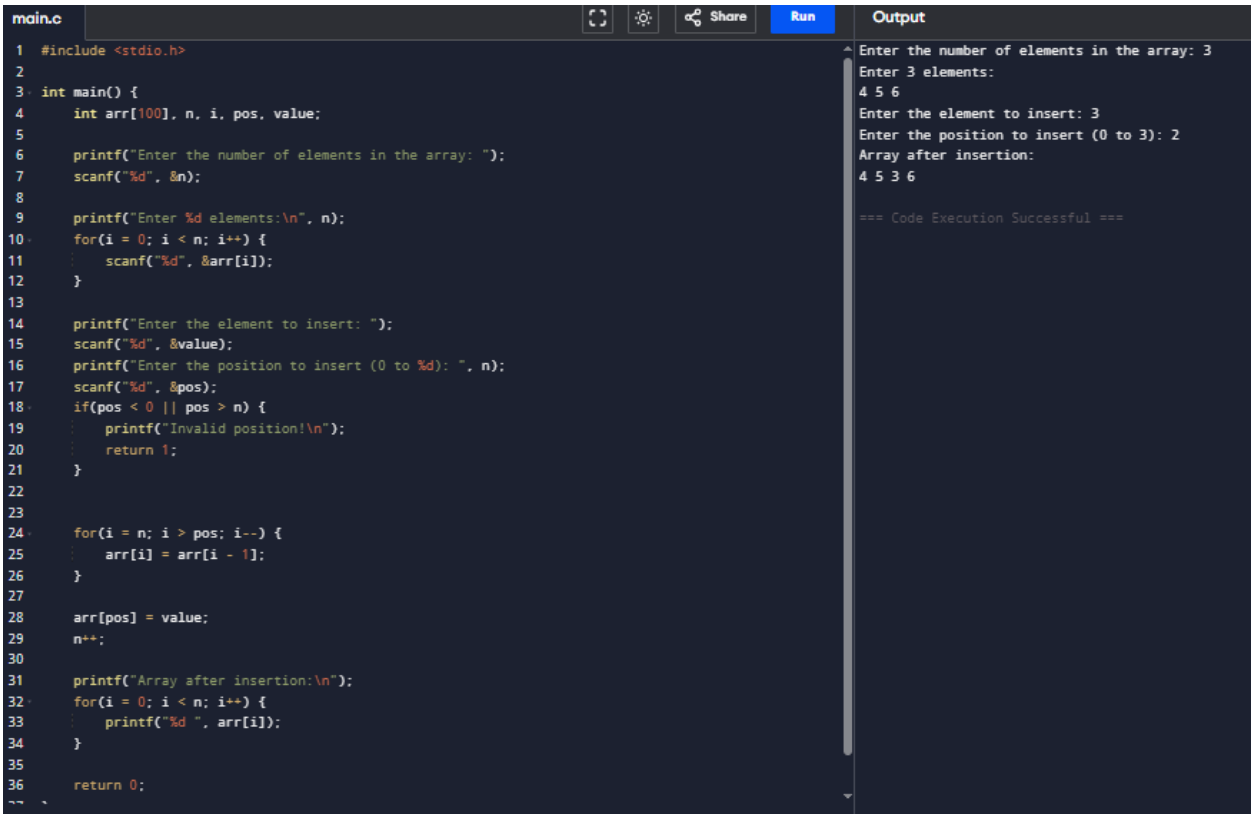
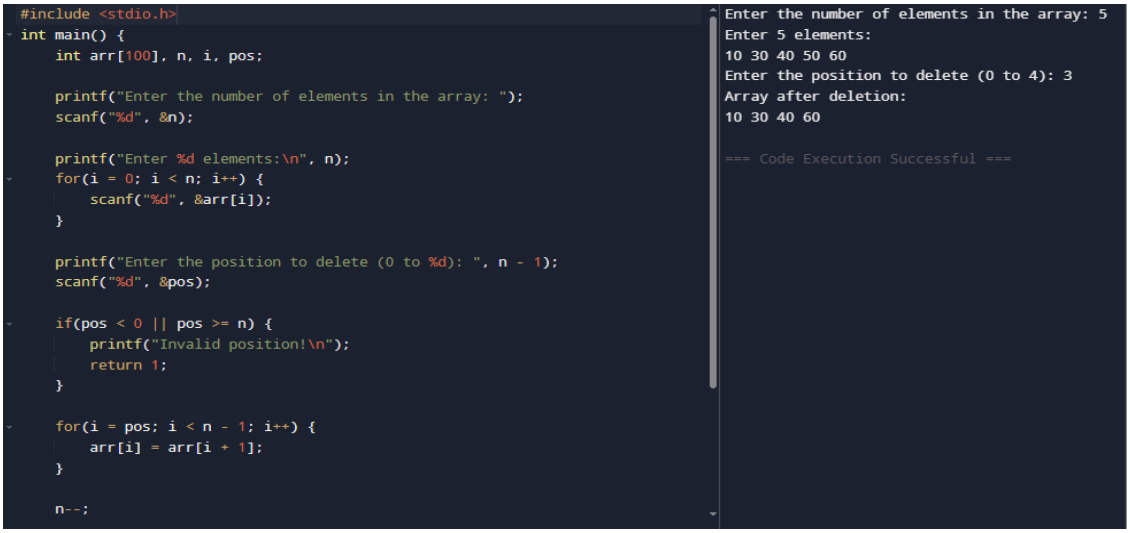
Data structures lab program

1. Write a program in C to implement insertion in 1-D Arrays



2. Write a program in C to implement deletion in 1-D Arrays



3. Write a program in C to implement linear and binary searching in 1-D Arrays

```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 n, int key) {
    int low = 0, high = n - 1, mid;
    while(low <= high) {
        mid = (low + high) / 2;
        if(arr[mid] == key)
            return mid;
        else if(arr[mid] < key)
            low = mid + 1;
        else
            high = mid - 1;
    }
    return -1;
}

int main() {
    int arr[100], n, key, choice, i, result;


    printf("Enter number of elements: ");
    scanf("%d", &n);
    printf("Enter %d elements:\n", n);
    for(i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    printf("Enter the element to search: ");
    scanf("%d", &key);
    printf("Choose search method:\n");
    printf("1. Linear Search\n");
    printf("2. Binary Search (array must be sorted)\n");
```

```
Enter number of elements: 4
Enter 4 elements:
30 50 40 60
Enter the element to search: 30
Choose search method:
1. Linear Search
2. Binary Search (array must be sorted)
2
Element found at index 0


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

4. Write a program in C to implement sorting in 1-D Arrays

main.c Share Run Output

```c
#include <stdio.h>

int main() {
    int arr[100], n, i, j, temp;

    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]);
    }

    for(i = 0; i < n - 1; i++) {
        for(j = 0; j < n - i - 1; j++) {
            if(arr[j] > arr[j + 1]) {
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }


    printf("Sorted array in ascending order:\n");
    for(i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }

    return 0;
}
```

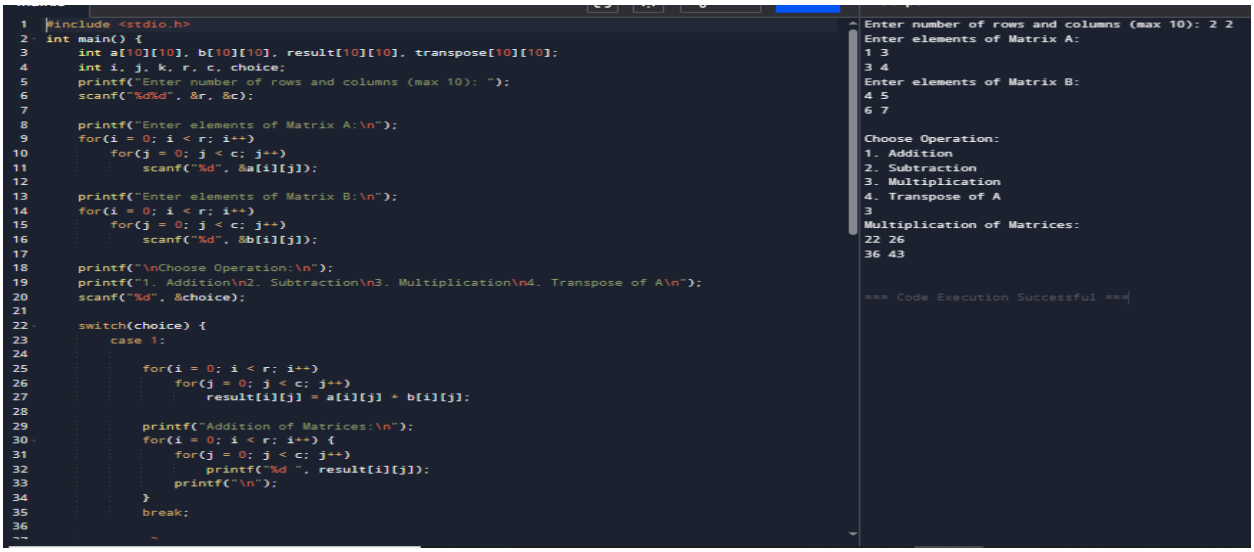
```
Enter the number of elements: 5
Enter 5 elements:
50 60 40 30 20
Sorted array in ascending order:
20 30 40 50 60

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

5. Write a program in C to concatenate two arrays

```c
#include <stdio.h>

int main() {
    int arr1[100], arr2[100], arr3[200];
    int n1, n2, i, j;

    printf("Enter the number of elements in the first array: ");
    scanf("%d", &n1);
    printf("Enter %d elements for the first array:\n", n1);
    for(i = 0; i < n1; i++) {
        scanf("%d", &arr1[i]);
    }

    printf("Enter the number of elements in the second array: ");
    scanf("%d", &n2);
    printf("Enter %d elements for the second array:\n", n2);
    for(i = 0; i < n2; i++) {
        scanf("%d", &arr2[i]);
    }

    for(i = 0; i < n1; i++) {
        arr3[i] = arr1[i];
    }

    for(j = 0; j < n2; j++) {
        arr3[i + j] = arr2[j];
    }

    printf("Concatenated array:\n");
    for(i = 0; i < n1 + n2; i++) {
        printf("%d ", arr3[i]);
    }

    return 0;
}
```

```
Enter the number of elements in the first array: 3
Enter 3 elements for the first array:
2 4 5
Enter the number of elements in the second array: 4
Enter 4 elements for the second array:
5 7 8 9
Concatenated array:
2 4 5 5 7 8 9

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

6. Write a program in C to implement the following Operations on 2-D Array (addition; subtraction; multiplication; transpose)



```c
#include <stdio.h>
int main() {
    int a[10][10], b[10][10], result[10][10], transpose[10][10];
    int i, j, k, r, c, choice;
    printf("Enter number of rows and columns (max 10): ");
    scanf("%d%d", &r, &c);

    printf("Enter elements of Matrix A:\n");
    for(i = 0; i < r; i++)
        for(j = 0; j < c; j++)
            scanf("%d", &a[i][j]);

    printf("Enter elements of Matrix B:\n");
    for(i = 0; i < r; i++)
        for(j = 0; j < c; j++)
            scanf("%d", &b[i][j]);

    printf("\nChoose Operation:\n");
    printf("1. Addition\n2. Subtraction\n3. Multiplication\n4. Transpose of A\n");
    scanf("%d", &choice);

    switch(choice) {
        case 1:

            for(i = 0; i < r; i++)
                for(j = 0; j < c; j++)
                    result[i][j] = a[i][j] + b[i][j];

            printf("Addition of Matrices:\n");
            for(i = 0; i < r; i++) {
                for(j = 0; j < c; j++)
                    printf("%d ", result[i][j]);
                printf("\n");
            }
            break;
```

```
Enter number of rows and columns (max 10): 2 2
Enter elements of Matrix A:
1 3
3 4
Enter elements of Matrix B:
4 5
6 7

Choose Operation:
1. Addition
2. Subtraction
3. Multiplication
4. Transpose of A
3
Multiplication of Matrices:
22 26
36 43


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

```c
    case 2:
        for(i = 0; i < r; i++)
            for(j = 0; j < c; j++)
                result[i][j] = a[i][j] - b[i][j];

        printf("Subtraction of Matrices:\n");
        for(i = 0; i < r; i++) {
            for(j = 0; j < c; j++)
                printf("%d ", result[i][j]);
            printf("\n");
        }
        break;

    case 3:

        for(i = 0; i < r; i++) {
            for(j = 0; j < c; j++) {
                result[i][j] = 0;
                for(k = 0; k < c; k++)
                    result[i][j] += a[i][k] * b[k][j];
            }
        }

        printf("Multiplication of Matrices:\n");
        for(i = 0; i < r; i++) {
            for(j = 0; j < c; j++)
                printf("%d ", result[i][j]);
            printf("\n");
        }
        break;

    case 4:

        for(i = 0; i < r; i++)
            for(j = 0; j < c; j++)
                transpose[j][i] = a[i][j];
```

```
Enter number of rows and columns (max 10): 2 2
Enter elements of Matrix A:
1 3
3 4
Enter elements of Matrix B:
4 5
6 7

Choose Operation:
1. Addition
2. Subtraction
3. Multiplication
4. Transpose of A
3
Multiplication of Matrices:
22 26
36 43


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

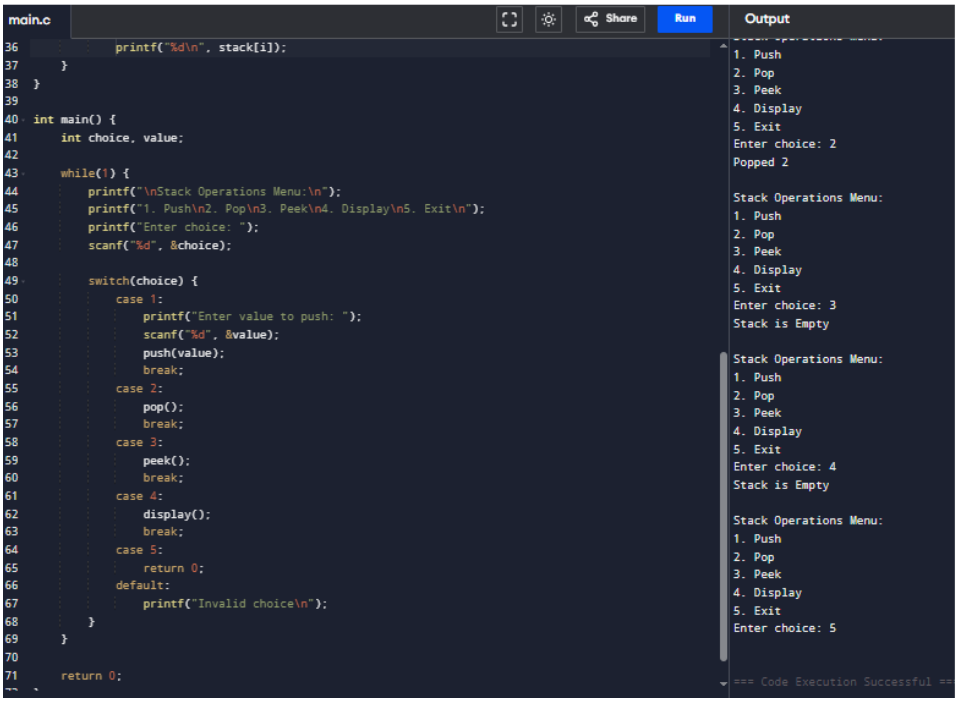
7. Write a program in C to implement operations on Stack using array

main.c

```c
#include <stdio.h>
#define SIZE 100

int stack[SIZE], top = -1;
void push(int value) {
    if(top == SIZE - 1)
        printf("Stack Overflow\n");
    else {
        top++;
        stack[top] = value;
        printf("Pushed %d\n", value);
    }
}
void pop() {
    if(top == -1)
        printf("Stack Underflow\n");
    else {
        printf("Popped %d\n", stack[top]);
        top--;
    }
}
void peek() {
    if(top == -1)
        printf("Stack is Empty\n");
    else
        printf("Top element is %d\n", stack[top]);
}


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

Output

```
Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter choice: 1
Enter value to push: 2
Pushed 2

Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter choice: 2
Popped 2

Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter choice: 3
Stack is Empty

Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter choice: 4
```



8. Write a program in C to implement operations on Stack using

linked list

main.c

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

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

struct Node* top = NULL;
void push(int value) {
    struct Node* newNode = (struct Node*) malloc(sizeof(struct Node));
    if(newNode == NULL) {
        printf("Stack Overflow\n");
        return;
    }
    newNode->data = value;
    newNode->next = top;
    top = newNode;
    printf("Pushed %d\n", value);
}

void pop() {
    if(top == NULL) {
        printf("Stack Underflow\n");
        return;
    }
    struct Node* temp = top;
    printf("Popped %d\n", top->data);
    top = top->next;
    free(temp);
}

void peek() {
    if(top == NULL)
        printf("Stack is Empty\n");
    else
```

Output

```
Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter choice: 10
Invalid choice

Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter choice: 2
Stack Underflow

Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter choice: 3
Stack is Empty

Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Display
5. Exit
Enter choice: 4
Stack is Empty
```

9. Write a program in C to implement applications of Stack

main.c

Share Run

Output

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

#define SIZE 100

char stack[SIZE];
int top = -1;

void push(char ch) {
    if(top < SIZE - 1)
        stack[++top] = ch;
}

char pop() {
    if(top != -1)
        return stack[top--];
    return '\0';
}

int isMatchingPair(char open, char close) {
    return (open == '(' && close == ')') ||
           (open == '[' && close == ']') ||
           (open == '{' && close == '}');
}

int isBalanced(char* expr) {
    for(int i = 0; i < strlen(expr); i++) {
        char ch = expr[i];
        if(ch == '(' || ch == '{' || ch == '[') {
            push(ch);
        } else if(ch == ')' || ch == '}' || ch == ']') {
            char topChar = pop();
            if(!isMatchingPair(topChar, ch))
                return 0;
        }
    }
    return top == -1;
}

int main() {
    char expr[SIZE];

    printf("Enter an expression: ");
    scanf("%s", expr);

    if(isBalanced(expr))
        printf("The expression is balanced.\n");
    else
        printf("The expression is NOT balanced.\n");

    return 0;
}
```

```
Enter an expression: (a+b)*(c-d)
The expression is balanced.


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

10. Write a program in C to implement operations on queue using array

```c
#include <stdio.h>
#define SIZE 100

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

void enqueue(int value) {
    if(rear == SIZE - 1) {
        printf("Queue Overflow\n");
    } else {
        if(front == -1)
            front = 0;
        rear++;
        queue[rear] = value;
        printf("Enqueued %d\n", value);
    }
}
void dequeue() {
    if(front == -1 || front > rear) {
        printf("Queue Underflow\n");
    } else {
        printf("Dequeued %d\n", queue[front]);
        front++;
    }
}
void display() {
    if(front == -1 || front > rear) {
        printf("Queue is Empty\n");
    } else {
        printf("Queue elements:\n");
        for(int i = front; i <= rear; i++)
            printf("%d ", queue[i]);
        printf("\n");
    }
}
```

```
Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 10
Enqueued 10

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 20
Enqueued 20

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue elements:
10 20

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 4
```

main.c

Share Run

Output

```c
            printf("%d ", queue[i]);
        printf("\n");
    }
}

int main() {
    int choice, value;

    while(1) {
        printf("\nQueue Operations Menu:\n");
        printf("1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch(choice) {
            case 1:
                printf("Enter value to enqueue: ");
                scanf("%d", &value);
                enqueue(value);
                break;
            case 2:
                dequeue();
                break;
            case 3:
                display();
                break;
            case 4:
                return 0;
            default:
                printf("Invalid choice\n");
        }
    }

    return 0;
}
```

```
Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 10
Enqueued 10

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 20
Enqueued 20

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue elements:
10 20

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 4


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

11. Write a program in C to implement operations on queue using linked list

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

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

struct Node* front = NULL;
struct Node* rear = NULL;

void enqueue(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed\n");
        return;
    }
    newNode->data = value;
    newNode->next = NULL;
    if (rear == NULL) {
        front = rear = newNode;
    } else {
        rear->next = newNode;
        rear = newNode;
    }
    printf("Enqueued %d\n", value);
}
void dequeue() {
    if (front == NULL) {
        printf("Queue Underflow\n");
        return;
    }
    struct Node* temp = front;
    printf("Dequeued %d\n", front->data);
    front = front->next;
    if (front == NULL)
```

```
Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 10
Enqueued 10

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Dequeued 10

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 10
Invalid choice

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue is Empty

Queue Operations Menu:
1. Enqueue
```

main.c

Share Run

```c
    if (front == NULL)
        rear = NULL;
    free(temp);
}

void display() {
    if (front == NULL) {
        printf("Queue is Empty\n");
        return;
    }
    struct Node* temp = front;
    printf("Queue elements:\n");
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    int choice, value;

    while (1) {
        printf("\nQueue Operations Menu:\n");
        printf("1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to enqueue: ");
                scanf("%d", &value);
                enqueue(value);
                break;
            case 2:
                dequeue();
```

Output

```
Enter value to enqueue: 10
Enqueued 10

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Dequeued 10

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 10
Invalid choice

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue is Empty

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 4


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

```c
        printf("\nQueue Operations Menu:\n");
        printf("1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to enqueue: ");
                scanf("%d", &value);
                enqueue(value);
                break;
            case 2:
                dequeue();
                break;
            case 3:
                display();
                break;
            case 4:
                return 0;
            default:
                printf("Invalid choice\n");
        }
    }

    return 0;
}
```

12. Write a program in C to implement operations on circular queue using array

main.c

Share Run

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

int queue[SIZE];
int front = -1, rear = -1;
void enqueue(int value) {
    if ((front == 0 && rear == SIZE - 1) || (rear + 1) % SIZE == front) {
        printf("Queue Overflow\n");
        return;
    }
    if (front == -1)
        front = rear = 0;
    else
        rear = (rear + 1) % SIZE;

    queue[rear] = value;
    printf("Enqueued %d\n", value);
}


void dequeue() {
    if (front == -1) {
        printf("Queue Underflow\n");
        return;
    }
    printf("Dequeued %d\n", queue[front]);

    if (front == rear)
        front = rear = -1;
    else
        front = (front + 1) % SIZE;
}

void display() {
    if (front == -1) {
        printf("Queue is Empty\n");
```

Output

```
Circular Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 10
Enqueued 10

Circular Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Dequeued 10

Circular Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Queue Underflow

Circular Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue is Empty

Circular Queue Menu:
1. Enqueue
```

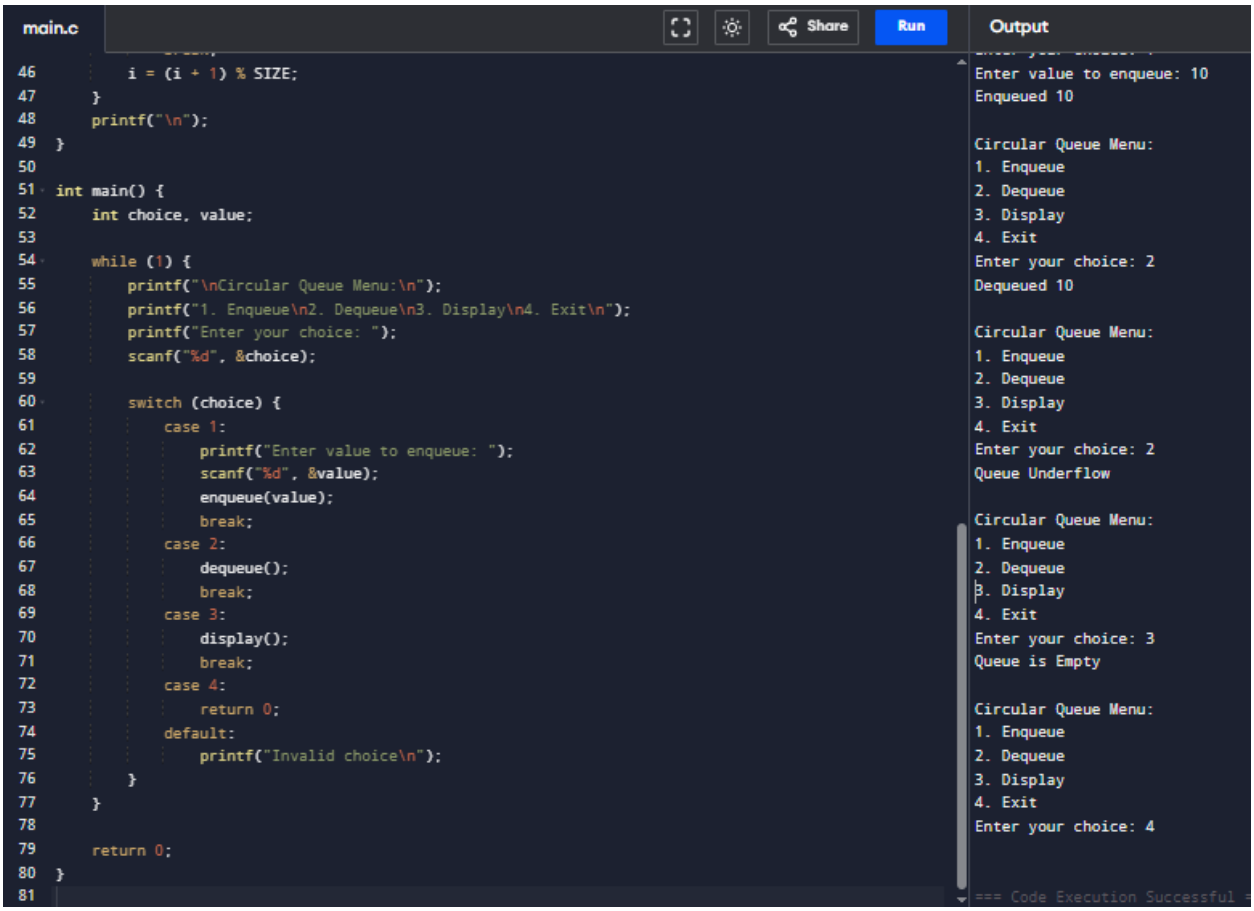
main.c

Share Run

```c
    if (front == -1) {
        printf("Queue is Empty\n");
        return;
    }

    printf("Queue elements:\n");
    int i = front;
    while (1) {
        printf("%d ", queue[i]);
        if (i == rear)
            break;
        i = (i + 1) % SIZE;
    }
    printf("\n");
}

int main() {
    int choice, value;

    while (1) {
        printf("\nCircular Queue Menu:\n");
        printf("1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

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

Output

```
Enter value to enqueue: 10
Enqueued 10

Circular Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Dequeued 10

Circular Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Queue Underflow

Circular Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue is Empty

Circular Queue Menu:
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 4


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



13. Write a program in C to implement insertion in a linked list(beg; mid; end)

main.c

Share Run

```c
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};

struct Node* head = NULL;
void insertAtBeginning(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = head;
    head = newNode;
    printf("Inserted %d at beginning\n", value);
}

void insertAtEnd(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = NULL;

    if (head == NULL) {
        head = newNode;
    } else {
        struct Node* temp = head;
        while (temp->next != NULL)
            temp = temp->next;
        temp->next = newNode;
    }
    printf("Inserted %d at end\n", value);
}

void insertAtPosition(int pos, int value) {
    if (pos == 0) {
        insertAtBeginning(value);
        return;
```

Output

```
Linked List Insertion Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Display
5. Exit
Enter choice: 1
Enter value: 10
Inserted 10 at beginning

Linked List Insertion Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Display
5. Exit
Enter choice: 2
Enter value: 20
Inserted 20 at end

Linked List Insertion Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Display
5. Exit
Enter choice: 3
Enter position (starting from 0): 30
Enter value: 4
Invalid position

Linked List Insertion Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
```

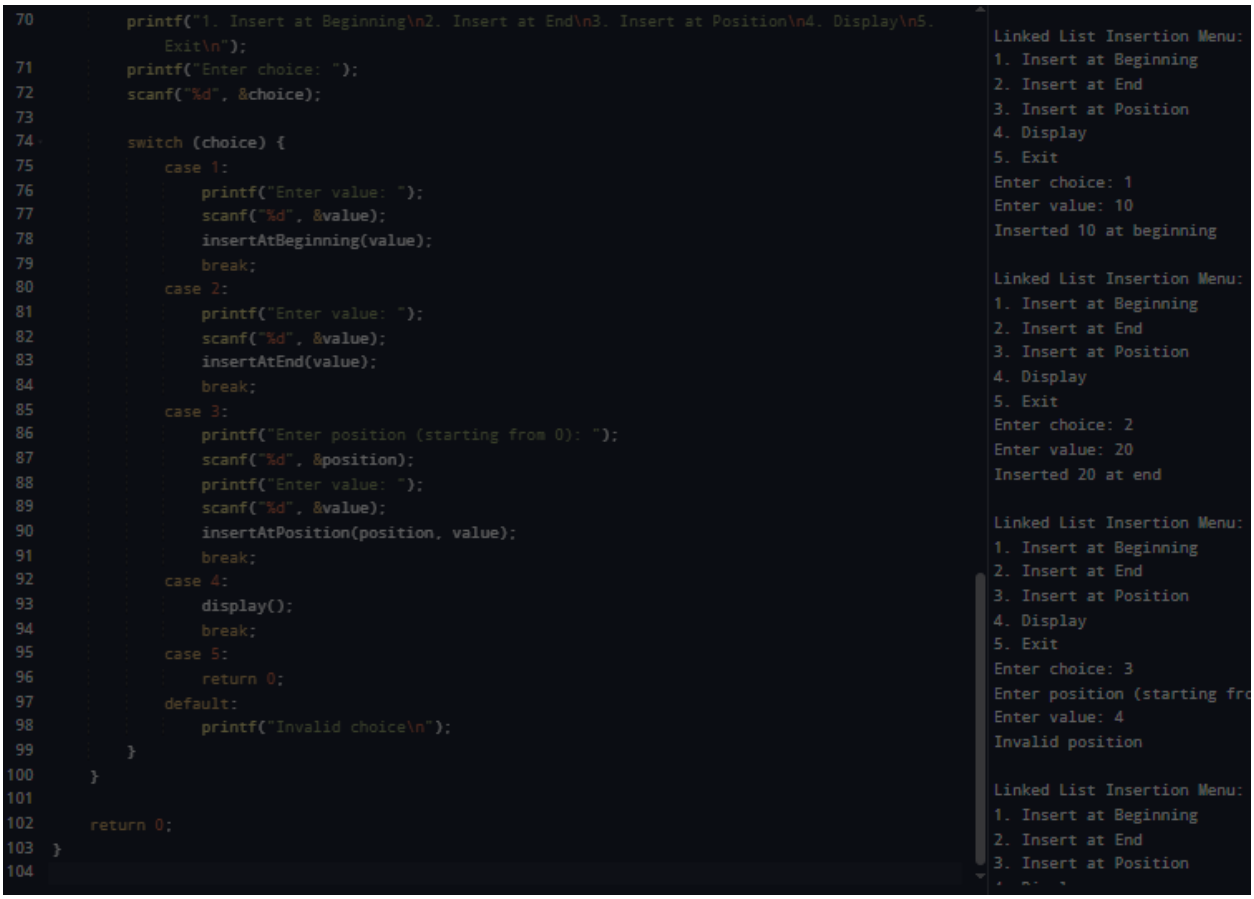
main.c

Share Run

```c
    if (pos == 0) {
        insertAtBeginning(value);
        return;
    }

    struct Node* temp = head;
    for (int i = 0; i < pos - 1 && temp != NULL; i++)
        temp = temp->next;

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

    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = temp->next;
    temp->next = newNode;
    printf("Inserted %d at position %d\n", value, pos);
}

void display() {
    struct Node* temp = head;
    printf("Linked List: ");
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

int main() {
    int choice, value, position;

    while (1) {
        printf("\nLinked List Insertion Menu:\n");
```

Output

```
Linked List Insertion Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Display
5. Exit
Enter choice: 1
Enter value: 10
Inserted 10 at beginning

Linked List Insertion Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Display
5. Exit
Enter choice: 2
Enter value: 20
Inserted 20 at end

Linked List Insertion Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Display
5. Exit
Enter choice: 3
Enter position (starting from 0): 30
Enter value: 4
Invalid position

Linked List Insertion Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
```



```c
        printf("1. Insert at Beginning\n2. Insert at End\n3. Insert at Position\n4. Display\n5.
            Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value: ");
                scanf("%d", &value);
                insertAtBeginning(value);
                break;
            case 2:
                printf("Enter value: ");
                scanf("%d", &value);
                insertAtEnd(value);
                break;
            case 3:
                printf("Enter position (starting from 0): ");
                scanf("%d", &position);
                printf("Enter value: ");
                scanf("%d", &value);
                insertAtPosition(position, value);
                break;
            case 4:
                display();
                break;
            case 5:
                return 0;
            default:
                printf("Invalid choice\n");
        }
    }

    return 0;
}
```

```
Linked List Insertion Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Display
5. Exit
Enter choice: 1
Enter value: 10
Inserted 10 at beginning

Linked List Insertion Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Display
5. Exit
Enter choice: 2
Enter value: 20
Inserted 20 at end

Linked List Insertion Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Display
5. Exit
Enter choice: 3
Enter position (starting fro
Enter value: 4
Invalid position

Linked List Insertion Menu:
1. Insert at Beginning
2. Insert at End
3. Insert at Position
```

14. Write a program in C to implement deletion from a linked list(beg; mid; end)

main.c

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

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

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

void deleteBeginning(struct Node** head) {
    if (*head == NULL) return;
    struct Node* temp = *head;
    *head = (*head)->next;
    free(temp);
}

void deleteEnd(struct Node** head) {
    if (*head == NULL) return;
    if ((*head)->next == NULL) {
        free(*head);
        *head = NULL;
        return;
    }
    struct Node* temp = *head;
    while (temp->next->next != NULL)
        temp = temp->next;
    free(temp->next);
    temp->next = NULL;
```

Output

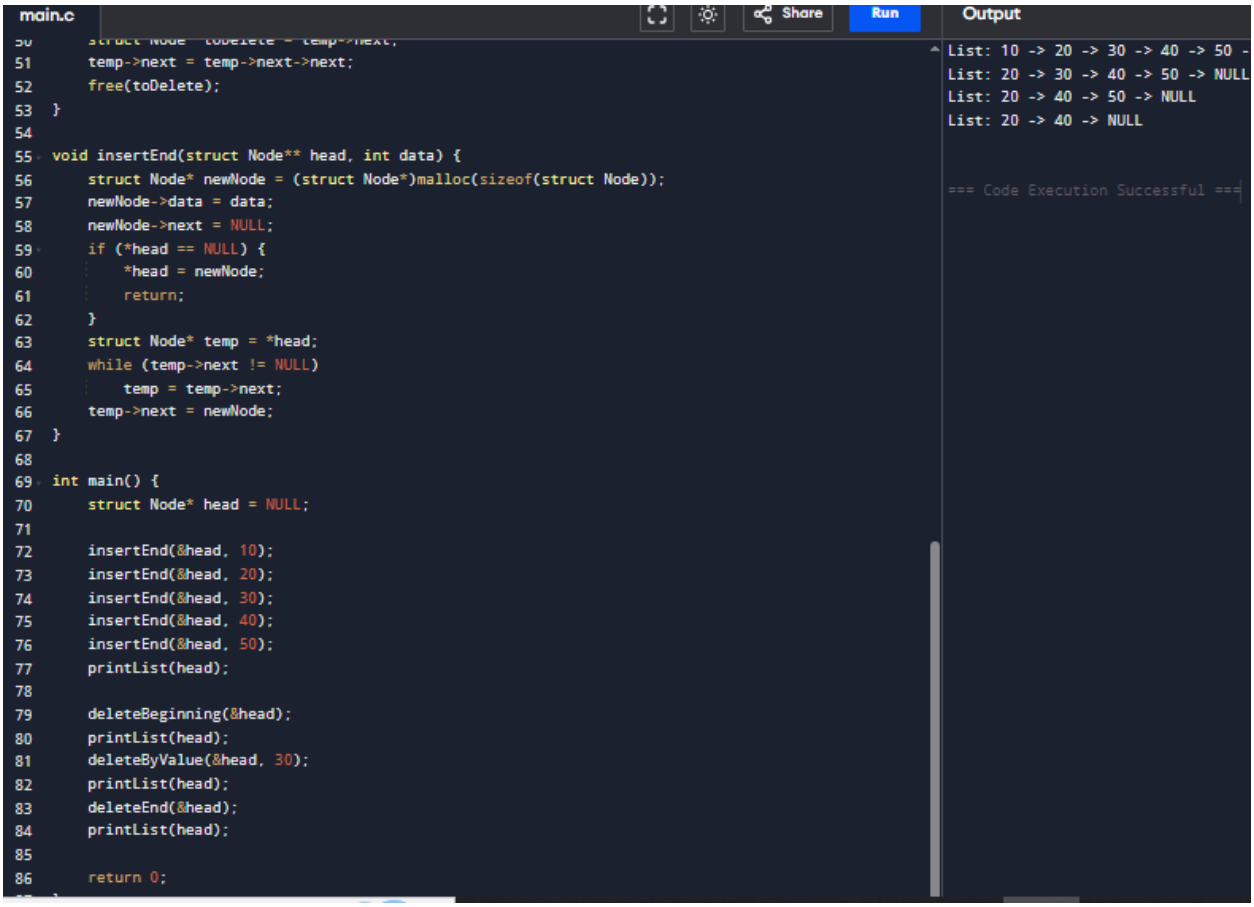
```
List: 10 -> 20 -> 30 -> 40 -> 50 -> NULL
List: 20 -> 30 -> 40 -> 50 -> NULL
List: 20 -> 40 -> 50 -> NULL
List: 20 -> 40 -> NULL

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

main.c

```c
}

void deleteByValue(struct Node** head, int value) {
    if (*head == NULL) return;
    if ((*head)->data == value) {
        deleteBeginning(head);
        return;
    }
    struct Node* temp = *head;
    while (temp->next != NULL && temp->next->data != value)
        temp = temp->next;
    if (temp->next == NULL) return;
    struct Node* toDelete = temp->next;
    temp->next = temp->next->next;
    free(toDelete);
}

void insertEnd(struct Node** head, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    struct Node* temp = *head;
    while (temp->next != NULL)
        temp = temp->next;
    temp->next = newNode;
}

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

    insertEnd(&head, 10);
    insertEnd(&head, 20);
```

Output

```
List: 10 -> 20 -> 30 -> 40 -> 50 -> NULL
List: 20 -> 30 -> 40 -> 50 -> NULL
List: 20 -> 40 -> 50 -> NULL
List: 20 -> 40 -> NULL

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



15. Write a program in C to implement insertion in a circular linked list(beg; mid; end)

main.c

Share Run

```c
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

void insertEnd(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        newNode->next = newNode;
        *head = newNode;
        return;
    }

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

    temp->next = newNode;
    newNode->next = *head;
}

void insertBegin(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        newNode->next = newNode;
        *head = newNode;
        return;
    }

```

Output

```
After inserting at end:
10 -> 20 -> 30 -> (back to 10)
After inserting at beginning:
5 -> 10 -> 20 -> 30 -> (back to 5)
After inserting 25 after 20:
5 -> 10 -> 20 -> 25 -> 30 -> (back to 5)


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

main.c

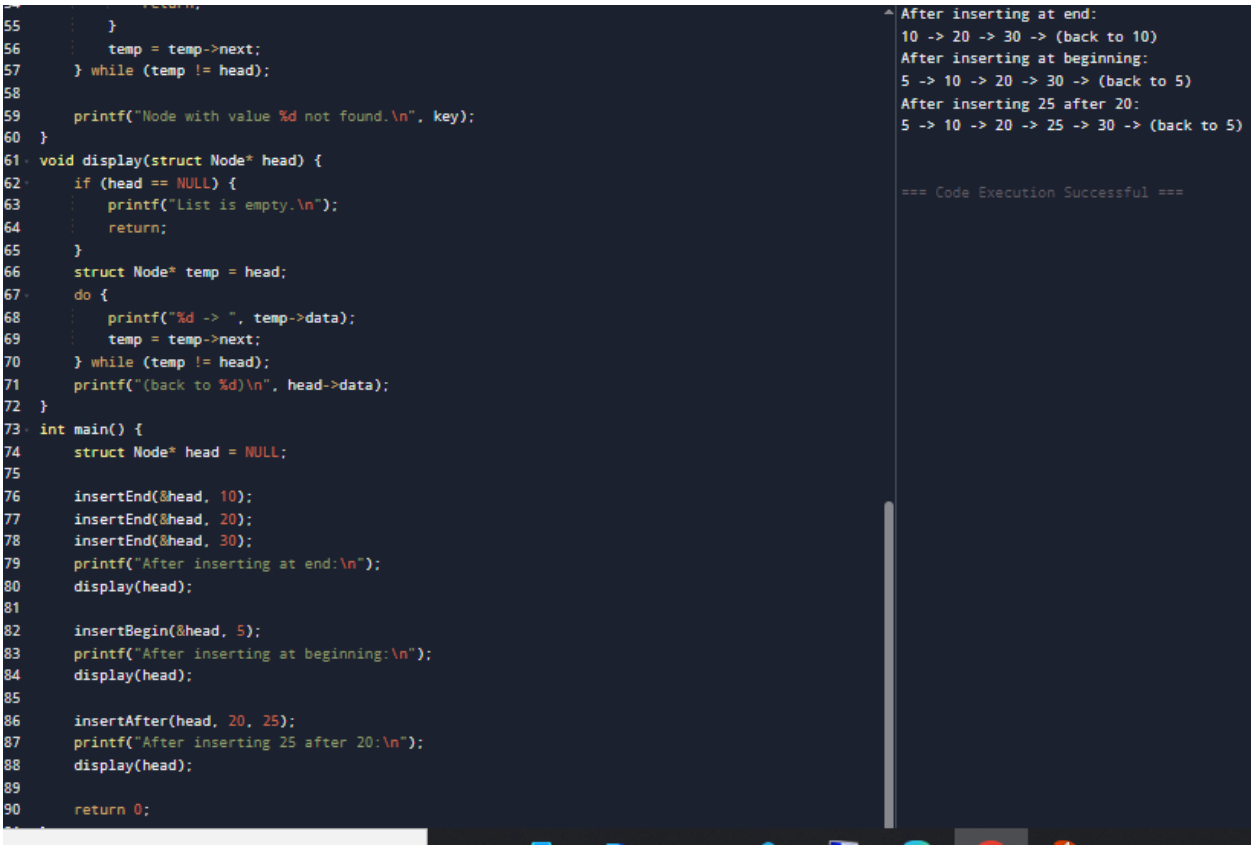
Share Run

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

    newNode->next = *head;
    temp->next = newNode;
    *head = newNode;
}

void insertAfter(struct Node* head, int key, int data) {
    struct Node* temp = head;
    do {
        if (temp->data == key) {
            struct Node* newNode = createNode(data);
            newNode->next = temp->next;
            temp->next = newNode;
            return;
        }
        temp = temp->next;
    } while (temp != head);

    printf("Node with value %d not found.\n", key);
}
void display(struct Node* head) {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }
    struct Node* temp = head;
    do {
        printf("%d -> ", temp->data);
        temp = temp->next;
    } while (temp != head);
    printf("(back to %d)\n", head->data);
}
int main() {
```

Output

After inserting at end:
10 -> 20 -> 30 -> (back to 10)
After inserting at beginning:
5 -> 10 -> 20 -> 30 -> (back to 5)
After inserting 25 after 20:
5 -> 10 -> 20 -> 25 -> 30 -> (back to 5)

=== Code Execution Successful ===



```
        }
        temp = temp->next;
    } while (temp != head);

    printf("Node with value %d not found.\n", key);
}
void display(struct Node* head) {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }
    struct Node* temp = head;
    do {
        printf("%d -> ", temp->data);
        temp = temp->next;
    } while (temp != head);
    printf("(back to %d)\n", head->data);
}
int main() {
    struct Node* head = NULL;

    insertEnd(&head, 10);
    insertEnd(&head, 20);
    insertEnd(&head, 30);
    printf("After inserting at end:\n");
    display(head);

    insertBegin(&head, 5);
    printf("After inserting at beginning:\n");
    display(head);

    insertAfter(head, 20, 25);
    printf("After inserting 25 after 20:\n");
    display(head);

    return 0;
```

```
After inserting at end:
10 -> 20 -> 30 -> (back to 10)
After inserting at beginning:
5 -> 10 -> 20 -> 30 -> (back to 5)
After inserting 25 after 20:
5 -> 10 -> 20 -> 25 -> 30 -> (back to 5)


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

16. Write a program in C to implement deletion from a circular linked list(beg; mid; end)

main.c

Share Run

```c
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};

struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}
void insertEnd(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        newNode->next = newNode;
        *head = newNode;
        return;
    }
    struct Node* temp = *head;
    while (temp->next != *head)
        temp = temp->next;
    temp->next = newNode;
    newNode->next = *head;
}
void deleteBegin(struct Node** head) {
    if (*head == NULL) return;

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

    struct Node* last = *head;
```

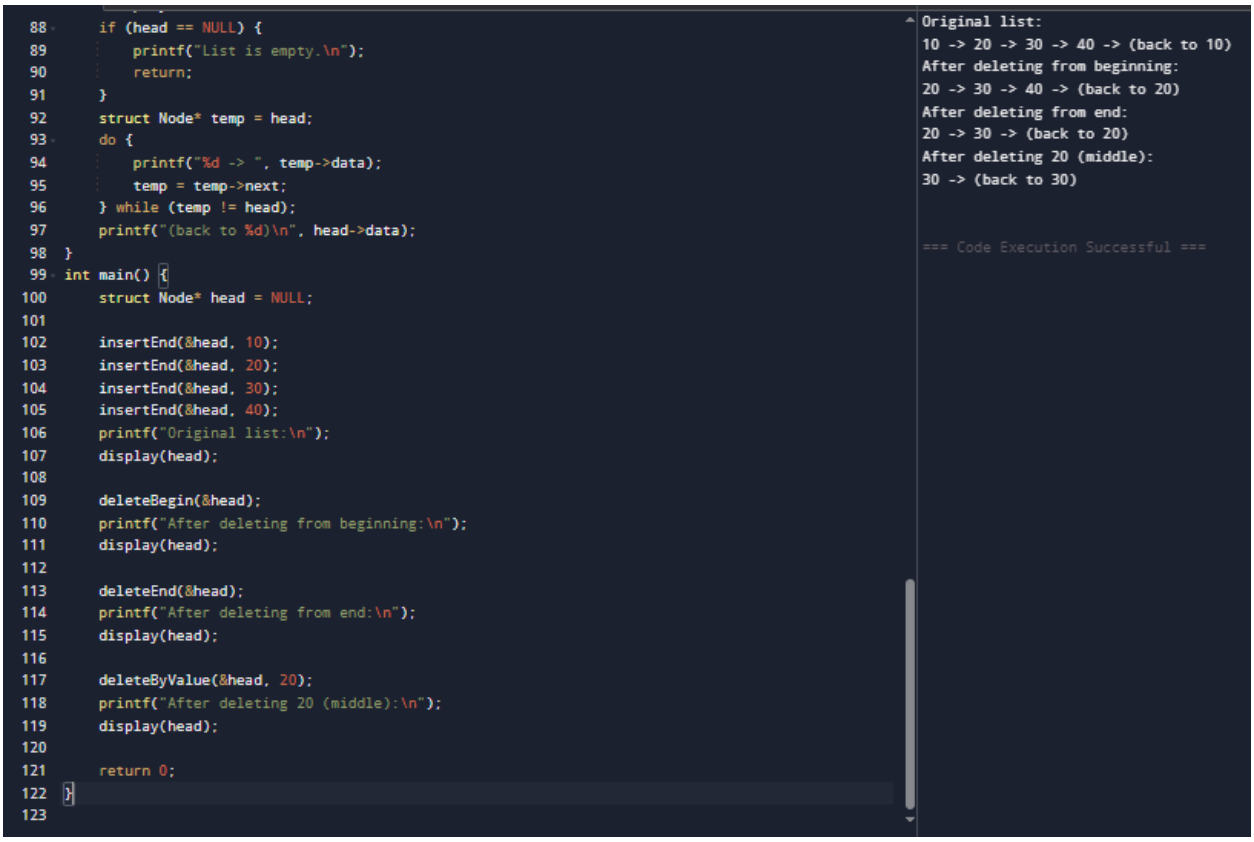
Output

```
Original list:
10 -> 20 -> 30 -> 40 -> (back to 10)
After deleting from beginning:
20 -> 30 -> 40 -> (back to 20)
After deleting from end:
20 -> 30 -> (back to 20)
After deleting 20 (middle):
30 -> (back to 30)


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

```c
    struct Node* last = *head;
    while (last->next != *head)
        last = last->next;

    *head = (*head)->next;
    last->next = *head;
    free(temp);
}
void deleteEnd(struct Node** head) {
    if (*head == NULL) return;

    struct Node* temp = *head;

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

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

    prev->next = *head;
    free(temp);
}
void deleteByValue(struct Node** head, int key) {
    if (*head == NULL) return;

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

Output

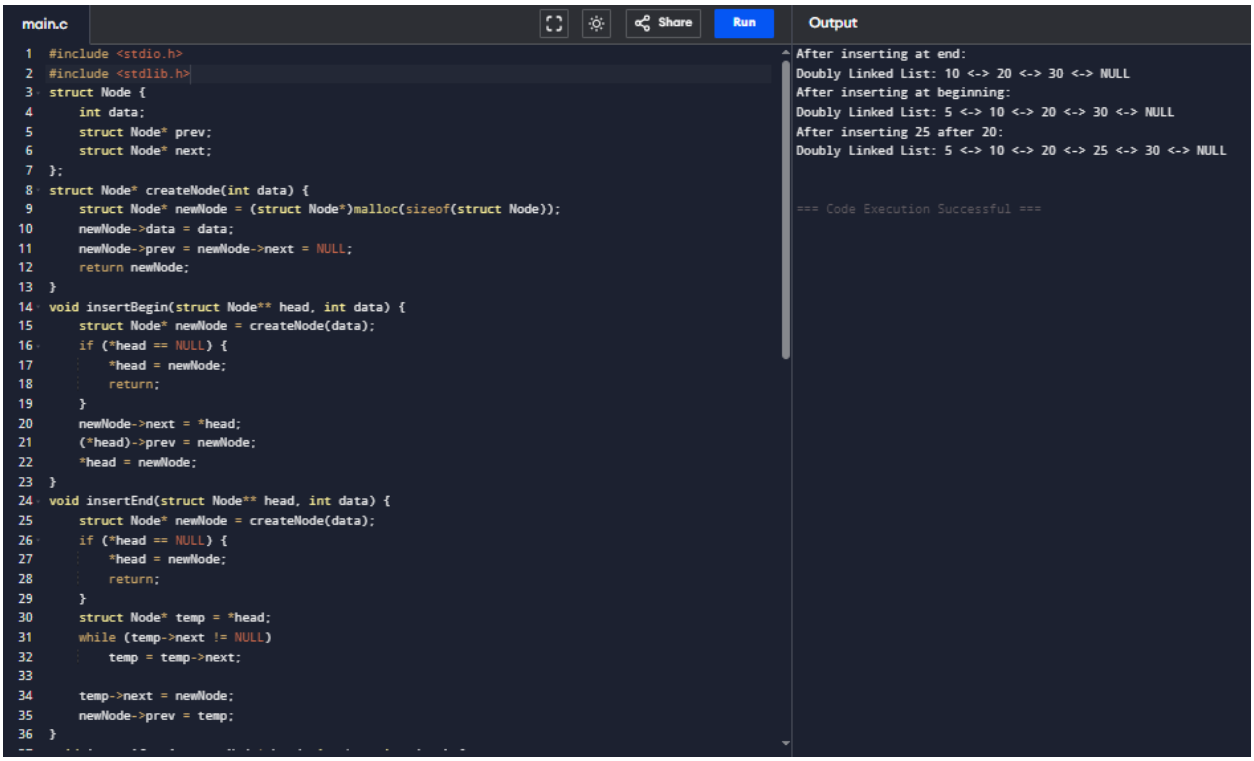
```
Original list:
10 -> 20 -> 30 -> 40 -> (back to 10)
After deleting from beginning:
20 -> 30 -> 40 -> (back to 20)
After deleting from end:
20 -> 30 -> (back to 20)
After deleting 20 (middle):
30 -> (back to 30)


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

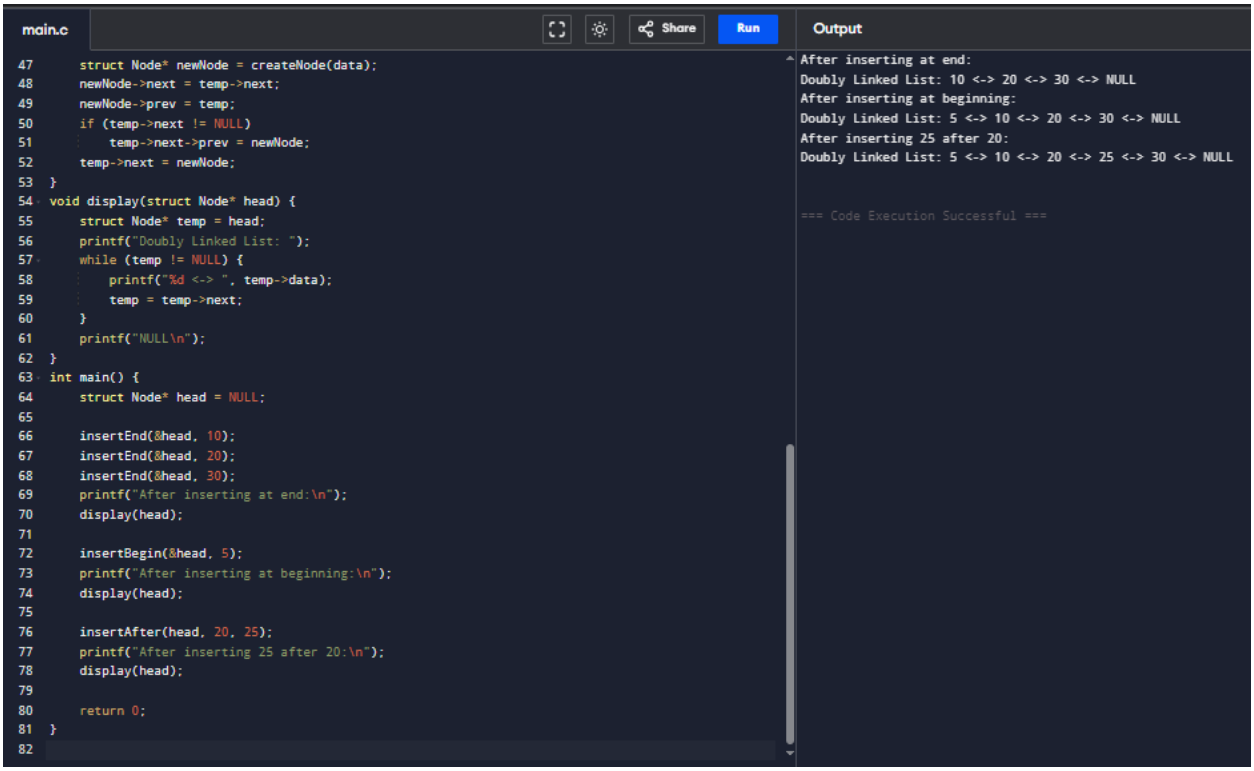


```c
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }
    struct Node* temp = head;
    do {
        printf("%d -> ", temp->data);
        temp = temp->next;
    } while (temp != head);
    printf("(back to %d)\n", head->data);
}
int main() {
    struct Node* head = NULL;

    insertEnd(&head, 10);
    insertEnd(&head, 20);
    insertEnd(&head, 30);
    insertEnd(&head, 40);
    printf("Original list:\n");
    display(head);

    deleteBegin(&head);
    printf("After deleting from beginning:\n");
    display(head);

    deleteEnd(&head);
    printf("After deleting from end:\n");
    display(head);

    deleteByValue(&head, 20);
    printf("After deleting 20 (middle):\n");
    display(head);

    return 0;
}
```

```
Original list:
10 -> 20 -> 30 -> 40 -> (back to 10)
After deleting from beginning:
20 -> 30 -> 40 -> (back to 20)
After deleting from end:
20 -> 30 -> (back to 20)
After deleting 20 (middle):
30 -> (back to 30)


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

17. Write a program in C to implement insertion in a doubly linked





18. Write a program in C to implement deletion from a doubly linked lis

main.c

```c
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* prev;
    struct Node* next;
};
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->prev = newNode->next = NULL;
    return newNode;
}
void insertEnd(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    struct Node* temp = *head;
    while (temp->next != NULL)
        temp = temp->next;
    temp->next = newNode;
    newNode->prev = temp;
}
void deleteBegin(struct Node** head) {
    if (*head == NULL) return;

    struct Node* temp = *head;
    *head = temp->next;
    if (*head != NULL)
        (*head)->prev = NULL;

    free(temp);
}
void deleteEnd(struct Node** head) {
```

Output

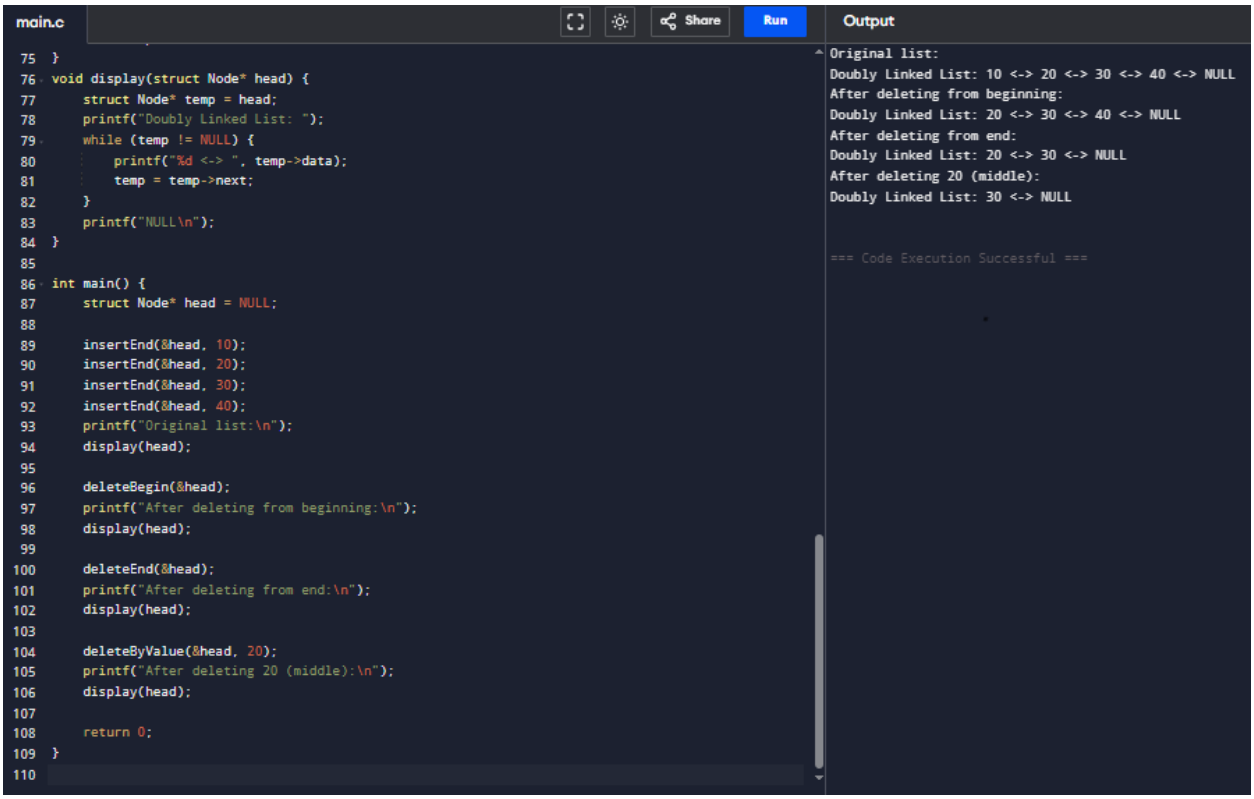
```
Original list:
Doubly Linked List: 10 <-> 20 <-> 30 <-> 40 <-> NULL
After deleting from beginning:
Doubly Linked List: 20 <-> 30 <-> 40 <-> NULL
After deleting from end:
Doubly Linked List: 20 <-> 30 <-> NULL
After deleting 20 (middle):
Doubly Linked List: 30 <-> NULL


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

main.c

```c

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

    while (temp->next != NULL)
        temp = temp->next;

    temp->prev->next = NULL;
    free(temp);
}
void deleteByValue(struct Node** head, int key) {
    if (*head == NULL) return;

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

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

    if (temp == NULL) {
        printf("Node with value %d not found.\n", key);
        return;
    }

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

```

Output

```
Original list:
Doubly Linked List: 10 <-> 20 <-> 30 <-> 40 <-> NULL
After deleting from beginning:
Doubly Linked List: 20 <-> 30 <-> 40 <-> NULL
After deleting from end:
Doubly Linked List: 20 <-> 30 <-> NULL
After deleting 20 (middle):
Doubly Linked List: 30 <-> NULL


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



19. Write a program in C to implement insertion in Binary tree

main.c

Share Run

```c
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};
struct Queue {
    int front, rear, size;
    struct Node** array;
};
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->left = newNode->right = NULL;
    return newNode;
}
struct Queue* createQueue(int size) {
    struct Queue* q = (struct Queue*)malloc(sizeof(struct Queue));
    q->front = q->rear = -1;
    q->size = size;
    q->array = (struct Node**)malloc(size * sizeof(struct Node*));
    return q;
}
int isEmpty(struct Queue* q) {
    return q->front == -1;
}

void enqueue(struct Queue* q, struct Node* node) {
    if (q->rear == q->size - 1) return;
    if (isEmpty(q)) q->front = 0;
    q->array[++q->rear] = node;
}

struct Node* dequeue(struct Queue* q) {
    if (isEmpty(q)) return NULL;
```

Output

```
Inorder traversal of Binary Tree:
40 20 50 10 30

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

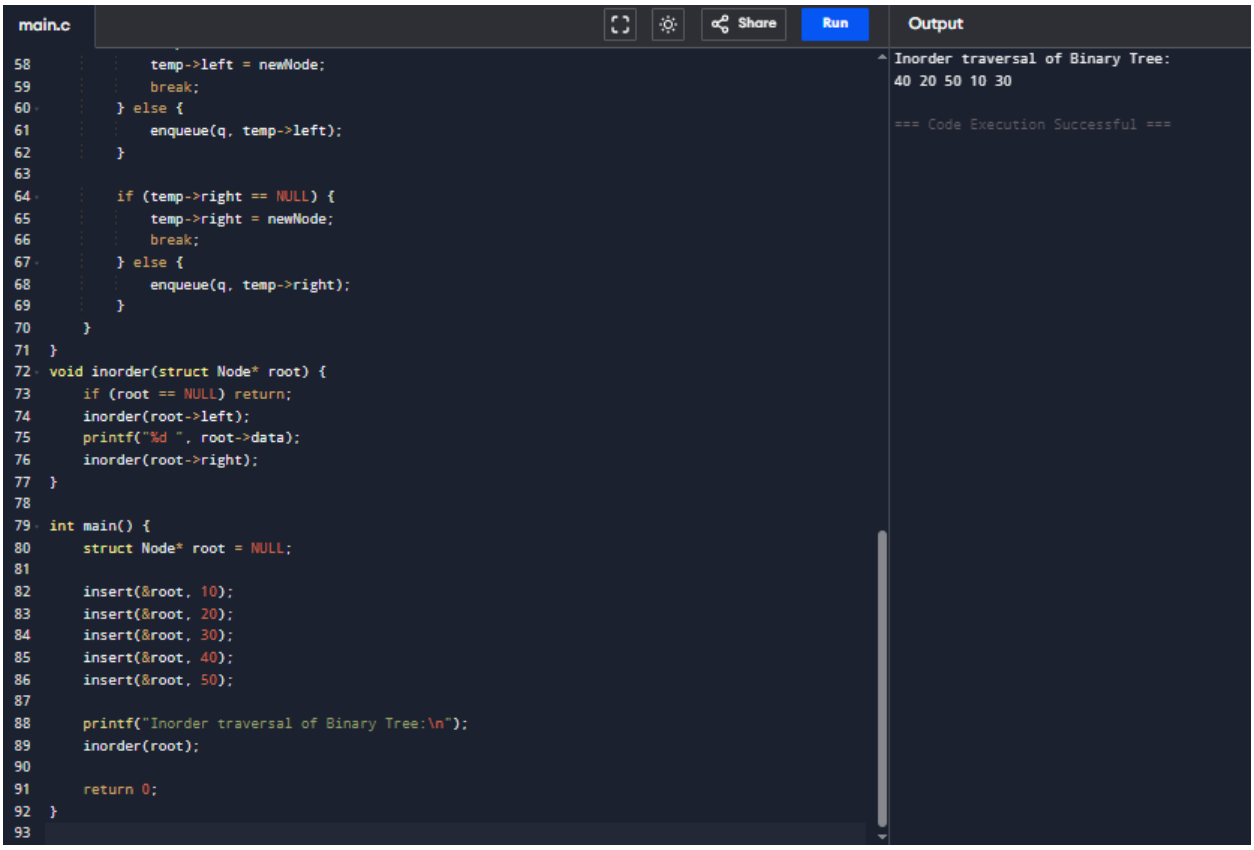
main.c

Share Run

```
    if (isEmpty(q)) return NULL;
    struct Node* temp = q->array[q->front];
    if (q->front == q->rear)
        q->front = q->rear = -1;
    else
        q->front++;
    return temp;
}
void insert(struct Node** root, int data) {
    struct Node* newNode = createNode(data);
    if (*root == NULL) {
        *root = newNode;
        return;
    }

    struct Queue* q = createQueue(100);
    enqueue(q, *root);

    while (!isEmpty(q)) {
        struct Node* temp = dequeue(q);

        if (temp->left == NULL) {
            temp->left = newNode;
            break;
        } else {
            enqueue(q, temp->left);
        }

        if (temp->right == NULL) {
            temp->right = newNode;
            break;
        } else {
            enqueue(q, temp->right);
        }
    }
}
```

Output

```
Inorder traversal of Binary Tree:
40 20 50 10 30

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



20. Write a program in C to implement deletion from Binary tree

main.c

Share Run

```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 = newNode->right = NULL;
    return newNode;
}
void inorder(struct Node* root) {
    if (root == NULL) return;
    inorder(root->left);
    printf("%d ", root->data);
    inorder(root->right);
}
void deleteDeepest(struct Node* root, struct Node* delNode) {
    if (root == NULL) return;

    if (root->left == delNode) {
        root->left = NULL;
        free(delNode);
        return;
    } else if (root->right == delNode) {
        root->right = NULL;
        free(delNode);
        return;
    }

    if (root->left)
        deleteDeepest(root->left, delNode);
    if (root->right)
        deleteDeepest(root->right, delNode);
```

Output

```
Inorder traversal before deletion:
40 20 50 10 30

Deleting node 20...
Inorder traversal after deletion:
40 50 10 30

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

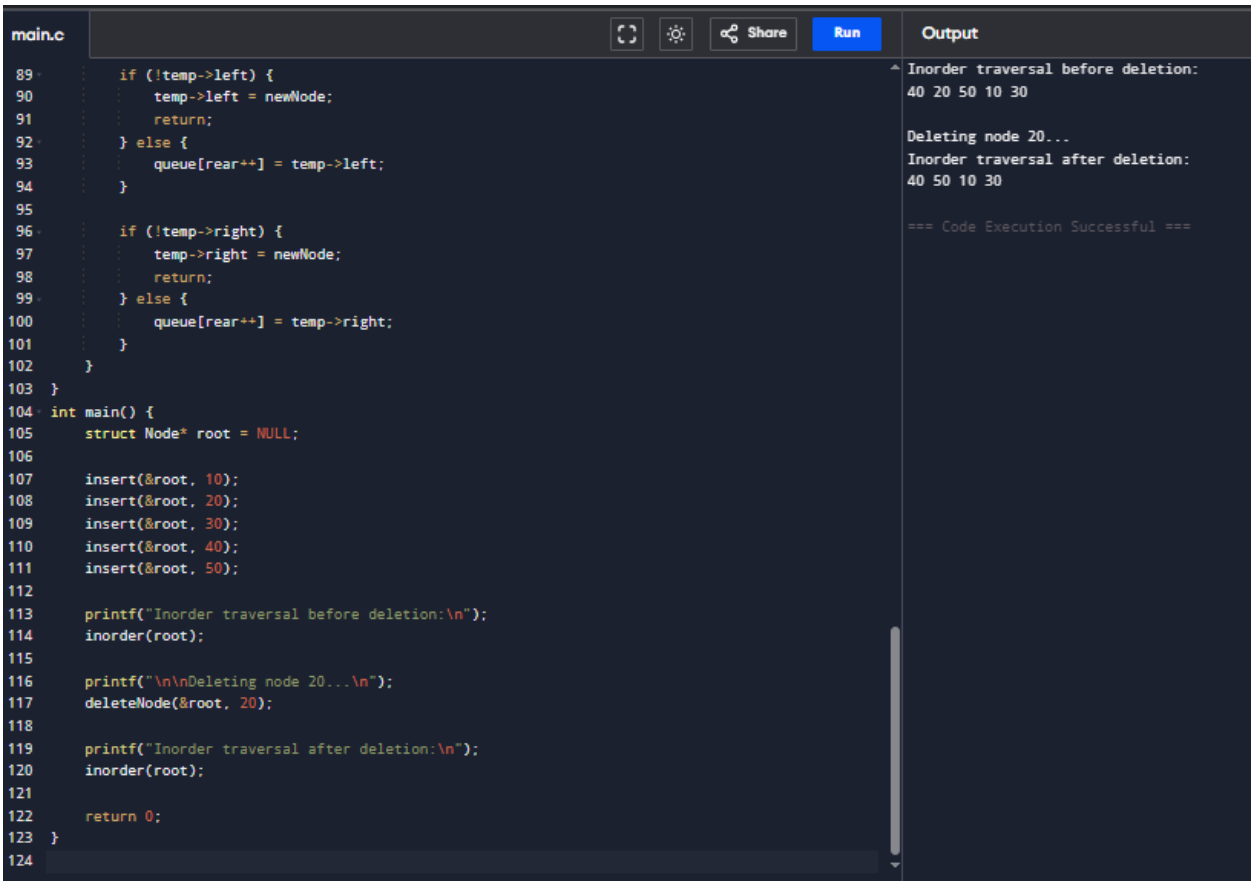
main.c

Share Run

```c
    if (root->right)
        deleteDeepest(root->right, delNode);
}
void deleteNode(struct Node** root, int key) {
    if (*root == NULL) return;

    if ((*root)->left == NULL && (*root)->right == NULL) {
        if ((*root)->data == key) {
            free(*root);
            *root = NULL;
        }
        return;
    }

    struct Node *keyNode = NULL, *temp;
    struct Node* queue[100];
    int front = 0, rear = 0;

    queue[rear++] = *root;

    while (front < rear) {
        temp = queue[front++];

        if (temp->data == key)
            keyNode = temp;

        if (temp->left)
            queue[rear++] = temp->left;
        if (temp->right)
            queue[rear++] = temp->right;
    }

    if (keyNode) {
        int x = temp->data;
        deleteDeepest(*root, temp);
        keyNode->data = x;
```

Output

```
Inorder traversal before deletion:
40 20 50 10 30

Deleting node 20...
Inorder traversal after deletion:
40 50 10 30

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



21. Write a program in C to implement recursive tree traversals \

main.c

Share Run

```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 = newNode->right = NULL;
    return newNode;
}
void preorder(struct Node* root) {
    if (root == NULL) return;
    printf("%d ", root->data);
    preorder(root->left);
    preorder(root->right);
}

void inorder(struct Node* root) {
    if (root == NULL) return;
    inorder(root->left);
    printf("%d ", root->data);
    inorder(root->right);
}
void postorder(struct Node* root) {
    if (root == NULL) return;
    postorder(root->left);
    postorder(root->right);
    printf("%d ", root->data);
}
int main() {
    struct Node* root = createNode(1);
    root->left = createNode(2);
    root->right = createNode(3);
```

Output

```
Preorder traversal: 1 2 4 5 3
Inorder traversal: 4 2 5 1 3
Postorder traversal: 4 5 2 3 1


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

main.c

Share Run

```c
}

void inorder(struct Node* root) {
    if (root == NULL) return;
    inorder(root->left);
    printf("%d ", root->data);
    inorder(root->right);
}
void postorder(struct Node* root) {
    if (root == NULL) return;
    postorder(root->left);
    postorder(root->right);
    printf("%d ", root->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("Preorder traversal: ");
    preorder(root);
    printf("\n");

    printf("Inorder traversal: ");
    inorder(root);
    printf("\n");

    printf("Postorder traversal: ");
    postorder(root);
    printf("\n");

    return 0;
}
```

Output

```
Preorder traversal: 1 2 4 5 3
Inorder traversal: 4 2 5 1 3
Postorder traversal: 4 5 2 3 1


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