Name: Pawan Sadashiv Bhirawandekar

Subject: Data Structure and Algorithms

Theory Question

1 What is data Structure? Explain the Classification of Data Structure in detail?

A Data Structure is a method of organizing, storing, and managing data in a computer so that it can be accessed and used efficiently.
It defines how data is arranged, how operations like insertion, deletion, searching, and sorting are performed, and how memory is managed.

Classification of Data Structures:

A. Primitive Data Structures

These are the basic data types provided by programming languages.

Examples:

- int
- float
- char
- double
- boolean

They are the building blocks for other complex data structures.

B. Non-Primitive Data Structures

These are more complex structures built using primitive data types.

Non-Primitive data structures are further divided into:

1. Linear Data Structures

In linear data structures, data elements are arranged in a sequential manner.
Each element is connected to the previous and next element.

Examples:

a) Array

- Stores elements of the same data type in contiguous memory.
- Example: int arr[5]

b) Linked List

- Collection of nodes connected using pointers.
- Types: Singly, Doubly, Circular Linked List

c) Stack

- Follows LIFO (Last In First Out) principle.
- Operations: push(), pop()

d) Queue

- Follows FIFO (First In First Out) principle.
- Types: Simple queue, Circular queue, Priority queue, Deque

C. Non-Linear Data Structures

In these structures, data is stored **hierarchically** or **in a network** form.

Examples:

a) Tree

- Represents data in hierarchical form.
- Types: Binary tree, BST, AVL, B-tree, Heap

b) Graph

- Represents relationships between nodes.
- Consists of vertices and edges.
- Types: Directed/Undirected, Weighted/Unweighted.

2. Explain the Types of an Array. Write down syntax for declaration, initialization and how to access elements from Array in C?

An array is a linear data structure that stores a fixed-size sequence of elements of the same data type in contiguous memory locations. Each element can be accessed directly using its index, which allows for efficient retrieval and modification.

Arrays are mainly classified into the following types:

1. One-Dimensional Array (1D Array)

Stores data in a single row.

Example: int marks[5];

2. Two-Dimensional Array (2D Array)

Stores data in **rows and columns**, like a matrix or table.

Example:

int matrix[3][3];

3. Multi-Dimensional Array (3D or more)

Arrays with more than two dimensions.

Example:

int arr[3][4][5];

- **Array Declaration Syntax:**
  data_type array_name[size];

  Example: int arr[10];

- **Array Initialization Syntax:**
  Example: int arr[5] = {10, 20, 30, 40, 50};
  int arr[] = {1, 2, 3, 4}; //**Without specifying size**

- **Accessing Elements of an Array:**
  Example: printf("%d", arr[2]);   // prints third element
  Using Loops:

```
for(int i=0; i<5; i++){
  printf("%d ", arr[i]);
}
```

  Printing 2D array

```
for(int i=0; i<3; i++){
  for(int j=0; j<3; j++){
    printf("%d ", mat[i][j]);
  }
  printf("\n");
}
```

3. What is stack? Explain the working of PUSH and POP operations in the stack with examples.

Stack is a linear data structure that follows **LIFO** (Last In First Out) Principle, the last element inserted is the first to be popped out. It means both insertion and deletion operations happen at one end only.

Basic Terminologies of Stack

- **Top**: The position of the most recently inserted element. Insertions (push) and deletions (pop) are always performed at the top.
- **Size**: Refers to the current number of elements present in the stack.

Common Operations on Stack:

In order to make manipulations in a stack, there are certain operations provided to us.

- **push()** to insert an element into the stack.
- **pop()** to remove an element from the stack.
- **top()** Returns the top element of the stack.
- **isEmpty()** returns true if stack is empty else false.
- **size()** returns the size of the stack.

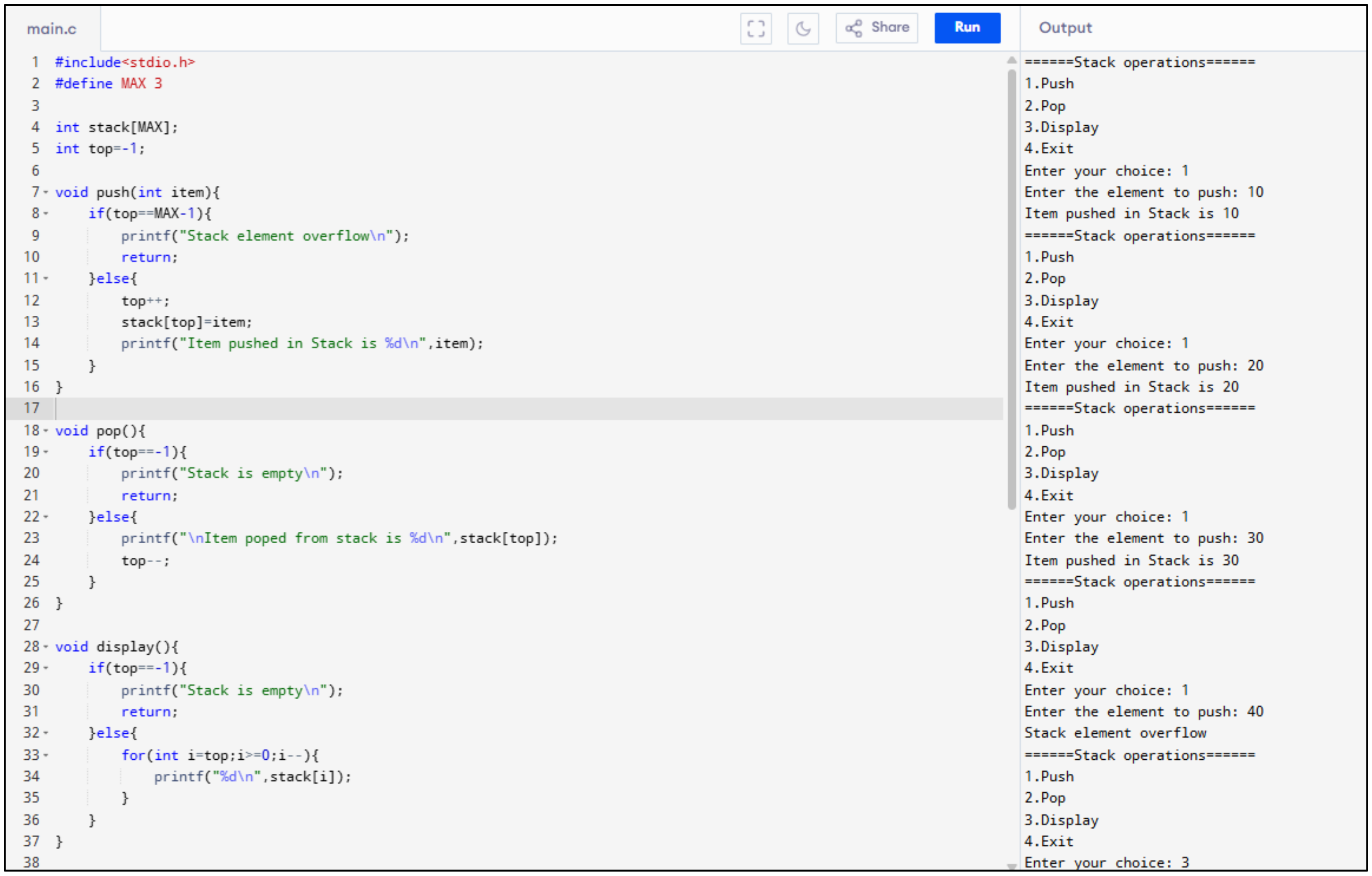
1. PUSH Operation

PUSH operation is used to **add an element** to the **top** of the stack.

2. POP Operation

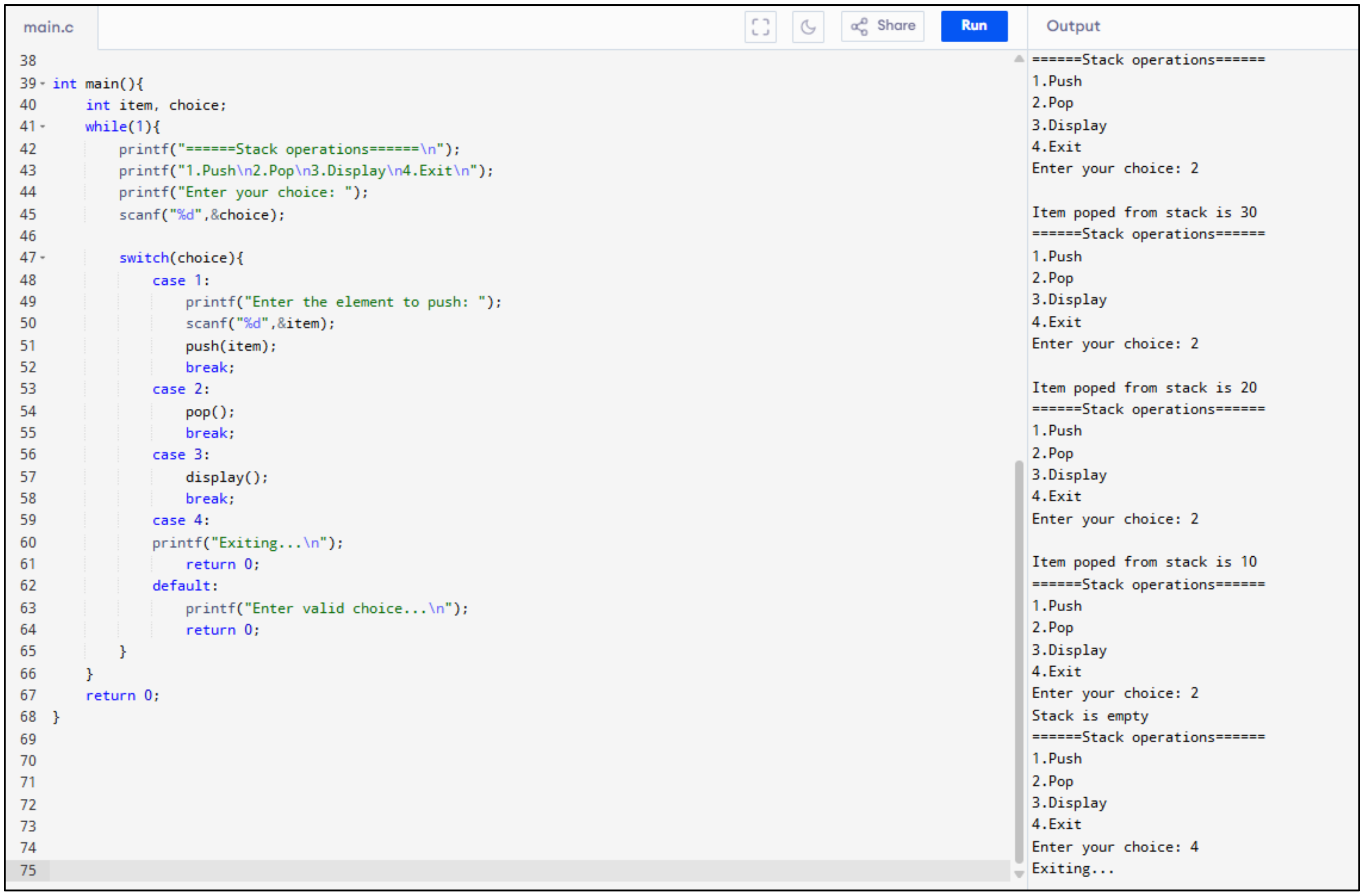
POP operation is used to **remove and return** the element from the **top** of the stack.

Example:



```c
#include<stdio.h>
#define MAX 3

int stack[MAX];
int top=-1;

void push(int item){
    if(top==MAX-1){
        printf("Stack element overflow\n");
        return;
    }else{
        top++;
        stack[top]=item;
        printf("Item pushed in Stack is %d\n",item);
    }
}

void pop(){
    if(top==-1){
        printf("Stack is empty\n");
        return;
    }else{
        printf("\nItem poped from stack is %d\n",stack[top]);
        top--;
    }
}

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

```

```
======Stack operations======
1.Push
2.Pop
3.Display
4.Exit
Enter your choice: 1
Enter the element to push: 10
Item pushed in Stack is 10
======Stack operations======
1.Push
2.Pop
3.Display
4.Exit
Enter your choice: 1
Enter the element to push: 20
Item pushed in Stack is 20
======Stack operations======
1.Push
2.Pop
3.Display
4.Exit
Enter your choice: 1
Enter the element to push: 30
Item pushed in Stack is 30
======Stack operations======
1.Push
2.Pop
3.Display
4.Exit
Enter your choice: 1
Enter the element to push: 40
Stack element overflow
======Stack operations======
1.Push
2.Pop
3.Display
4.Exit
Enter your choice: 3
```



```c

int main(){
    int item, choice;
    while(1){
        printf("======Stack operations======\n");
        printf("1.Push\n2.Pop\n3.Display\n4.Exit\n");
        printf("Enter your choice: ");
        scanf("%d",&choice);

        switch(choice){
            case 1:
                printf("Enter the element to push: ");
                scanf("%d",&item);
                push(item);
                break;
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
            case 4:
            printf("Exiting...\n");
                return 0;
            default:
                printf("Enter valid choice...\n");
                return 0;
        }
    }
    return 0;
}
```

```
======Stack operations======
1.Push
2.Pop
3.Display
4.Exit
Enter your choice: 2

Item poped from stack is 30
======Stack operations======
1.Push
2.Pop
3.Display
4.Exit
Enter your choice: 2

Item poped from stack is 20
======Stack operations======
1.Push
2.Pop
3.Display
4.Exit
Enter your choice: 2

Item poped from stack is 10
======Stack operations======
1.Push
2.Pop
3.Display
4.Exit
Enter your choice: 2
Stack is empty
======Stack operations======
1.Push
2.Pop
3.Display
4.Exit
Enter your choice: 4
Exiting...
```

4. Give the difference between Simple Queue V/S Circular Queue?

Simple Queue	Circular Queue
Also called Linear Queue.	Also called Ring Buffer Queue.
Elements are arranged in a straight line.	Last position is connected back to the first, forming a circle.
FIFO (First In First Out) order.	Also follows FIFO.
Once the rear reaches the end, no more insertions are allowed even if space is free at the beginning. (Wastage of memory)	Space is fully utilized. After the rear reaches the end, it loops back to the front. No memory wastage.
Condition for Overflow: rear == size-1	Condition for Overflow: (rear + 1) % size == front
Condition for Underflow: ` front == -1	
Front always moves forward and never returns to previous positions.	Front and rear move in a circular manner using modulo (%) operator.
Implementation is simpler.	Slightly more difficult because of circular linkage.
Example use: Basic line systems (ticket counter).	Example use: CPU scheduling, buffering, real-time systems.

5. How do singly and doubly linked list differ from each other? Explain with Proper Example?

Singly linked list : A singly linked list is a set of nodes where each node has two fields 'data' and 'link'. The 'data' field stores actual piece of information and 'link' field is used to point to next node. Basically the 'link' field stores the address of the next node.

In Singly linked list, the traversal can be done using the next node link only. Thus traversal is possible in one direction only.

Complexity of deletion with a given node is O(n), because the previous node needs to be known, and traversal takes O(n)

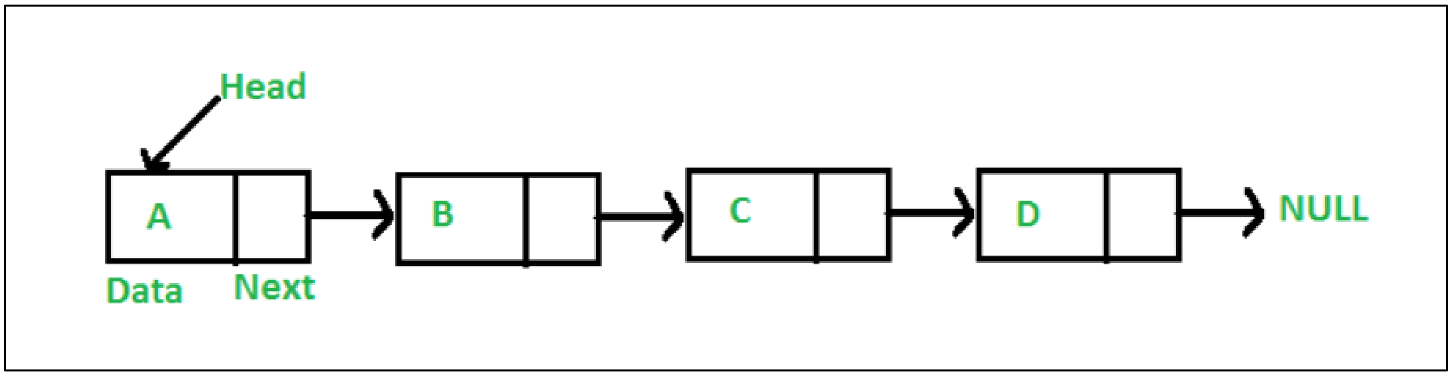
A singly linked list consumes less memory as compared to the doubly linked list.

Singly linked list is relatively less used in practice due to limited number of operations

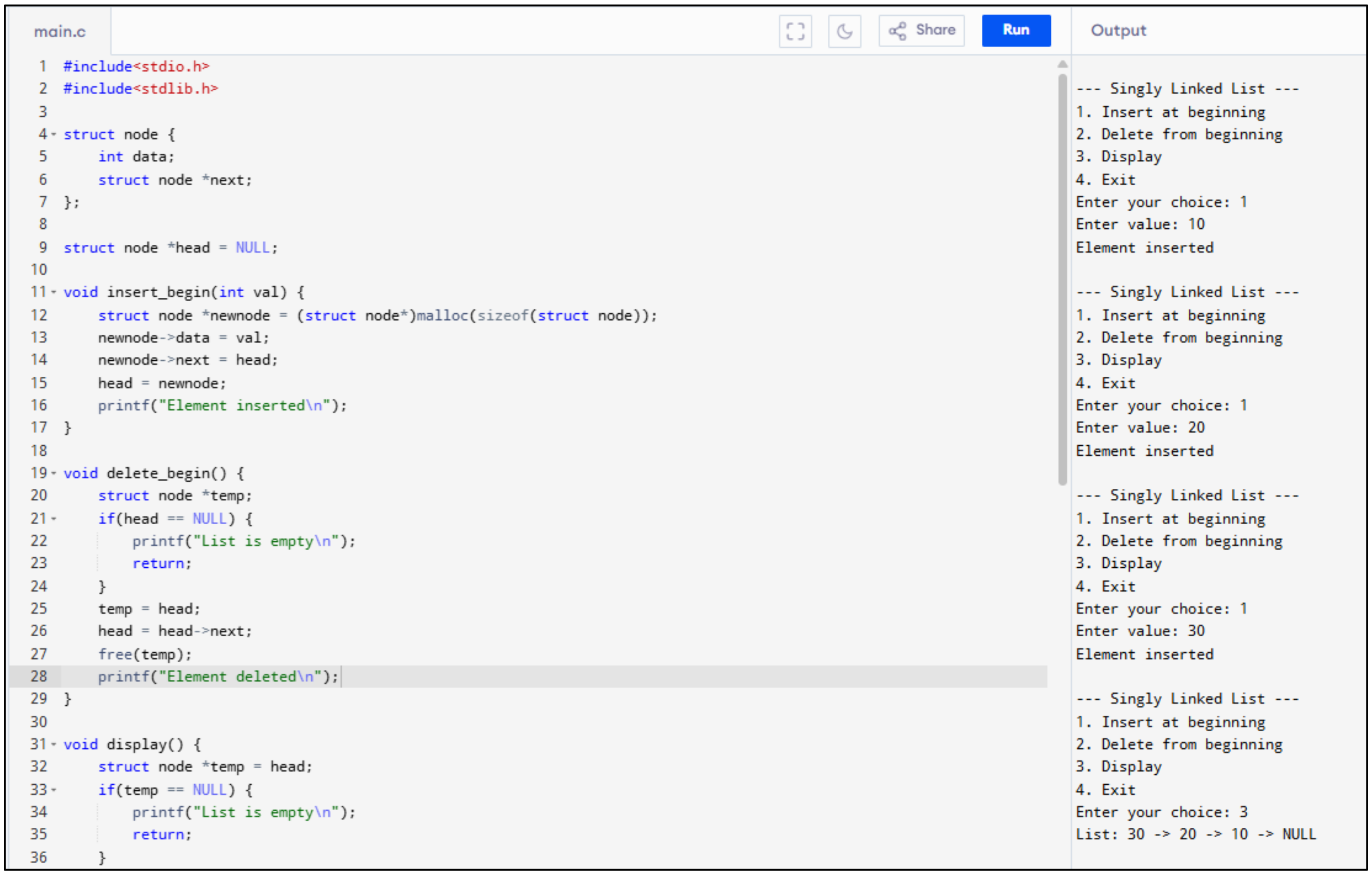
Characteristics:

- Moves **in one direction** only (forward).
- Requires **less memory** (only one pointer per node).

- **Traversal** is simple but **reverse traversal is not possible**.
- **Deletion of previous node** is difficult (need to traverse from head).

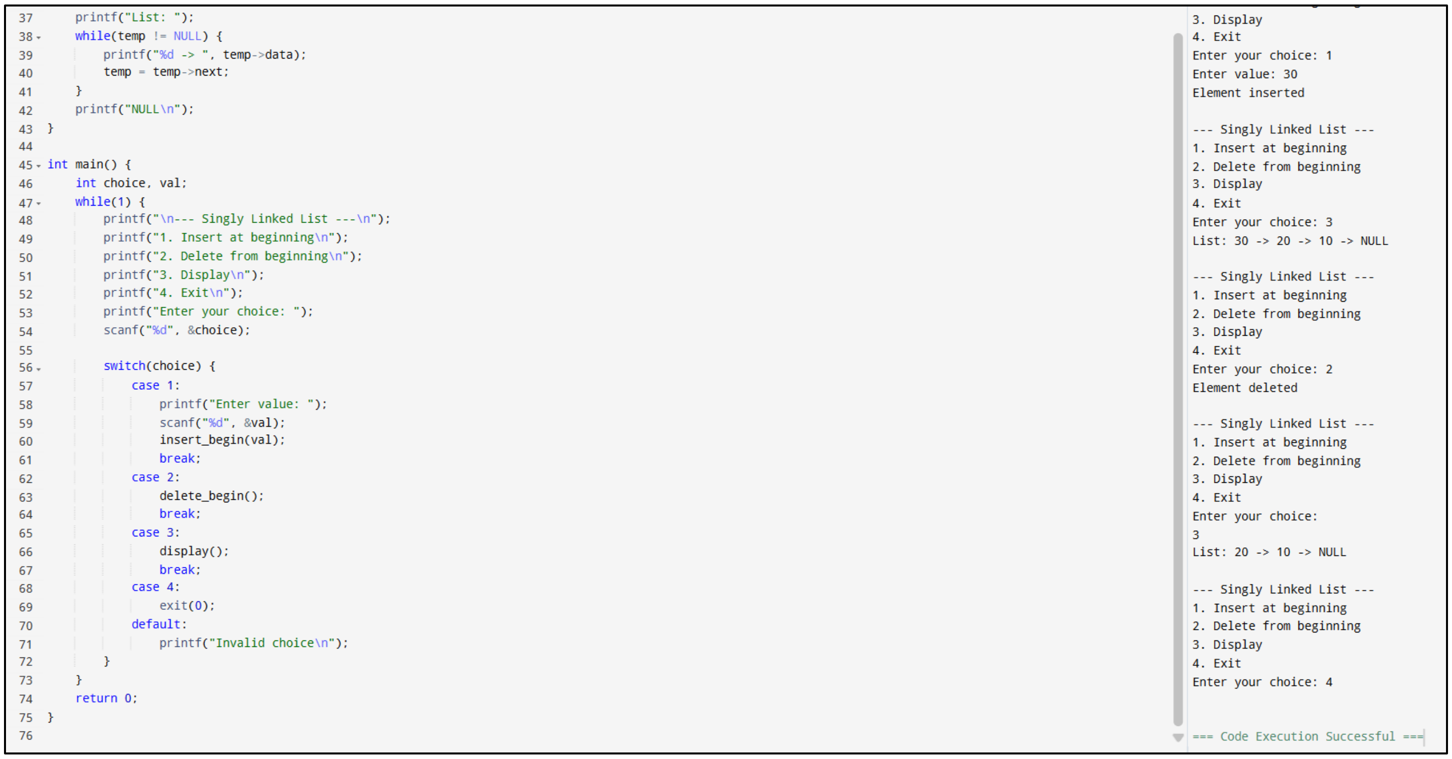


Example:



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

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

struct node *head = NULL;

void insert_begin(int val) {
    struct node *newnode = (struct node*)malloc(sizeof(struct node));
    newnode->data = val;
    newnode->next = head;
    head = newnode;
    printf("Element inserted\n");
}

void delete_begin() {
    struct node *temp;
    if(head == NULL) {
        printf("List is empty\n");
        return;
    }
    temp = head;
    head = head->next;
    free(temp);
    printf("Element deleted\n");
}

void display() {
    struct node *temp = head;
    if(temp == NULL) {
        printf("List is empty\n");
        return;
    }
```

```
--- Singly Linked List ---
1. Insert at beginning
2. Delete from beginning
3. Display
4. Exit
Enter your choice: 1
Enter value: 10
Element inserted

--- Singly Linked List ---
1. Insert at beginning
2. Delete from beginning
3. Display
4. Exit
Enter your choice: 1
Enter value: 20
Element inserted

--- Singly Linked List ---
1. Insert at beginning
2. Delete from beginning
3. Display
4. Exit
Enter your choice: 1
Enter value: 30
Element inserted

--- Singly Linked List ---
1. Insert at beginning
2. Delete from beginning
3. Display
4. Exit
Enter your choice: 3
List: 30 -> 20 -> 10 -> NULL
```



```c
    printf("List: ");
    while(temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

int main() {
    int choice, val;
    while(1) {
        printf("\n--- Singly Linked List ---\n");
        printf("1. Insert at beginning\n");
        printf("2. Delete from beginning\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch(choice) {
            case 1:
                printf("Enter value: ");
                scanf("%d", &val);
                insert_begin(val);
                break;
            case 2:
                delete_begin();
                break;
            case 3:
                display();
                break;
            case 4:
                exit(0);
            default:
                printf("Invalid choice\n");
        }
    }
    return 0;
}
```

```
3. Display
4. Exit
Enter your choice: 1
Enter value: 30
Element inserted

--- Singly Linked List ---
1. Insert at beginning
2. Delete from beginning
3. Display
4. Exit
Enter your choice: 3
List: 30 -> 20 -> 10 -> NULL

--- Singly Linked List ---
1. Insert at beginning
2. Delete from beginning
3. Display
4. Exit
Enter your choice: 2
Element deleted

--- Singly Linked List ---
1. Insert at beginning
2. Delete from beginning
3. Display
4. Exit
Enter your choice:
3
List: 20 -> 10 -> NULL

--- Singly Linked List ---
1. Insert at beginning
2. Delete from beginning
3. Display
4. Exit
Enter your choice: 4

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

Doubly linked list : A Doubly Linked List (DLL) contains an extra pointer, typically called *previous pointer*, together with next pointer and data which are there in singly linked list.

In Doubly linked list, the traversal can be done using the previous node link or the next node link. Thus traversal is possible in both directions (forward and backward).

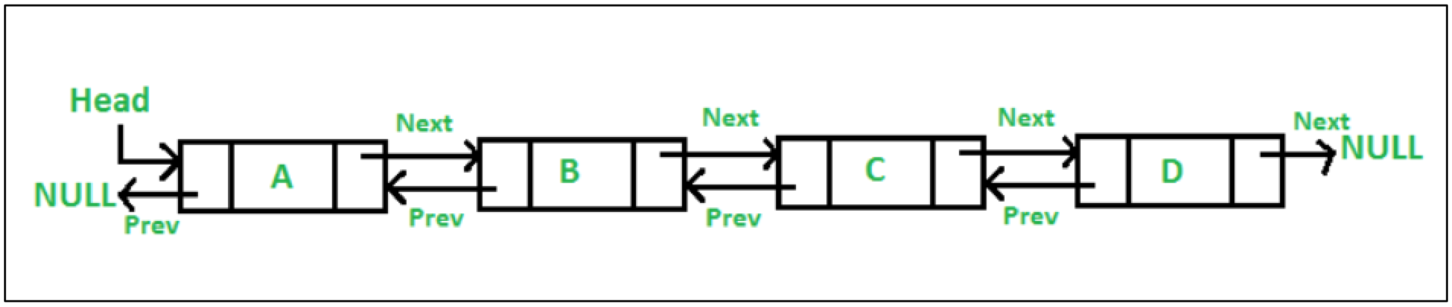
Complexity of deletion with a given node is O(1) because the previous node can be accessed easily

The doubly linked list consumes more memory as compared to the singly linked list.

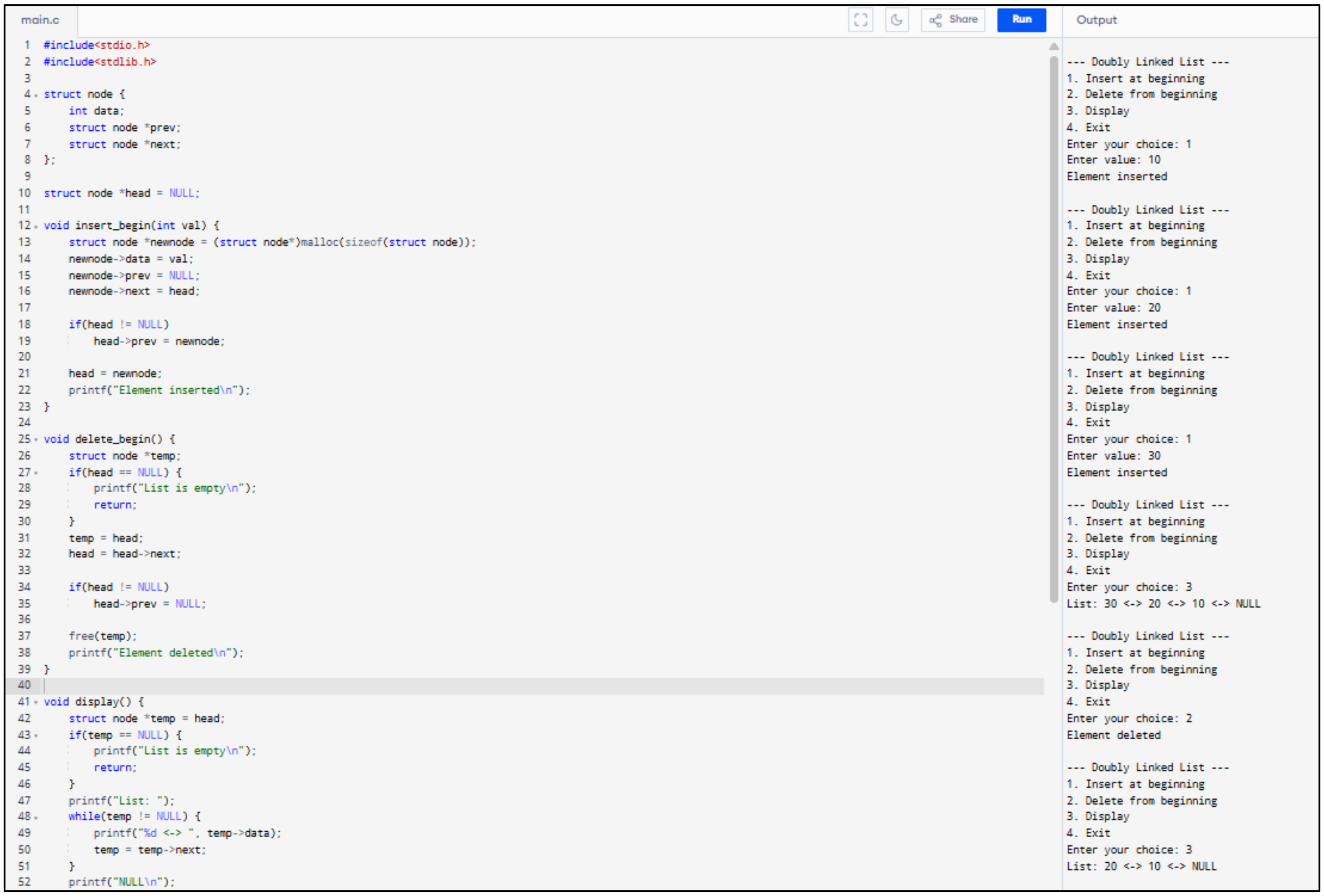
Doubly linked list is implemented more in libraries due to wider number of operations.

Characteristics:

- Moves **in both directions** (forward and backward).
- Requires **more memory** (two pointers per node).
- Easy backward traversal.
- Easy deletion of any node because the previous pointer is available.



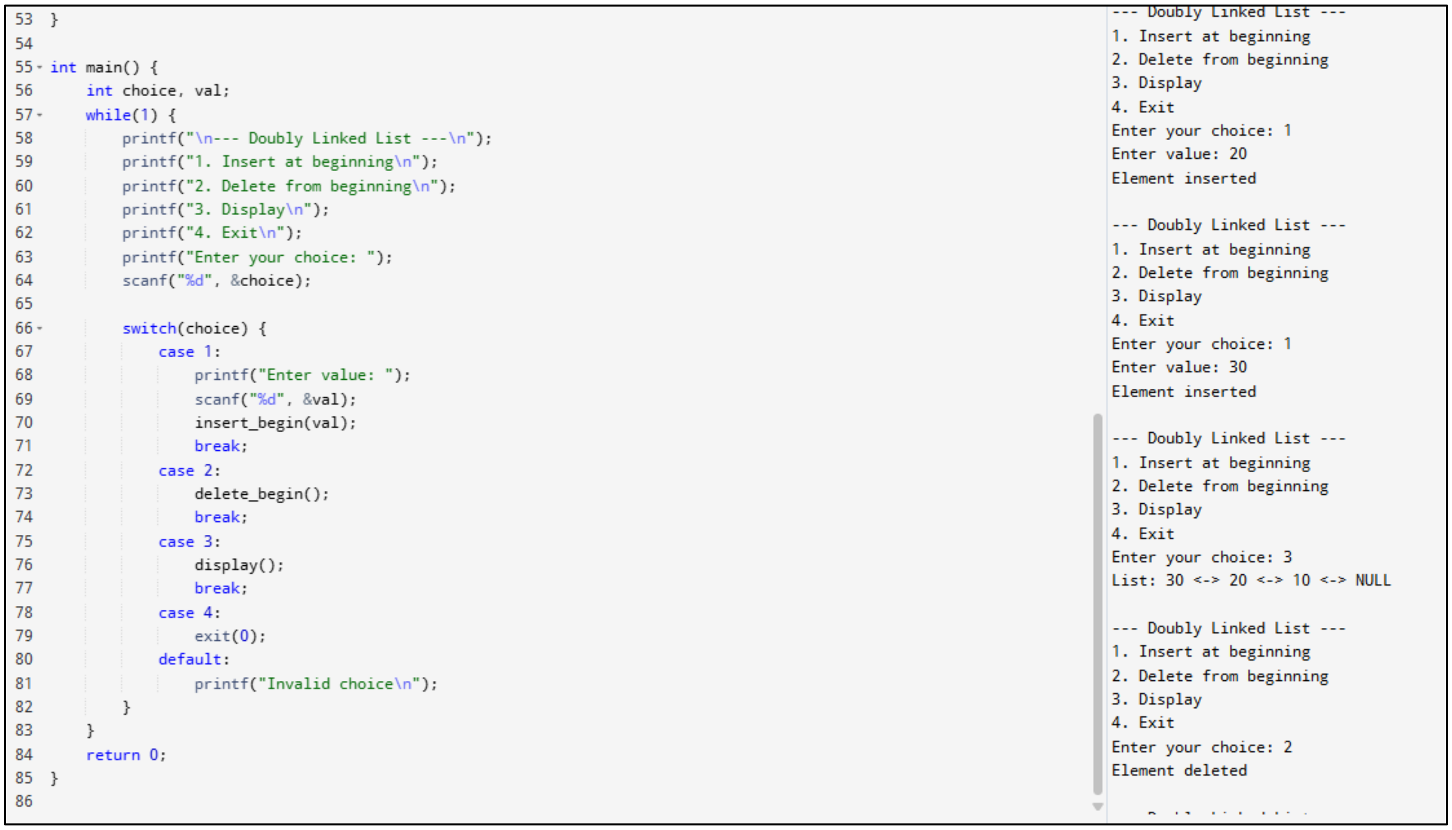
Example:


main.c

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

struct node {
    int data;
    struct node *prev;
    struct node *next;
};

struct node *head = NULL;

void insert_begin(int val) {
    struct node *newnode = (struct node*)malloc(sizeof(struct node));
    newnode->data = val;
    newnode->prev = NULL;
    newnode->next = head;

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

    head = newnode;
    printf("Element inserted\n");
}

void delete_begin() {
    struct node *temp;
    if(head == NULL) {
        printf("List is empty\n");
        return;
    }
    temp = head;
    head = head->next;

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

    free(temp);
    printf("Element deleted\n");
}

void display() {
    struct node *temp = head;
    if(temp == NULL) {
        printf("List is empty\n");
        return;
    }
    printf("List: ");
    while(temp != NULL) {
        printf("%d <-> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
```

Output

```
--- Doubly Linked List ---
1. Insert at beginning
2. Delete from beginning
3. Display
4. Exit
Enter your choice: 1
Enter value: 10
Element inserted

--- Doubly Linked List ---
1. Insert at beginning
2. Delete from beginning
3. Display
4. Exit
Enter your choice: 1
Enter value: 20
Element inserted

--- Doubly Linked List ---
1. Insert at beginning
2. Delete from beginning
3. Display
4. Exit
Enter your choice: 1
Enter value: 30
Element inserted

--- Doubly Linked List ---
1. Insert at beginning
2. Delete from beginning
3. Display
4. Exit
Enter your choice: 3
List: 30 <-> 20 <-> 10 <-> NULL

--- Doubly Linked List ---
1. Insert at beginning
2. Delete from beginning
3. Display
4. Exit
Enter your choice: 2
Element deleted

--- Doubly Linked List ---
1. Insert at beginning
2. Delete from beginning
3. Display
4. Exit
Enter your choice: 3
List: 20 <-> 10 <-> NULL
```

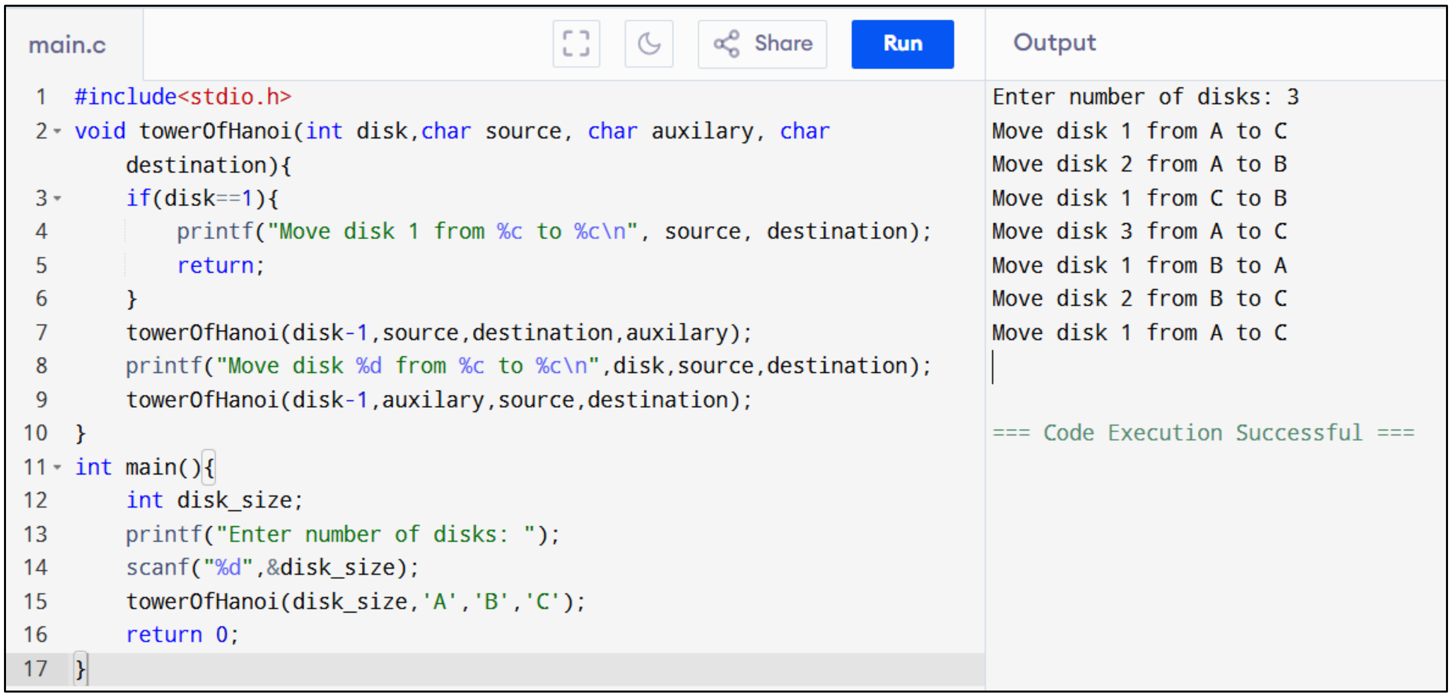


```c
}

int main() {
    int choice, val;
    while(1) {
        printf("\n--- Doubly Linked List ---\n");
        printf("1. Insert at beginning\n");
        printf("2. Delete from beginning\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch(choice) {
            case 1:
                printf("Enter value: ");
                scanf("%d", &val);
                insert_begin(val);
                break;
            case 2:
                delete_begin();
                break;
            case 3:
                display();
                break;
            case 4:
                exit(0);
            default:
                printf("Invalid choice\n");
        }
    }
    return 0;
}
```

```
--- Doubly Linked List ---
1. Insert at beginning
2. Delete from beginning
3. Display
4. Exit
Enter your choice: 1
Enter value: 20
Element inserted

--- Doubly Linked List ---
1. Insert at beginning
2. Delete from beginning
3. Display
4. Exit
Enter your choice: 1
Enter value: 30
Element inserted

--- Doubly Linked List ---
1. Insert at beginning
2. Delete from beginning
3. Display
4. Exit
Enter your choice: 3
List: 30 <-> 20 <-> 10 <-> NULL

--- Doubly Linked List ---
1. Insert at beginning
2. Delete from beginning
3. Display
4. Exit
Enter your choice: 2
Element deleted
```

Practical Question

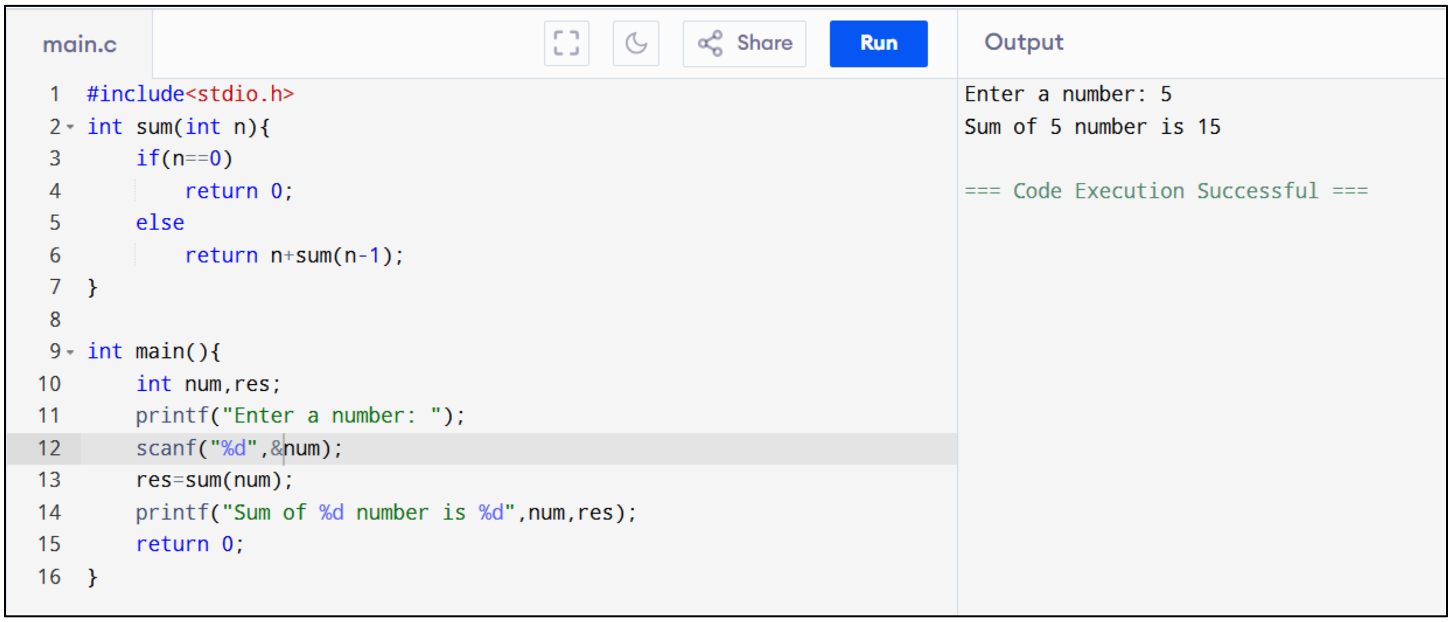
1. Write a Program to Perform Tower of Hanoi Problem.



```c
#include<stdio.h>
void towerOfHanoi(int disk,char source, char auxilary, char
    destination){
    if(disk==1){
        printf("Move disk 1 from %c to %c\n", source, destination);
        return;
    }
    towerOfHanoi(disk-1,source,destination,auxilary);
    printf("Move disk %d from %c to %c\n",disk,source,destination);
    towerOfHanoi(disk-1,auxilary,source,destination);
}
int main(){
    int disk_size;
    printf("Enter number of disks: ");
    scanf("%d",&disk_size);
    towerOfHanoi(disk_size,'A','B','C');
    return 0;
}
```

```
Enter number of disks: 3
Move disk 1 from A to C
Move disk 2 from A to B
Move disk 1 from C to B
Move disk 3 from A to C
Move disk 1 from B to A
Move disk 2 from B to C
Move disk 1 from A to C


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

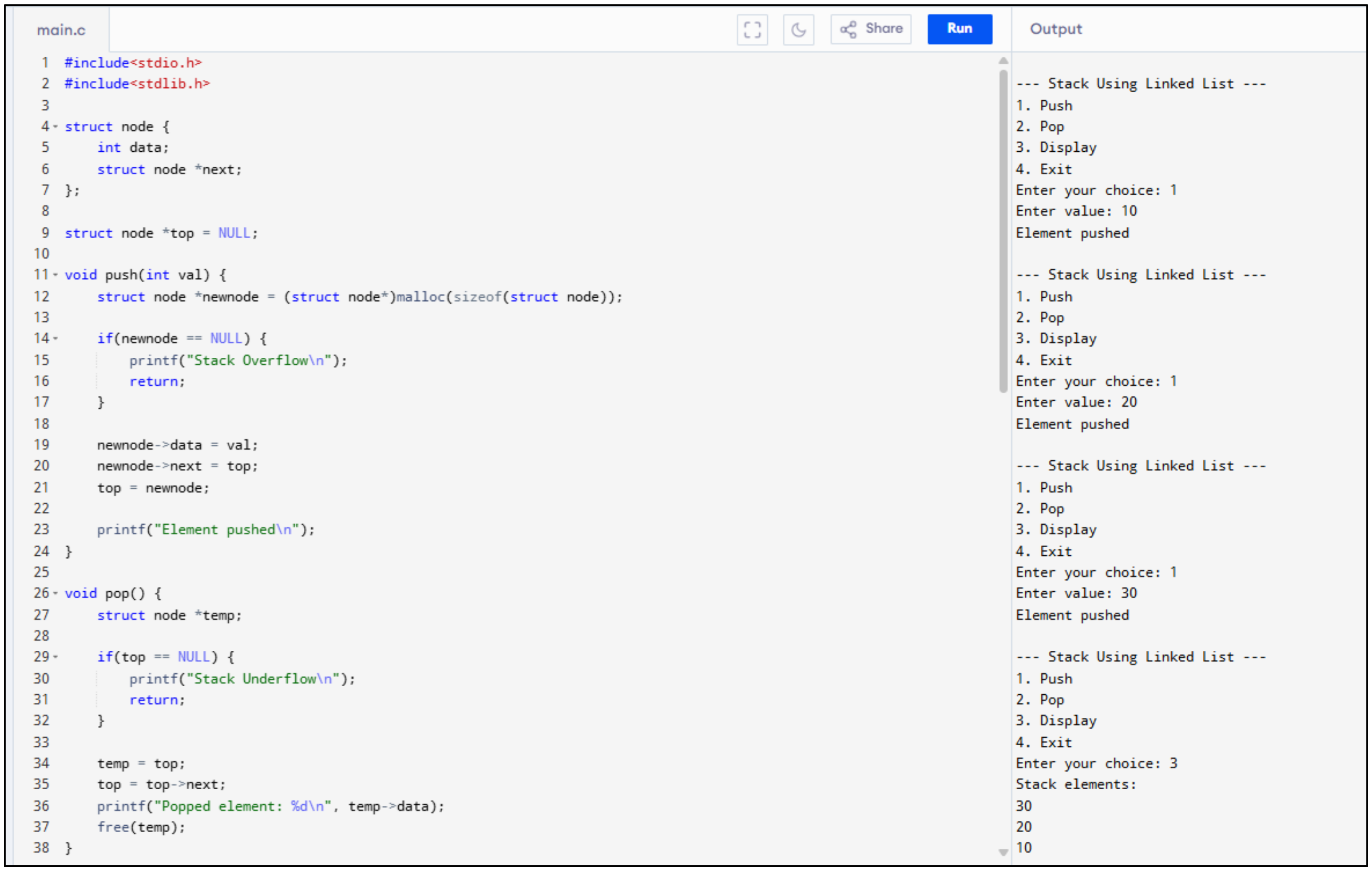
2. Write a Program to calculate the sum of the first N natural numbers using Recursive function.



```c
#include<stdio.h>
int sum(int n){
    if(n==0)
        return 0;
    else
        return n+sum(n-1);
}

int main(){
    int num,res;
    printf("Enter a number: ");
    scanf("%d",&num);
    res=sum(num);
    printf("Sum of %d number is %d",num,res);
    return 0;
}
```

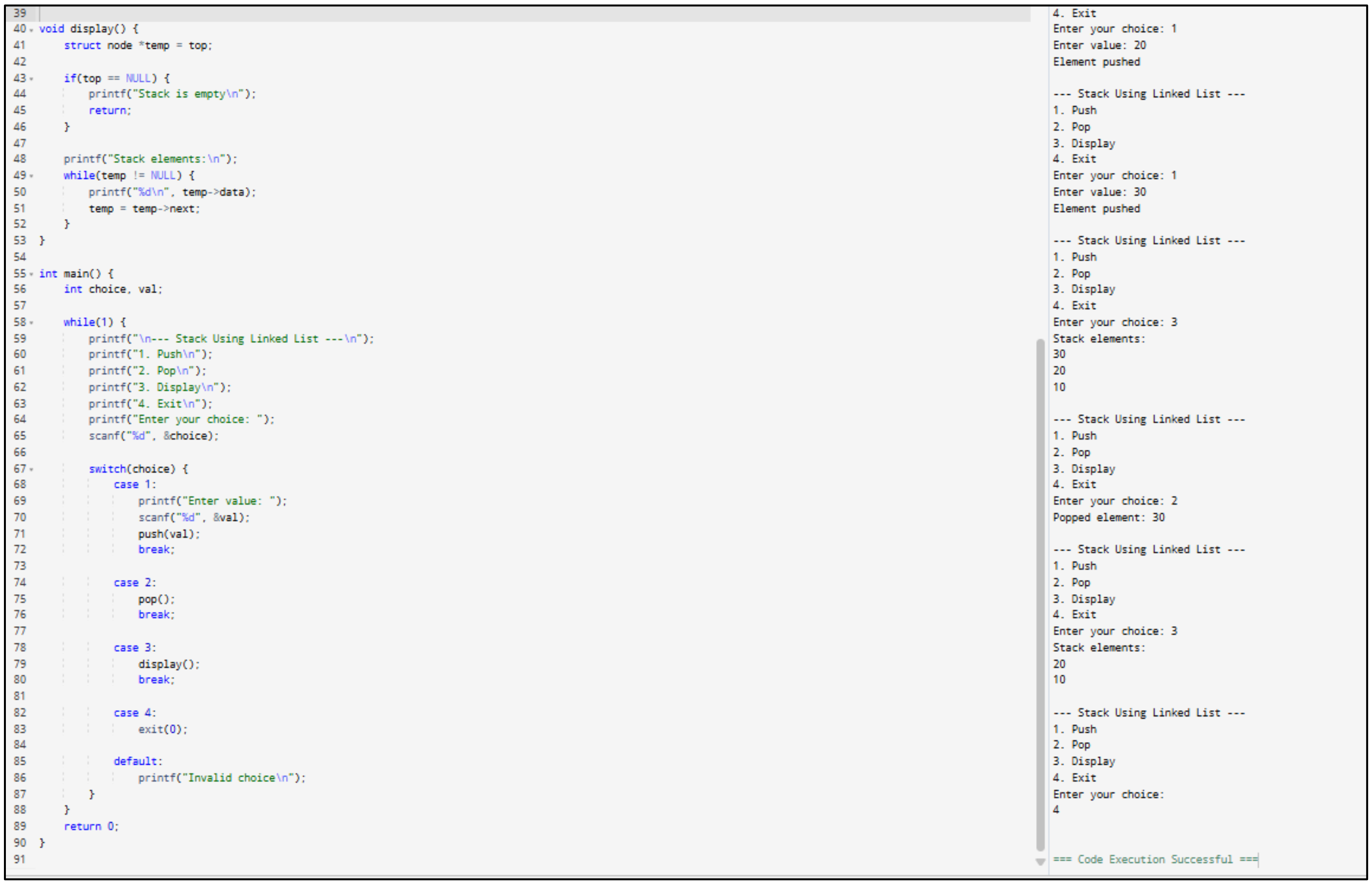
```
Enter a number: 5
Sum of 5 number is 15

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

3. Write a program to implement Insertion (Push) & deletion (Pop) operation using linked list in c.



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

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

struct node *top = NULL;

void push(int val) {
    struct node *newnode = (struct node*)malloc(sizeof(struct node));

    if(newnode == NULL) {
        printf("Stack Overflow\n");
        return;
    }

    newnode->data = val;
    newnode->next = top;
    top = newnode;

    printf("Element pushed\n");
}

void pop() {
    struct node *temp;

    if(top == NULL) {
        printf("Stack Underflow\n");
        return;
    }

    temp = top;
    top = top->next;
    printf("Popped element: %d\n", temp->data);
    free(temp);
}
```

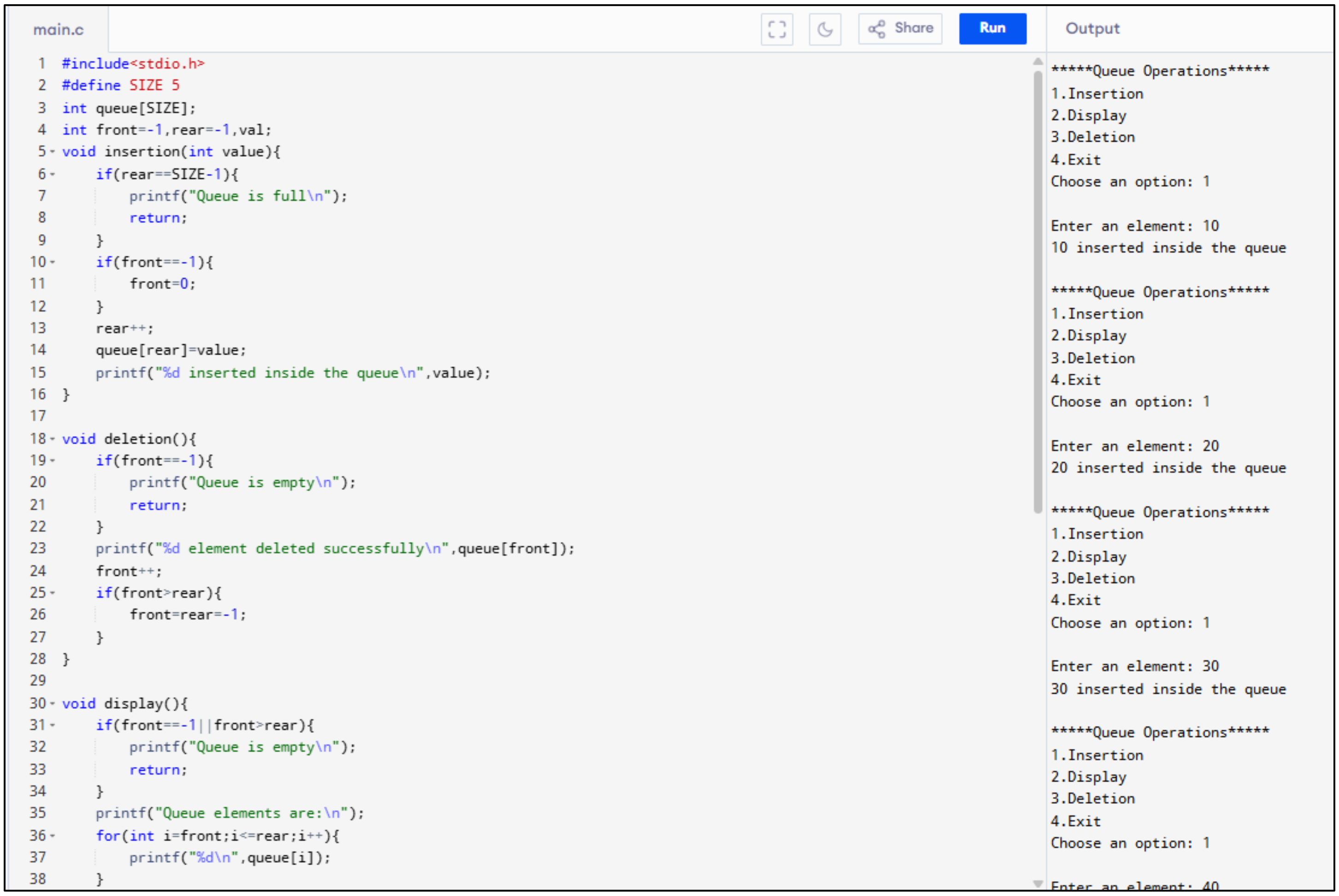
```
--- Stack Using Linked List ---
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter value: 10
Element pushed

--- Stack Using Linked List ---
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter value: 20
Element pushed

--- Stack Using Linked List ---
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter value: 30
Element pushed

--- Stack Using Linked List ---
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3
Stack elements:
30
20
10
```



```c

void display() {
    struct node *temp = top;

    if(top == NULL) {
        printf("Stack is empty\n");
        return;
    }

    printf("Stack elements:\n");
    while(temp != NULL) {
        printf("%d\n", temp->data);
        temp = temp->next;
    }
}

int main() {
    int choice, val;

    while(1) {
        printf("\n--- Stack Using Linked List ---\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: ");
                scanf("%d", &val);
                push(val);
                break;

            case 2:
                pop();
                break;

            case 3:
                display();
                break;

            case 4:
                exit(0);

            default:
                printf("Invalid choice\n");
        }
    }
    return 0;
}

```

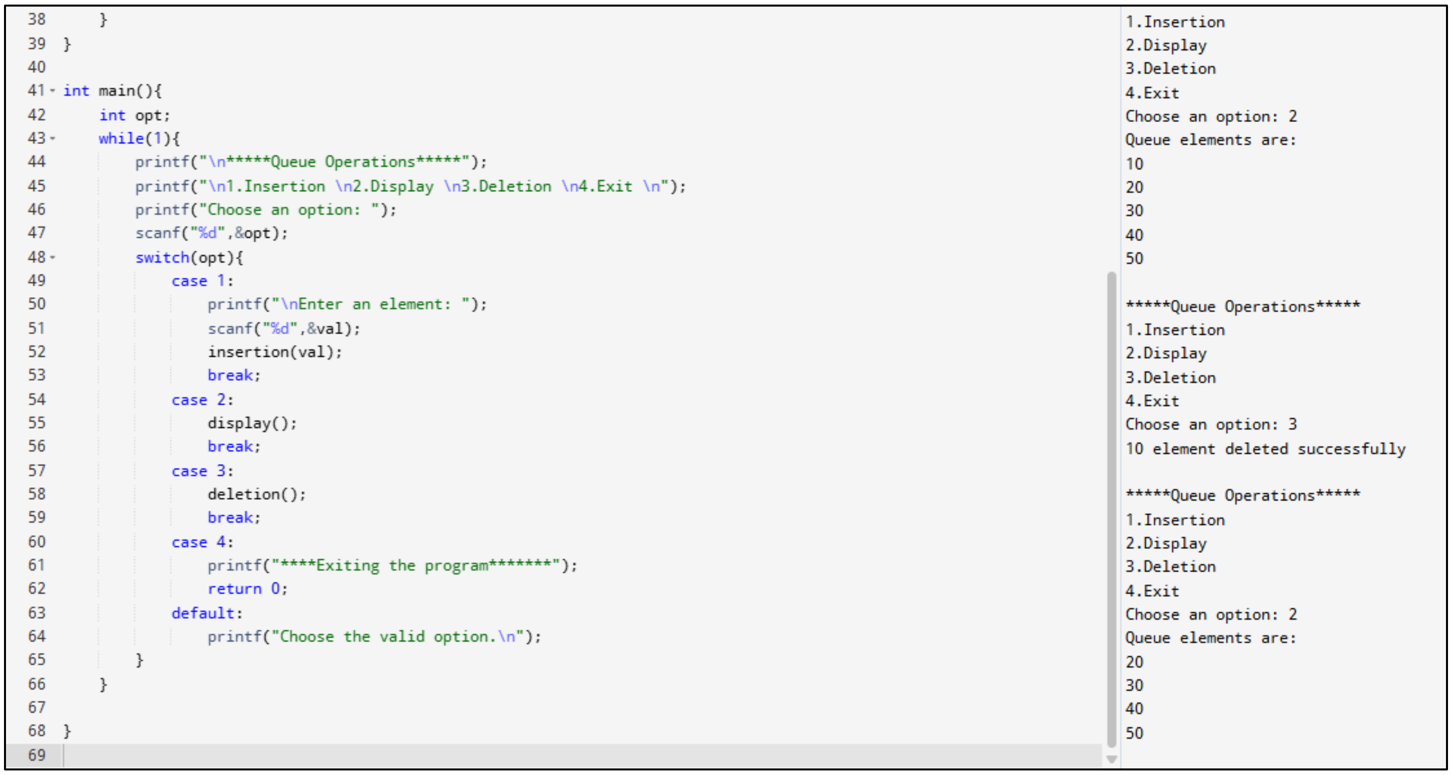
```
4. Exit
Enter your choice: 1
Enter value: 20
Element pushed

--- Stack Using Linked List ---
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter value: 30
Element pushed

--- Stack Using Linked List ---
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3
Stack elements:
30
20
10

--- Stack Using Linked List ---
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2
Popped element: 30

--- Stack Using Linked List ---
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3
Stack elements:
20
10

--- Stack Using Linked List ---
1. Push
2. Pop
3. Display
4. Exit
Enter your choice:
4

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

4. Write a C Program to Implement a queue using array.



```c
#include<stdio.h>
#define SIZE 5
int queue[SIZE];
int front=-1,rear=-1,val;
void insertion(int value){
    if(rear==SIZE-1){
        printf("Queue is full\n");
        return;
    }
    if(front==-1){
        front=0;
    }
    rear++;
    queue[rear]=value;
    printf("%d inserted inside the queue\n",value);
}

void deletion(){
    if(front==-1){
        printf("Queue is empty\n");
        return;
    }
    printf("%d element deleted successfully\n",queue[front]);
    front++;
    if(front>rear){
        front=rear=-1;
    }
}

void display(){
    if(front==-1||front>rear){
        printf("Queue is empty\n");
        return;
    }
    printf("Queue elements are:\n");
    for(int i=front;i<=rear;i++){
        printf("%d\n",queue[i]);
    }
```

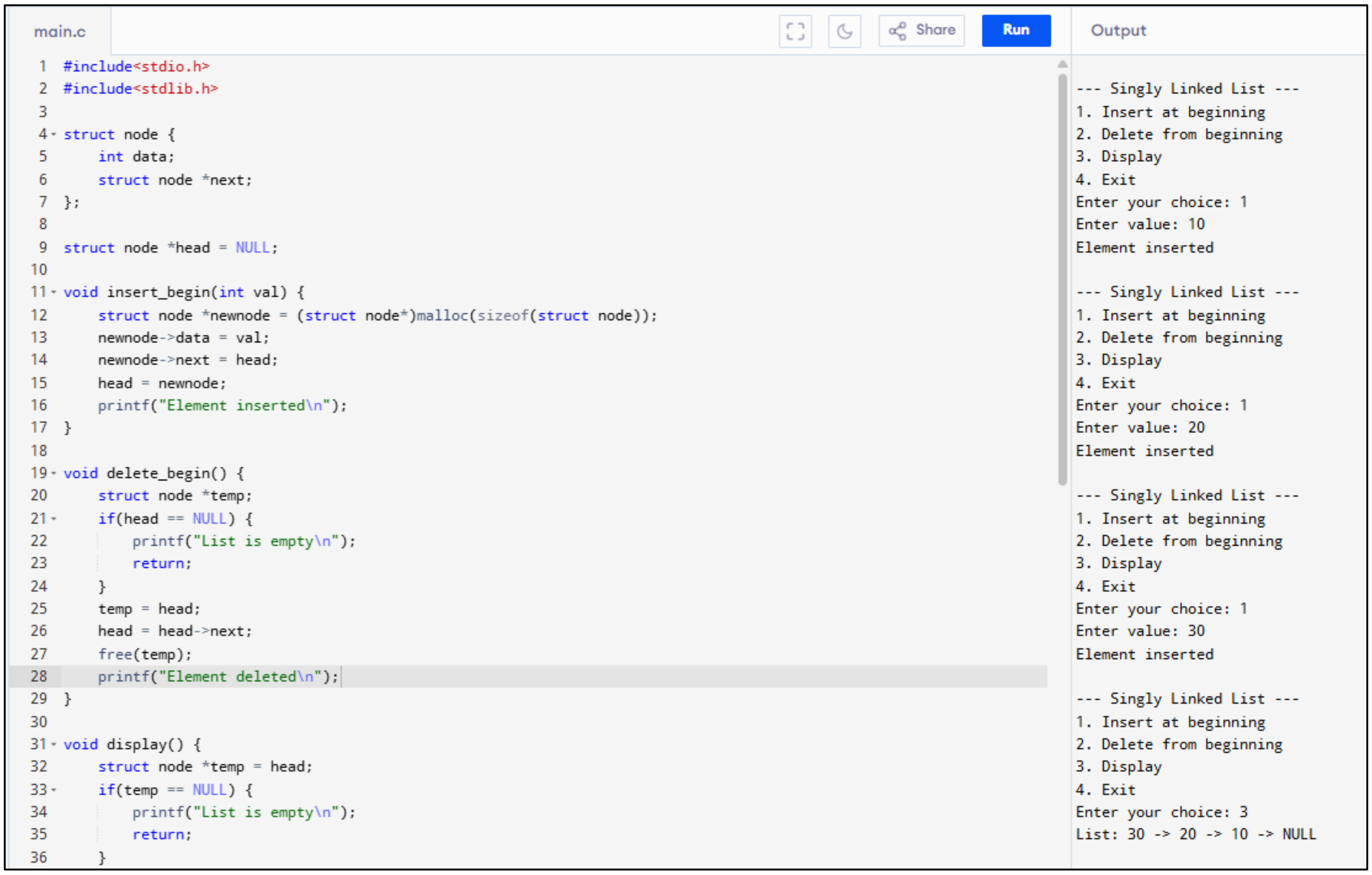
```
*****Queue Operations*****
1.Insertion
2.Display
3.Deletion
4.Exit
Choose an option: 1

Enter an element: 10
10 inserted inside the queue

*****Queue Operations*****
1.Insertion
2.Display
3.Deletion
4.Exit
Choose an option: 1

Enter an element: 20
20 inserted inside the queue

*****Queue Operations*****
1.Insertion
2.Display
3.Deletion
4.Exit
Choose an option: 1

Enter an element: 30
30 inserted inside the queue

*****Queue Operations*****
1.Insertion
2.Display
3.Deletion
4.Exit
Choose an option: 1

Enter an element: 40
```



```c
    }
}

int main(){
    int opt;
    while(1){
        printf("\n*****Queue Operations*****");
        printf("\n1.Insertion \n2.Display \n3.Deletion \n4.Exit \n");
        printf("Choose an option: ");
        scanf("%d",&opt);
        switch(opt){
            case 1:
                printf("\nEnter an element: ");
                scanf("%d",&val);
                insertion(val);
                break;
            case 2:
                display();
                break;
            case 3:
                deletion();
                break;
            case 4:
                printf("****Exiting the program*******");
                return 0;
            default:
                printf("Choose the valid option.\n");
        }
    }
}

```

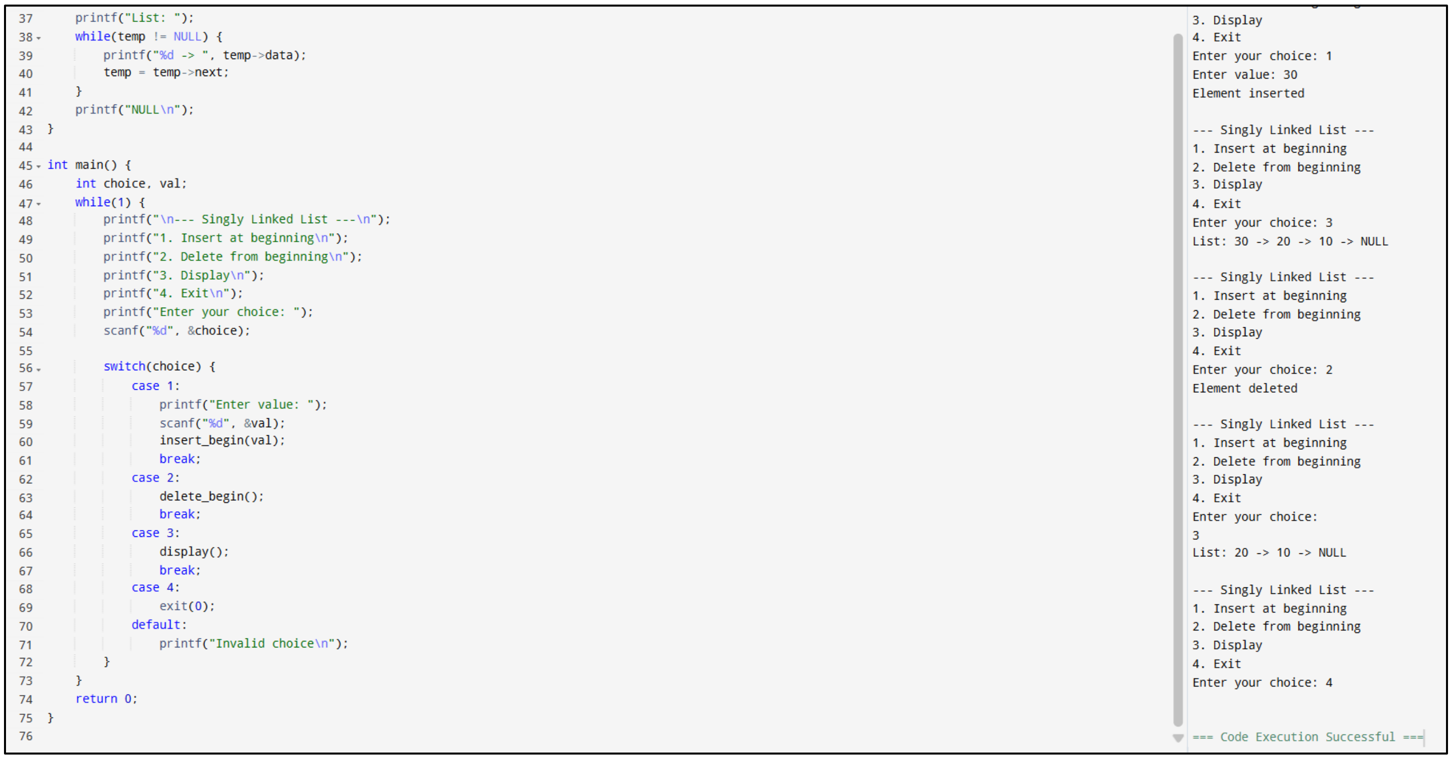
```
1.Insertion
2.Display
3.Deletion
4.Exit
Choose an option: 2
Queue elements are:
10
20
30
40
50

*****Queue Operations*****
1.Insertion
2.Display
3.Deletion
4.Exit
Choose an option: 3
10 element deleted successfully

*****Queue Operations*****
1.Insertion
2.Display
3.Deletion
4.Exit
Choose an option: 2
Queue elements are:
20
30
40
50
```

5. Write a program to implement Singly Linked List.



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

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

struct node *head = NULL;

void insert_begin(int val) {
    struct node *newnode = (struct node*)malloc(sizeof(struct node));
    newnode->data = val;
    newnode->next = head;
    head = newnode;
    printf("Element inserted\n");
}

void delete_begin() {
    struct node *temp;
    if(head == NULL) {
        printf("List is empty\n");
        return;
    }
    temp = head;
    head = head->next;
    free(temp);
    printf("Element deleted\n");
}

void display() {
    struct node *temp = head;
    if(temp == NULL) {
        printf("List is empty\n");
        return;
    }
```

```
--- Singly Linked List ---
1. Insert at beginning
2. Delete from beginning
3. Display
4. Exit
Enter your choice: 1
Enter value: 10
Element inserted

--- Singly Linked List ---
1. Insert at beginning
2. Delete from beginning
3. Display
4. Exit
Enter your choice: 1
Enter value: 20
Element inserted

--- Singly Linked List ---
1. Insert at beginning
2. Delete from beginning
3. Display
4. Exit
Enter your choice: 1
Enter value: 30
Element inserted

--- Singly Linked List ---
1. Insert at beginning
2. Delete from beginning
3. Display
4. Exit
Enter your choice: 3
List: 30 -> 20 -> 10 -> NULL
```



```c
    printf("List: ");
    while(temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

int main() {
    int choice, val;
    while(1) {
        printf("\n--- Singly Linked List ---\n");
        printf("1. Insert at beginning\n");
        printf("2. Delete from beginning\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch(choice) {
            case 1:
                printf("Enter value: ");
                scanf("%d", &val);
                insert_begin(val);
                break;
            case 2:
                delete_begin();
                break;
            case 3:
                display();
                break;
            case 4:
                exit(0);
            default:
                printf("Invalid choice\n");
        }
    }
    return 0;
}
```

```
3. Display
4. Exit
Enter your choice: 1
Enter value: 30
Element inserted

--- Singly Linked List ---
1. Insert at beginning
2. Delete from beginning
3. Display
4. Exit
Enter your choice: 3
List: 30 -> 20 -> 10 -> NULL

--- Singly Linked List ---
1. Insert at beginning
2. Delete from beginning
3. Display
4. Exit
Enter your choice: 2
Element deleted

--- Singly Linked List ---
1. Insert at beginning
2. Delete from beginning
3. Display
4. Exit
Enter your choice:
3
List: 20 -> 10 -> NULL

--- Singly Linked List ---
1. Insert at beginning
2. Delete from beginning
3. Display
4. Exit
Enter your choice: 4


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