\n");
    }
    return 0;
}
```

15.BINARY SEARCH



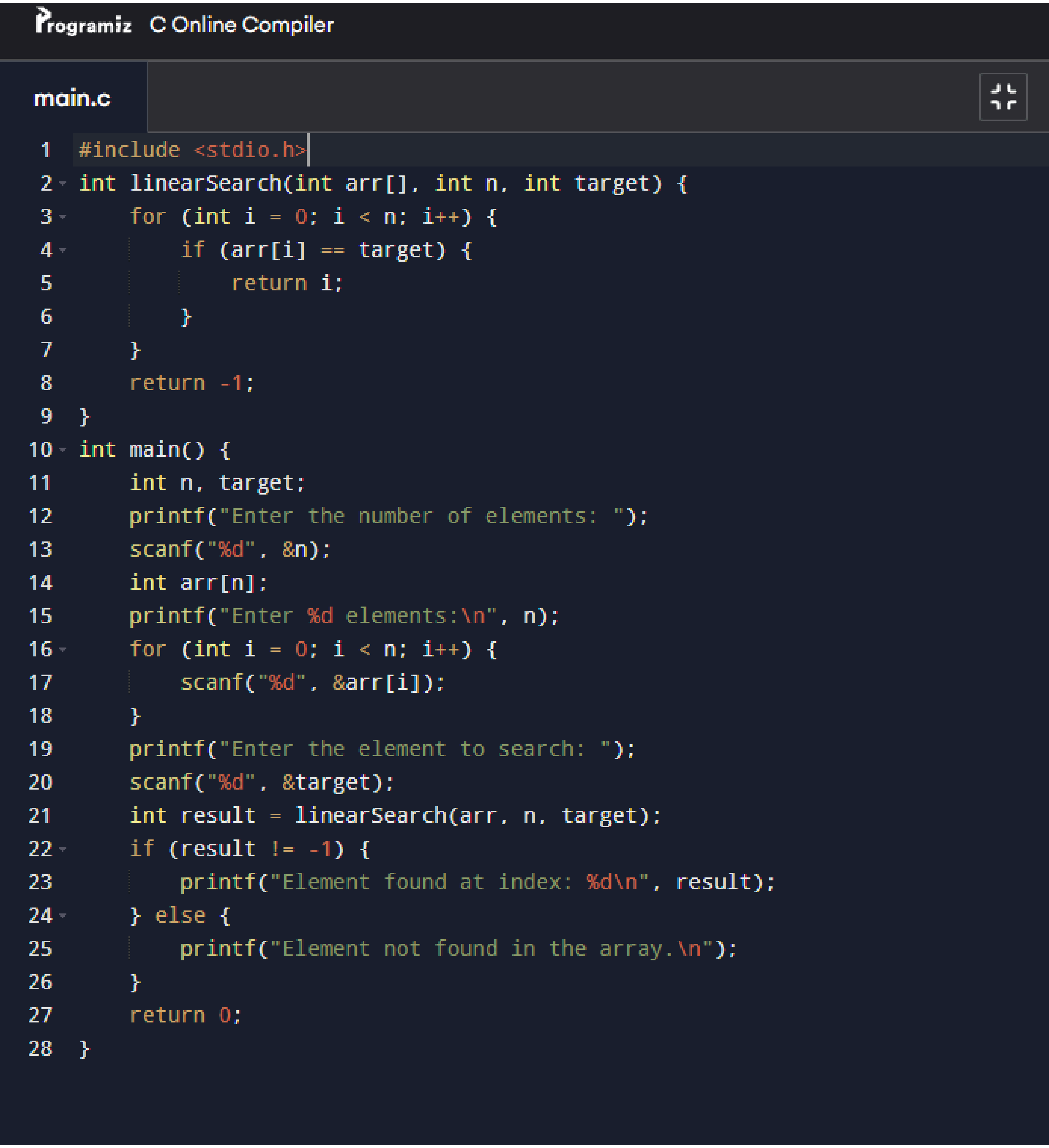
```c
#include <stdio.h>
int binarySearch(int arr[], int n, int target) {
    int low = 0, high = n - 1;
    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (arr[mid] == target) {
            return mid;
        } else if (arr[mid] < target) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }
    return -1;
}
int main() {
    int n, target;
    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++) {
        scanf("%d", &arr[i]);
    }
    printf("Enter the element to search: ");
    scanf("%d", &target);
    int result = binarySearch(arr, n, target);
    if (result != -1) {
        printf("Element found at index: %d\n", result);
    } else {
        printf("Element not found.\n");
    }
    return 0;
}
```

16.SORTING USING QUICK SORT



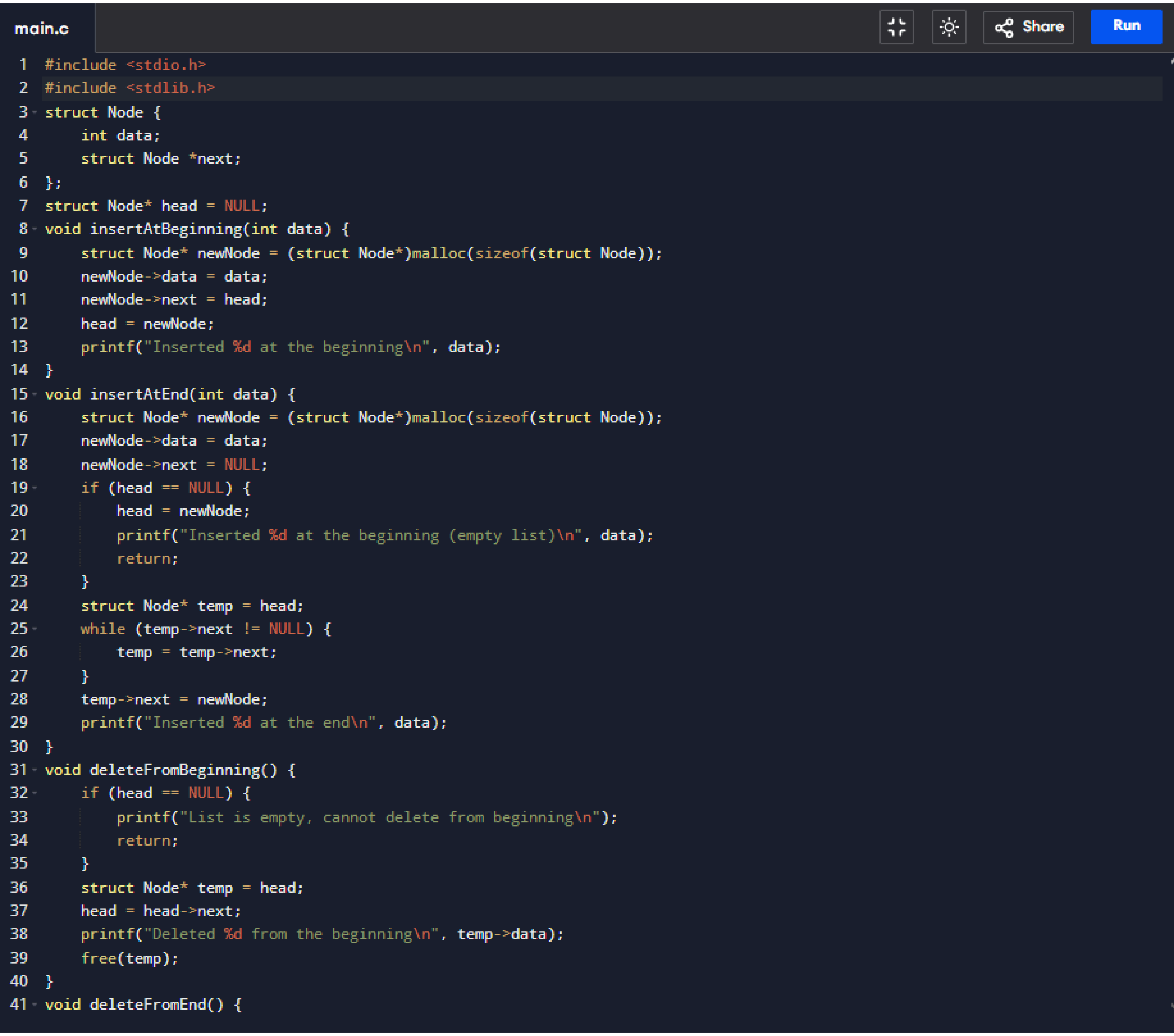
```c
#include <stdio.h>
void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}
int partition(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = (low - 1);

    for (int j = low; j <= high - 1; j++) {
        if (arr[j] < pivot) {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}
int main() {
    int n;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter %d elements:\n", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    printf("Sorted array using Quick Sort: ");
    quickSort(arr, 0, n - 1);
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    return 0;
}
```

17.LINEAR SEARCH METHOD

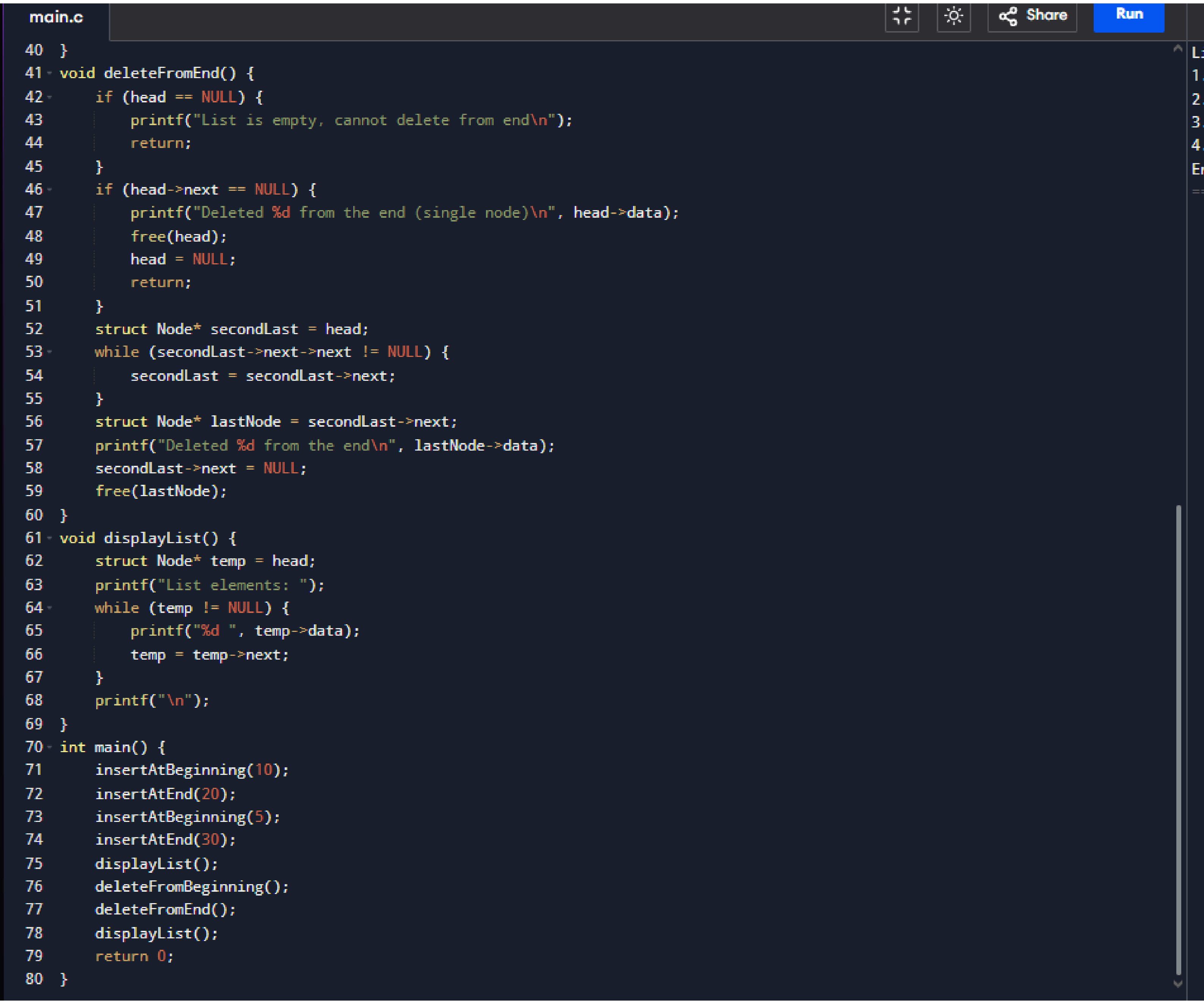


```c
#include <stdio.h>
int linearSearch(int arr[], int n, int target) {
    for (int i = 0; i < n; i++) {
        if (arr[i] == target) {
            return i;
        }
    }
    return -1;
}
int main() {
    int n, target;
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter %d elements:\n", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    printf("Enter the element to search: ");
    scanf("%d", &target);
    int result = linearSearch(arr, n, target);
    if (result != -1) {
        printf("Element found at index: %d\n", result);
    } else {
        printf("Element not found in the array.\n");
    }
    return 0;
}
```

18.CONCATENATION OF TWO LINKED LIST

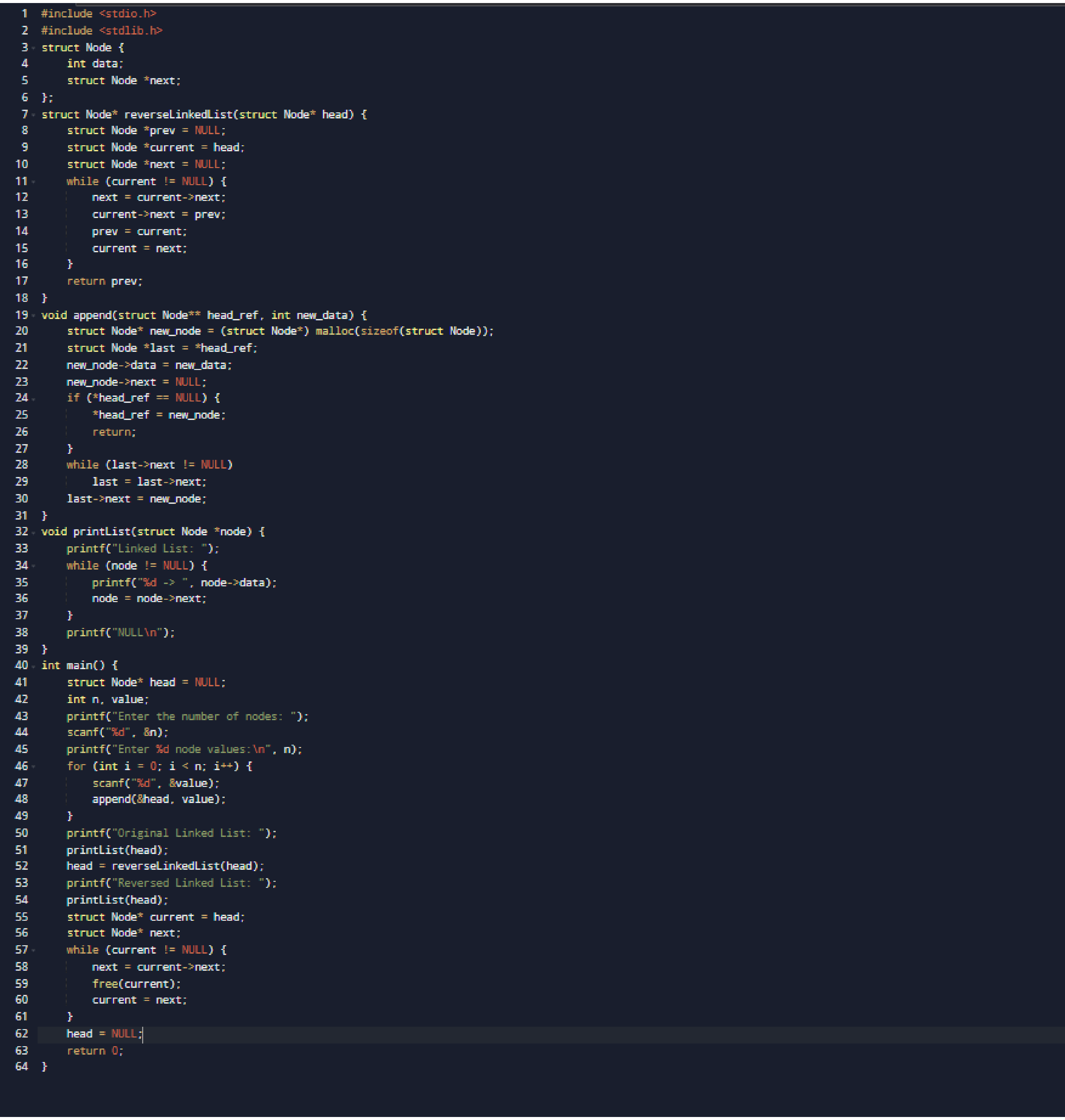


```c
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node *next;
};
struct Node* head = NULL;
void insertAtBeginning(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = head;
    head = newNode;
    printf("Inserted %d at the beginning\n", data);
}
void insertAtEnd(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    if (head == NULL) {
        head = newNode;
        printf("Inserted %d at the beginning (empty list)\n", data);
        return;
    }
    struct Node* temp = head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
    printf("Inserted %d at the end\n", data);
}
void deleteFromBeginning() {
    if (head == NULL) {
        printf("List is empty, cannot delete from beginning\n");
        return;
    }
    struct Node* temp = head;
    head = head->next;
    printf("Deleted %d from the beginning\n", temp->data);
    free(temp);
}
void deleteFromEnd() {
```



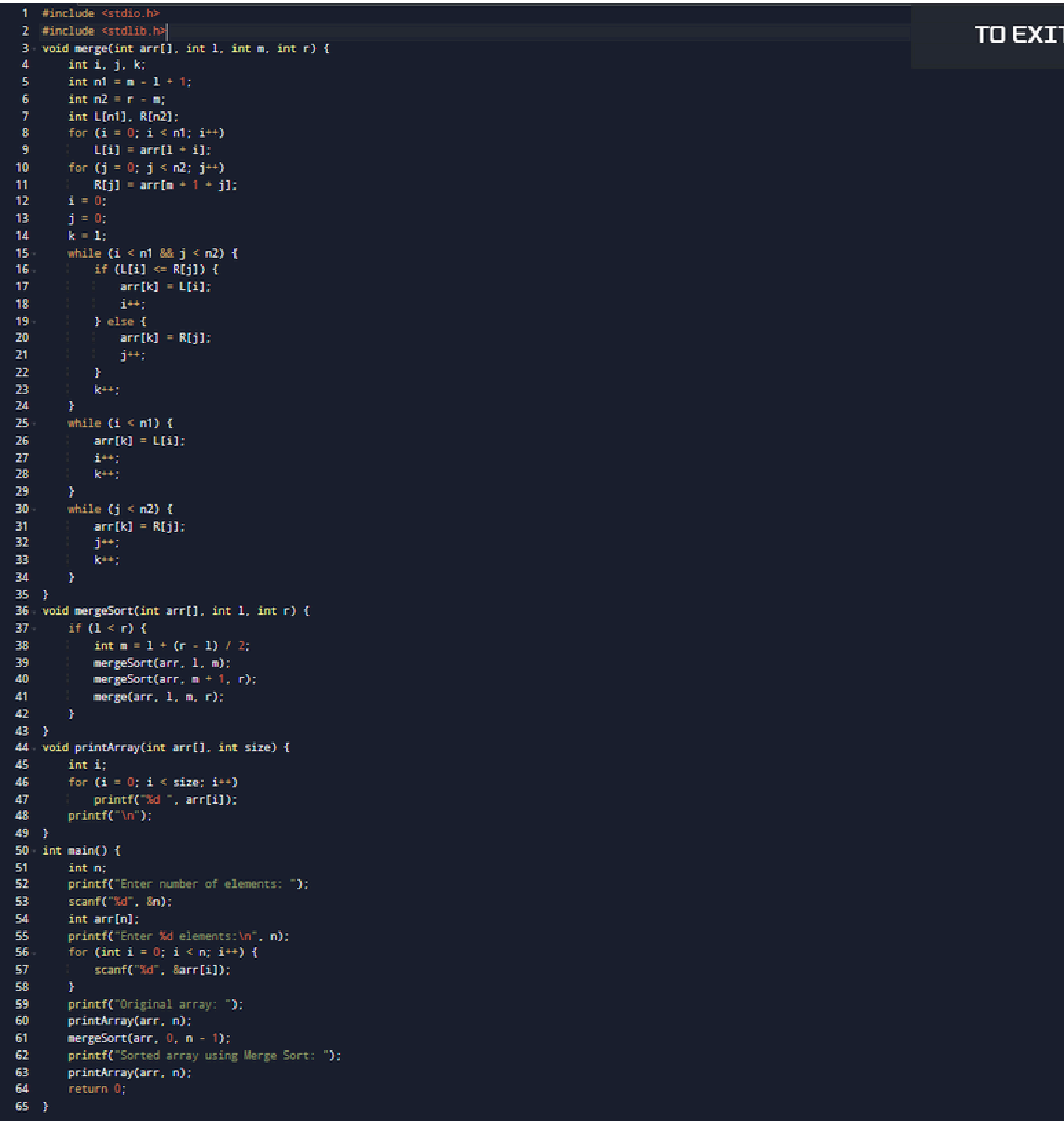
```c
}
void deleteFromEnd() {
    if (head == NULL) {
        printf("List is empty, cannot delete from end\n");
        return;
    }
    if (head->next == NULL) {
        printf("Deleted %d from the end (single node)\n", head->data);
        free(head);
        head = NULL;
        return;
    }
    struct Node* secondLast = head;
    while (secondLast->next->next != NULL) {
        secondLast = secondLast->next;
    }
    struct Node* lastNode = secondLast->next;
    printf("Deleted %d from the end\n", lastNode->data);
    secondLast->next = NULL;
    free(lastNode);
}
void displayList() {
    struct Node* temp = head;
    printf("List elements: ");
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
int main() {
    insertAtBeginning(10);
    insertAtEnd(20);
    insertAtBeginning(5);
    insertAtEnd(30);
    displayList();
    deleteFromBeginning();
    deleteFromEnd();
    displayList();
    return 0;
}
```

19.REVERSE A LINKED LIST



```c
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node *next;
};
struct Node* reverseLinkedList(struct Node* head) {
    struct Node *prev = NULL;
    struct Node *current = head;
    struct Node *next = NULL;
    while (current != NULL) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    return prev;
}
void append(struct Node** head_ref, int new_data) {
    struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));
    struct Node *last = *head_ref;
    new_node->data = new_data;
    new_node->next = NULL;
    if (*head_ref == NULL) {
        *head_ref = new_node;
        return;
    }
    while (last->next != NULL)
        last = last->next;
    last->next = new_node;
}
void printList(struct Node *node) {
    printf("Linked List: ");
    while (node != NULL) {
        printf("%d -> ", node->data);
        node = node->next;
    }
    printf("NULL\n");
}
int main() {
    struct Node* head = NULL;
    int n, value;
    printf("Enter the number of nodes: ");
    scanf("%d", &n);
    printf("Enter %d node values:\n", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &value);
        append(&head, value);
    }
    printf("Original Linked List: ");
    printList(head);
    head = reverseLinkedList(head);
    printf("Reversed Linked List: ");
    printList(head);
    struct Node* current = head;
    struct Node* next;
    while (current != NULL) {
        next = current->next;
        free(current);
        current = next;
    }
    head = NULL;
    return 0;
}
```

20.SORTING USING MERGE SORT



```c
#include <stdio.h>
#include <stdlib.h>
void merge(int arr[], int l, int m, int r) {
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;
    int L[n1], R[n2];
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];
    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}
void mergeSort(int arr[], int l, int r) {
    if (l < r) {
        int m = l + (r - l) / 2;
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}
void printArray(int arr[], int size) {
    int i;
    for (i = 0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}
int main() {
    int n;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter %d elements:\n", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    printf("Original array: ");
    printArray(arr, n);
    mergeSort(arr, 0, n - 1);
    printf("Sorted array using Merge Sort: ");
    printArray(arr, n);
    return 0;
}
```