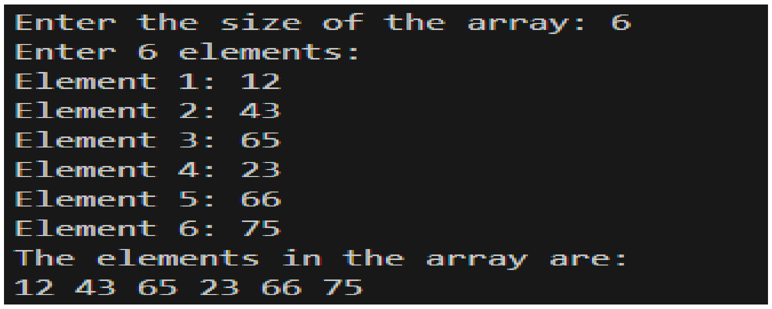
Data Structure and Algorithm using C

1. A Program to create an Array of any Specific size and traverse it.

```c
#include <stdio.h>

int main() {
    int n;

    printf("Enter the size of the array: ")
    scanf("%d", &n);

    int arr[n];

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

    // Traversal
    printf("The elements in the array are:\n");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }

    printf("\n");

    return 0;
}
```

Output :

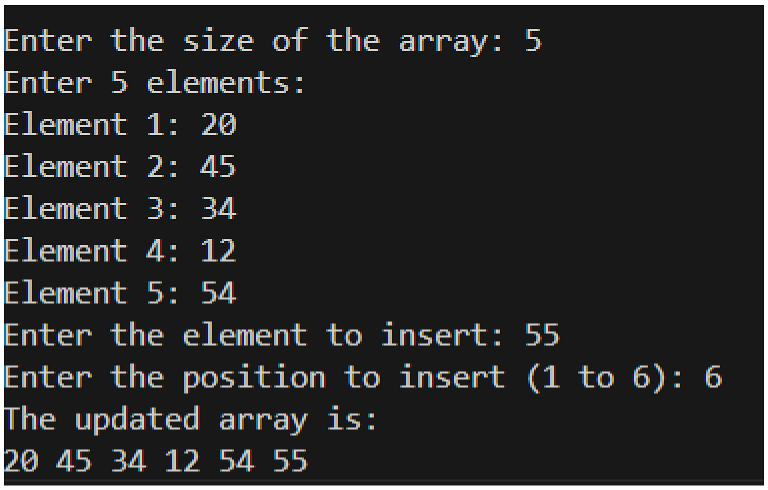


```
Enter the size of the array: 6
Enter 6 elements:
Element 1: 12
Element 2: 43
Element 3: 65
Element 4: 23
Element 5: 66
Element 6: 75
The elements in the array are:
12 43 65 23 66 75
```

2. A program to insert an specific element at any specific position in array.

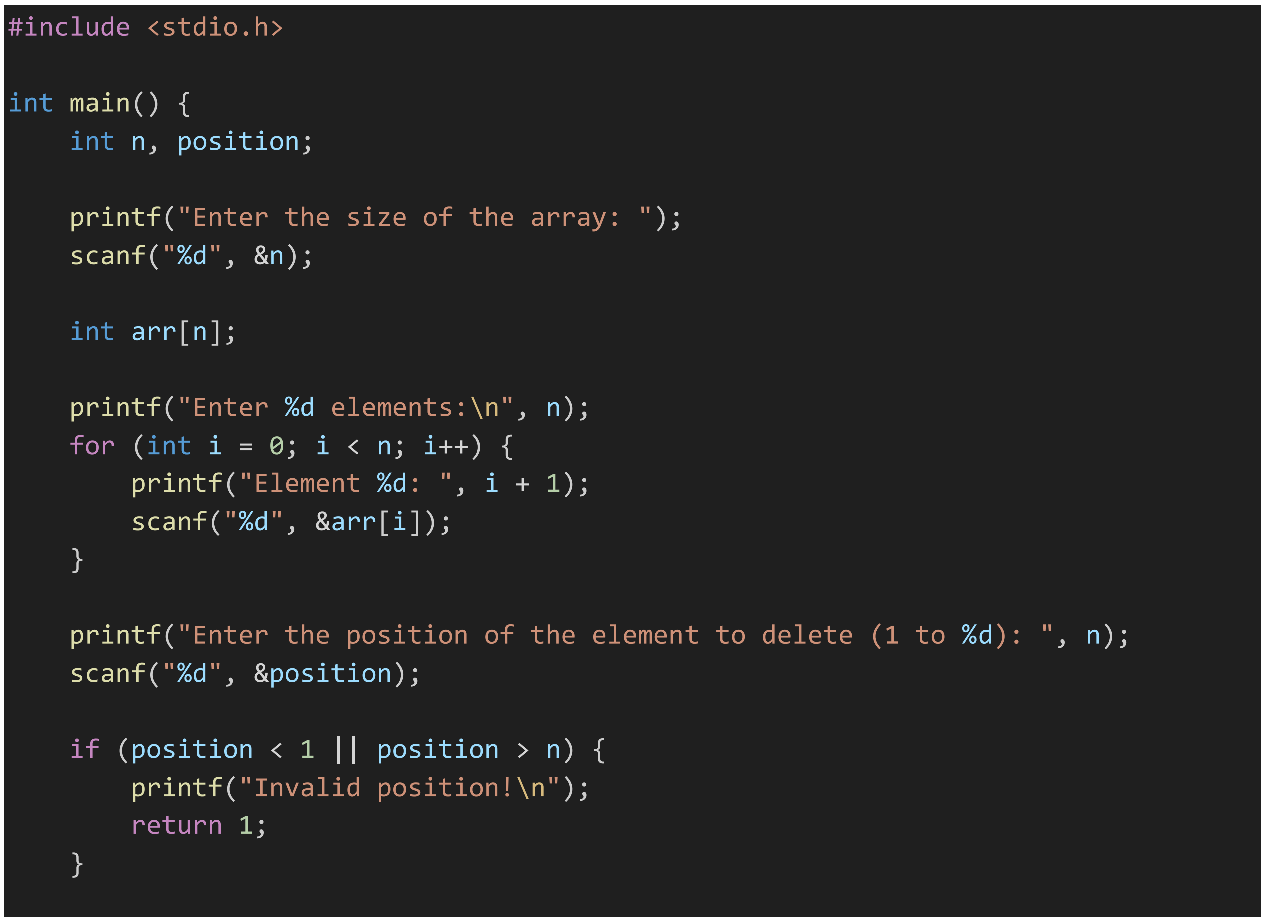
```c
#include <stdio.h>

int main() {
    int n, position, element;

    printf("Enter the size of the array: ");
    scanf("%d", &n);

    int arr[n + 1];

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

    printf("Enter the element to insert: ");
    scanf("%d", &element);
    printf("Enter the position to insert (1 to %d): ", n + 1);
    scanf("%d", &position);

    if (position < 1 || position > n + 1) {
        printf("Invalid position!\n");
        return 1;
    }

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

    arr[position - 1] = element;

    printf("The updated array is:\n");
    for (int i = 0; i <= n; i++) {
        printf("%d ", arr[i]);
    }

    printf("\n");

    return 0;
```

Output :

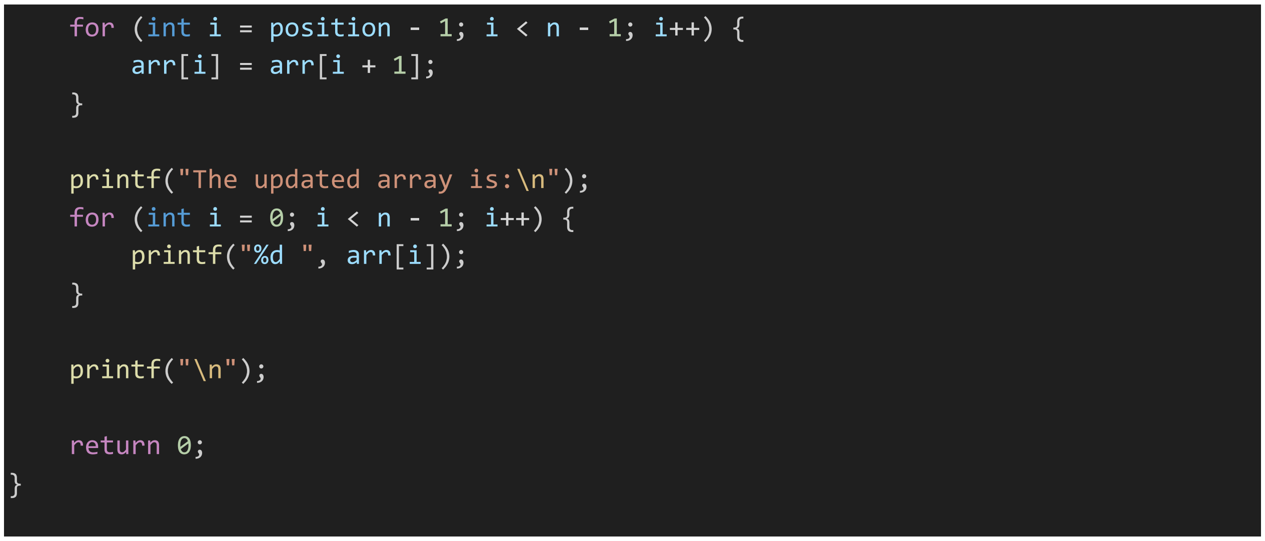


```
Enter the size of the array: 5
Enter 5 elements:
Element 1: 20
Element 2: 45
Element 3: 34
Element 4: 12
Element 5: 54
Enter the element to insert: 55
Enter the position to insert (1 to 6): 6
The updated array is:
20 45 34 12 54 55
```

3. A program to delete any specific element from array.

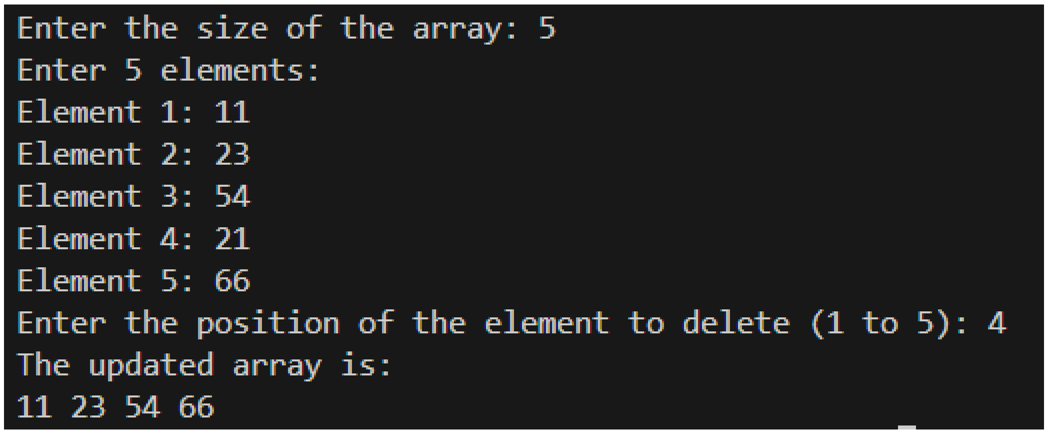


```c
#include <stdio.h>

int main() {
    int n, position;

    printf("Enter the size of the array: ");
    scanf("%d", &n);

    int arr[n];

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

    printf("Enter the position of the element to delete (1 to %d): ", n);
    scanf("%d", &position);

    if (position < 1 || position > n) {
        printf("Invalid position!\n");
        return 1;
    }
```



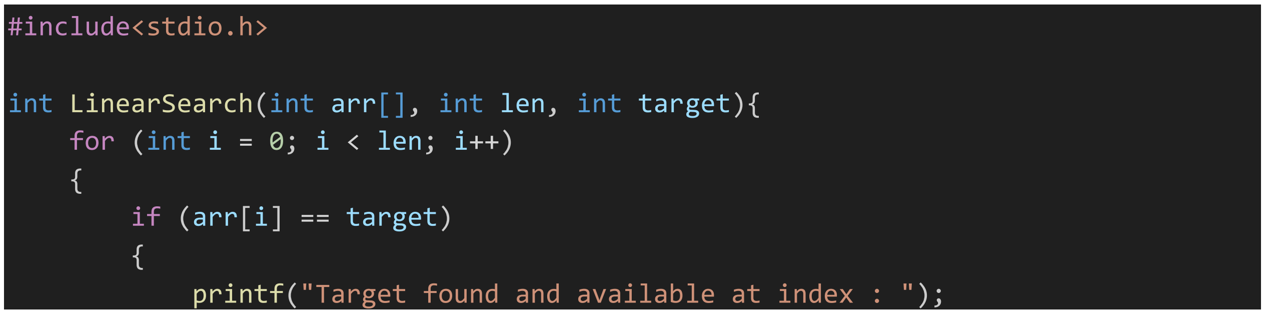
```
    for (int i = position - 1; i < n - 1; i++) {
        arr[i] = arr[i + 1];
    }

    printf("The updated array is:\n");
    for (int i = 0; i < n - 1; i++) {
        printf("%d ", arr[i]);
    }

    printf("\n");

    return 0;
}
```

Output :

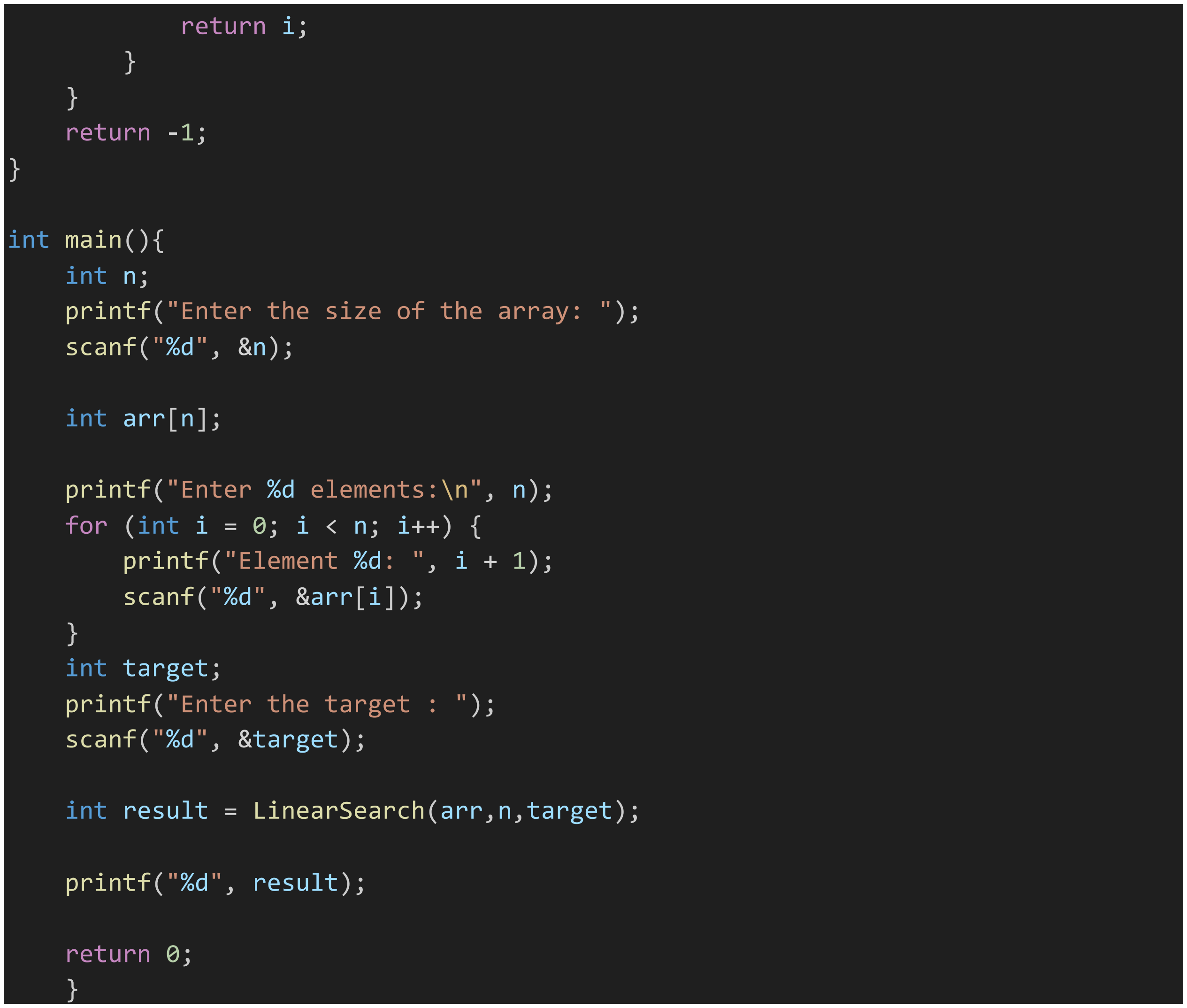


```
Enter the size of the array: 5
Enter 5 elements:
Element 1: 11
Element 2: 23
Element 3: 54
Element 4: 21
Element 5: 66
Enter the position of the element to delete (1 to 5): 4
The updated array is:
11 23 54 66
```

4. A program to search any specific element from the array using Linear Search.

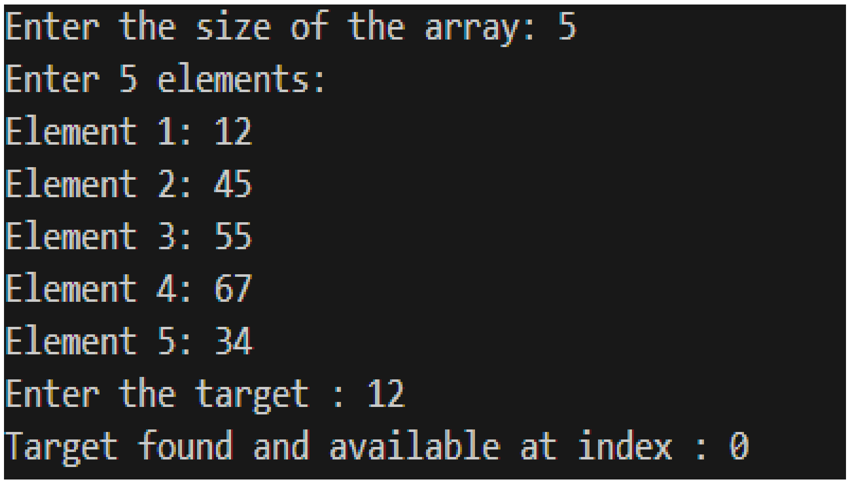


```
#include<stdio.h>

int LinearSearch(int arr[], int len, int target){
    for (int i = 0; i < len; i++)
    {
        if (arr[i] == target)
        {
            printf("Target found and available at index : ");
```



```
            return i;
        }
    }
    return -1;
}

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

    int arr[n];

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

    int result = LinearSearch(arr,n,target);

    printf("%d", result);

    return 0;
    }
```

Output :

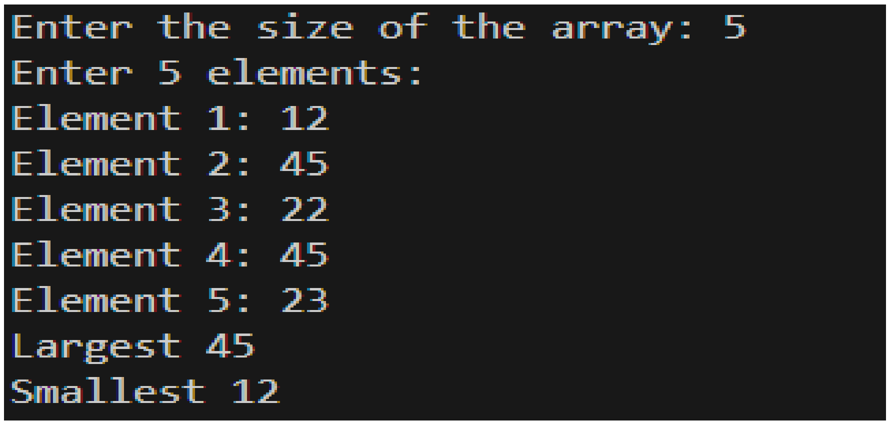


```
Enter the size of the array: 5
Enter 5 elements:
Element 1: 12
Element 2: 45
Element 3: 55
Element 4: 67
Element 5: 34
Enter the target : 12
Target found and available at index : 0
```

5. Program to find largest and smallest in given array using function.

```c
#include<stdio.h>

void Largest(int arr[], int len){
   int left = 0;
   int right = len-1;

   while (left<=right)
   {
    if (arr[left]>arr[right])
    {
       right--;
    }else{
       left++;
    }
   }
   printf("Largest %d \n", arr[right]);
}

void Smallest(int arr[], int len){
   int left = 0;
   int right = len-1;

   while (left<=right)
   {
    if (arr[left]<arr[right])
    {
       right--;
    }else{
       left++;
    }
   }
   printf("Smallest %d \n", arr[right]);
}

int main(){
    int n;

    printf("Enter the size of the array: ");
    scanf("%d", &n);

    int arr[n];

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

Largest(arr, n);
Smallest(arr, n);
return 0;
}
```

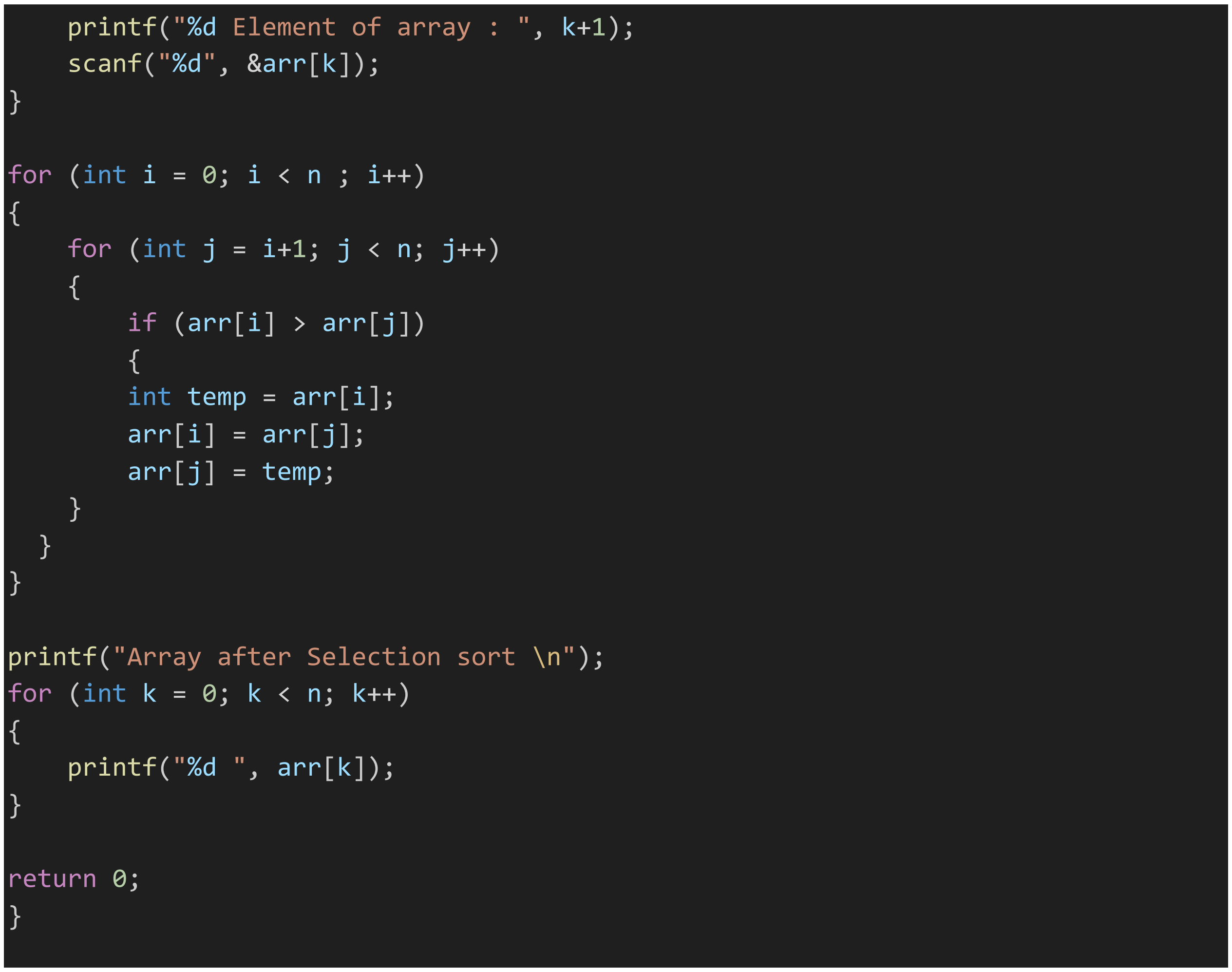
Output :



```
Enter the size of the array: 5
Enter 5 elements:
Element 1: 12
Element 2: 45
Element 3: 22
Element 4: 45
Element 5: 23
Largest 45
Smallest 12
```

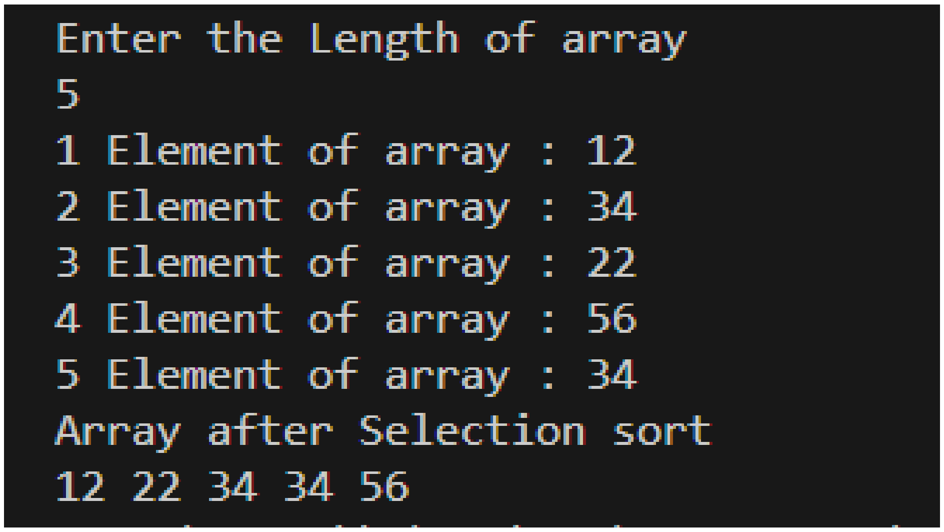
6. A program to implement Selection sort.

```
#include<stdio.h>

int main(){

int n;
printf("Enter the Length of array \n");
scanf("%d", &n);

int arr[n];

for (int k = 0; k < n; k++)
{
```



```
    printf("%d Element of array : ", k+1);
    scanf("%d", &arr[k]);
}

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

printf("Array after Selection sort \n");
for (int k = 0; k < n; k++)
{
    printf("%d ", arr[k]);
}

return 0;
}
```

Output :

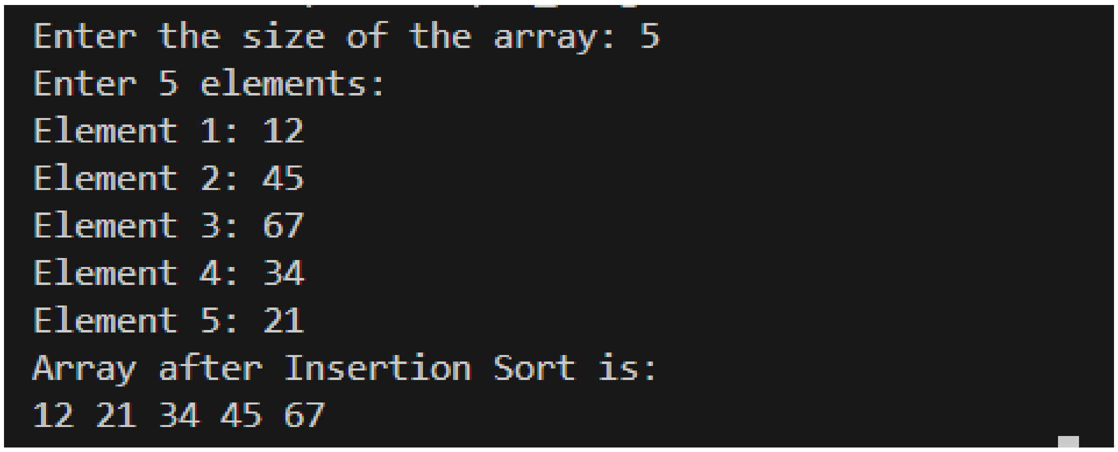


```
Enter the Length of array
5
1 Element of array : 12
2 Element of array : 34
3 Element of array : 22
4 Element of array : 56
5 Element of array : 34
Array after Selection sort
12 22 34 34 56
```

7. A program to implement Insertion sort.

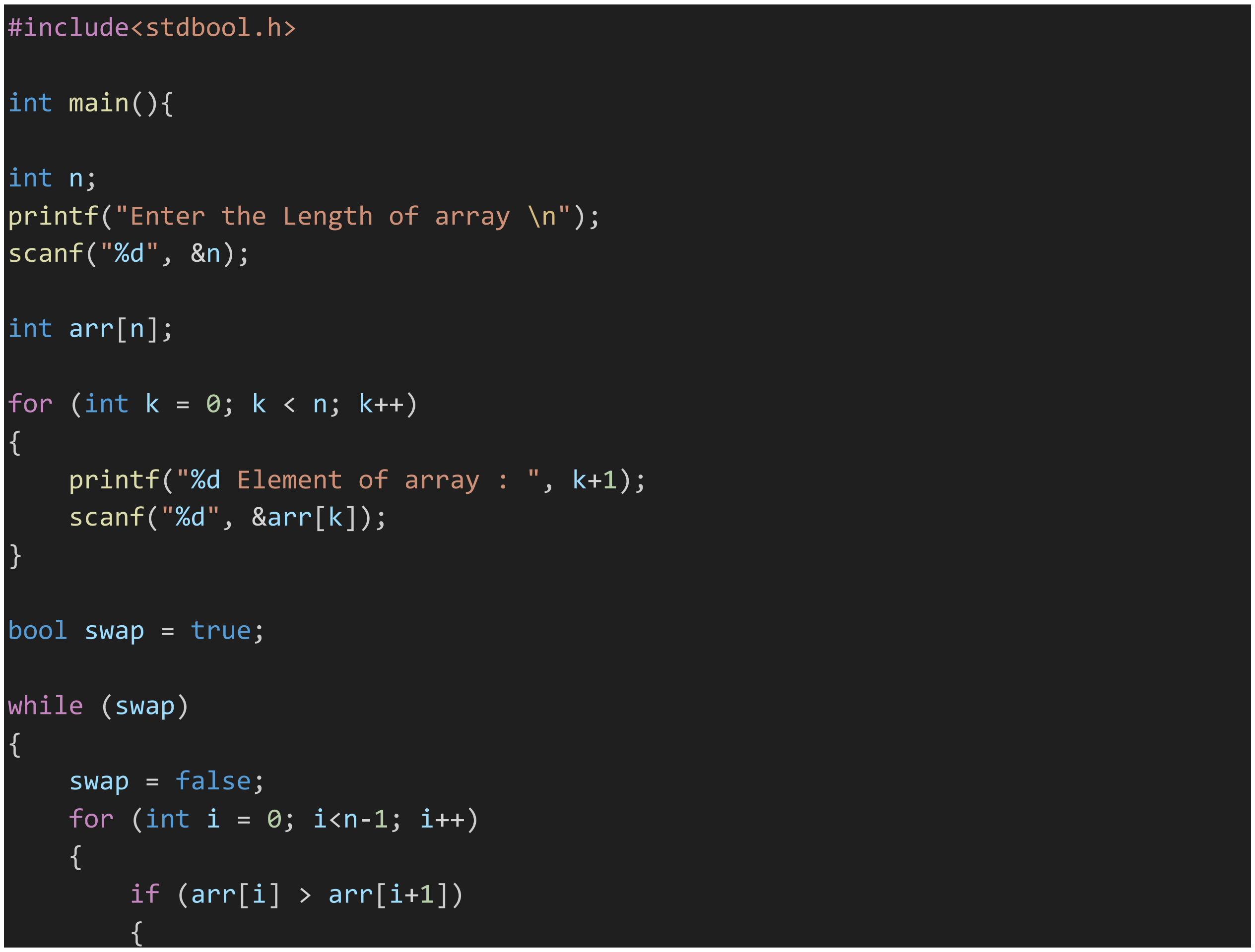
```c
#include <stdio.h>

void insertionSort(int arr[], int n) {
    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;

        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }

        arr[j + 1] = key;
    }
}

int main() {
    int n;

    printf("Enter the size of the array: ");
    scanf("%d", &n);

    int arr[n];


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

    insertionSort(arr, n);

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

    return 0;
}
```

Output :



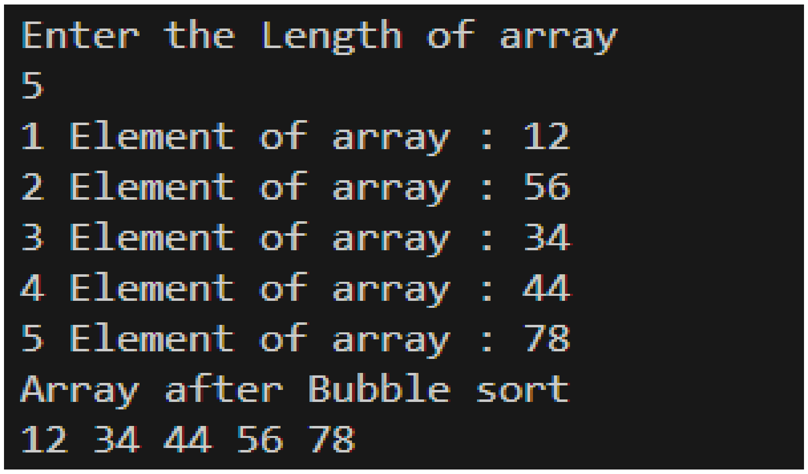
```
Enter the size of the array: 5
Enter 5 elements:
Element 1: 12
Element 2: 45
Element 3: 67
Element 4: 34
Element 5: 21
Array after Insertion Sort is:
12 21 34 45 67
```

8. A program to implement Bubble sort.



```c
#include<stdbool.h>

int main(){

int n;
printf("Enter the Length of array \n");
scanf("%d", &n);

int arr[n];

for (int k = 0; k < n; k++)
{
    printf("%d Element of array : ", k+1);
    scanf("%d", &arr[k]);
}

bool swap = true;

while (swap)
{
    swap = false;
    for (int i = 0; i<n-1; i++)
    {
        if (arr[i] > arr[i+1])
        {
           int temp = arr[i];
           arr[i] = arr[i+1];
           arr[i+1] = temp;
            swap = true;
        }
    }
    if (swap == false)
    {
        break;
    }
}

printf("Array after Bubble sort \n");
for (int k = 0; k < n; k++)
{
    printf("%d ", arr[k]);
}

}
```

Output :

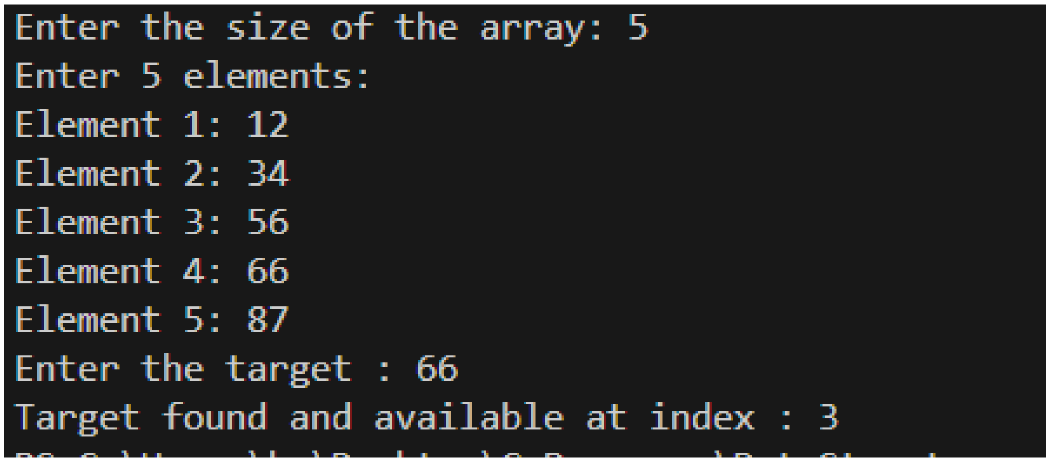


```
Enter the Length of array
5
1 Element of array : 12
2 Element of array : 56
3 Element of array : 34
4 Element of array : 44
5 Element of array : 78
Array after Bubble sort
12 34 44 56 78
```

9. A program to search any specific element from array using Binary Search.

```c
#include<stdio.h>

int BinarySearch(int arr[], int len, int target){
    int left =0;
    int right =len-1;

    while (left <= right)
    {
        int mid = left + (right - left)/2;

        if (target == arr[mid])
        {
            printf("Target found and available at index : ");
            return mid;
        }

        else if (target < arr[mid])
        {
            right = mid - 1;
        }
        else{
            left = mid + 1;
        }
    }
      return -1;

}

int main(){
    int n;

    printf("Enter the size of the array: ");
    scanf("%d", &n);

    int arr[n];

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

    int target;
    printf("Enter the target : ");
    scanf("%d", &target);
    int result = BinarySearch(arr,n,target);
    printf("%d", result);

    return 0;
}
```

Output :



```
Enter the size of the array: 5
Enter 5 elements:
Element 1: 12
Element 2: 34
Element 3: 56
Element 4: 66
Element 5: 87
Enter the target : 66
Target found and available at index : 3
```

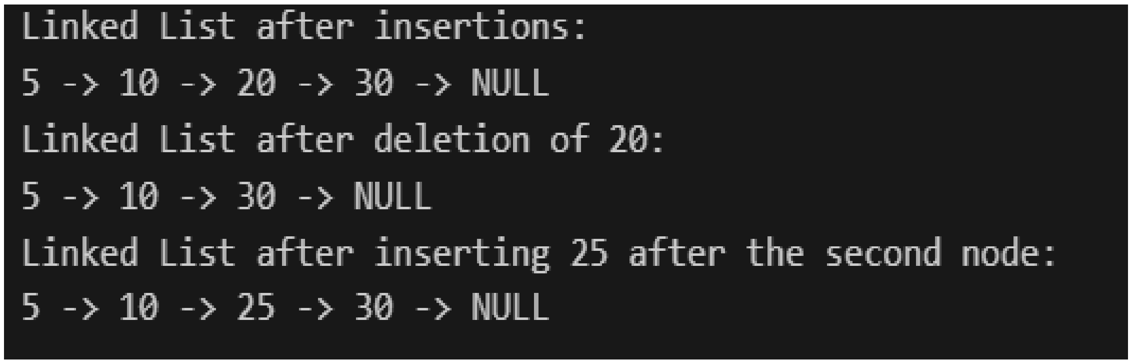
10. A program to implement single linked List.

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

// Define the structure of a node
struct Node {
    int data;
    struct Node* next;
};

// Function to create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}
```

```
// Function to insert a node at the beginning
void insertAtBeginning(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    newNode->next = *head;
    *head = newNode;
}

void insertAtEnd(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;
}

void insertAfterNode(struct Node* prevNode, int data) {
    if (prevNode == NULL) {
        printf("The given previous node cannot be NULL.\n");
        return;
    }
    struct Node* newNode = createNode(data);
    newNode->next = prevNode->next;
    prevNode->next = newNode;
}

void deleteNode(struct Node** head, int key) {
    struct Node* temp = *head;
    struct Node* prev = NULL;

    if (temp != NULL && temp->data == key) {
        *head = temp->next; // Change head
        free(temp);         // Free old head
        return;
    }

    while (temp != NULL && temp->data != key) {
        prev = temp;
        temp = temp->next;
    }
```

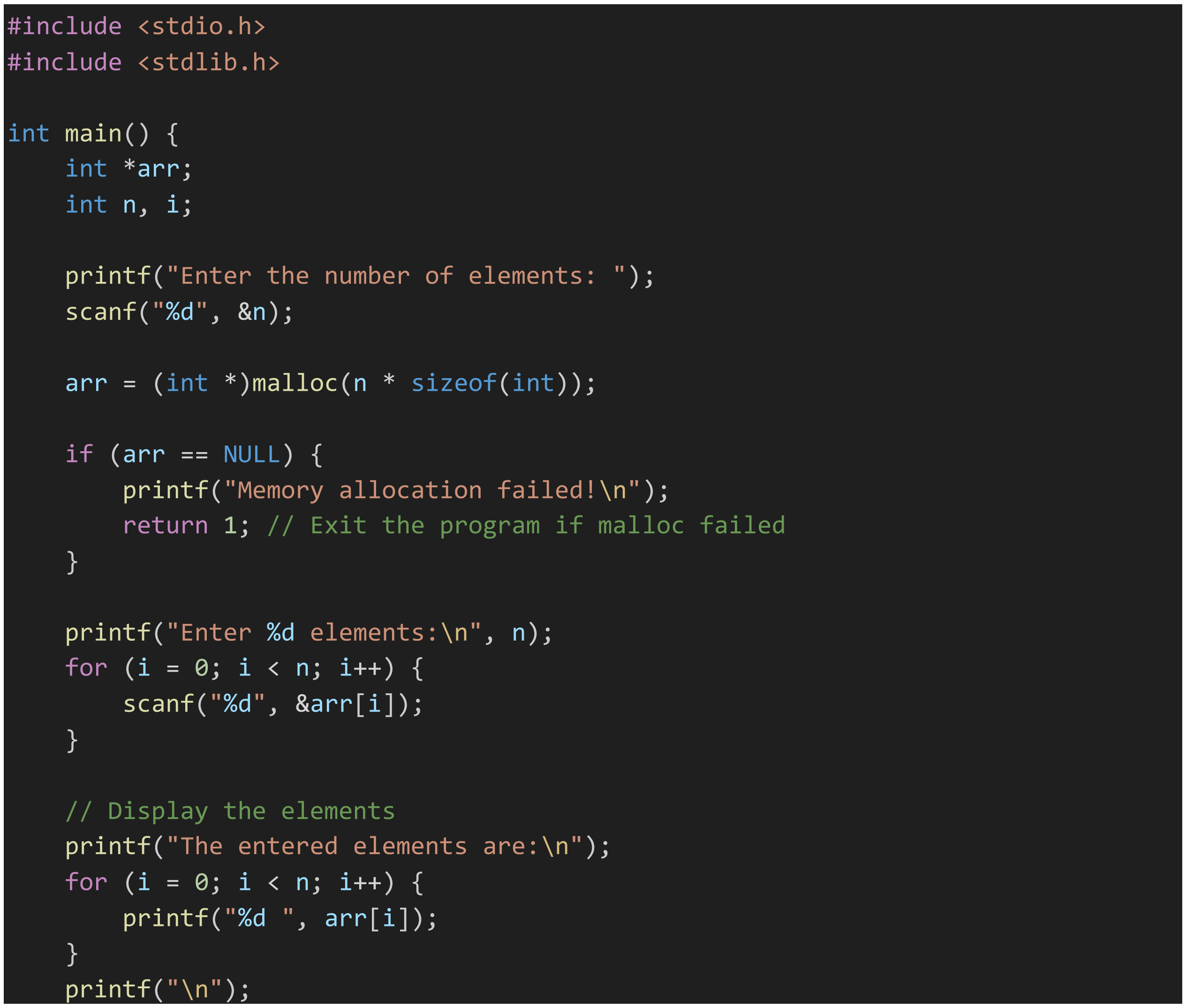
```c
    // If the key was not found
    if (temp == NULL) {
        printf("Node with value %d not found.\n", key);
        return;
    }

    prev->next = temp->next;
    free(temp); // Free memory
}

// Function to display the linked list
void displayList(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

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

    // Perform operations
    insertAtBeginning(&head, 10);
    insertAtEnd(&head, 20);
    insertAtEnd(&head, 30);
    insertAtBeginning(&head, 5);

    printf("Linked List after insertions:\n");
    displayList(head);

    deleteNode(&head, 20);
    printf("Linked List after deletion of 20:\n");
    displayList(head);

    insertAfterNode(head->next, 25);
    printf("Linked List after inserting 25 after the second node:\n");
    displayList(head);

    return 0;
}
```

Output :



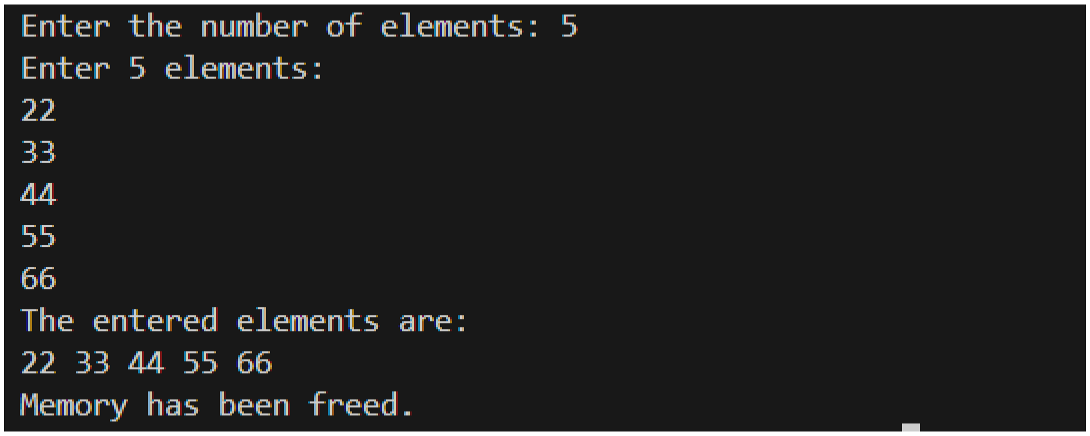
```
Linked List after insertions:
5 -> 10 -> 20 -> 30 -> NULL
Linked List after deletion of 20:
5 -> 10 -> 30 -> NULL
Linked List after inserting 25 after the second node:
5 -> 10 -> 25 -> 30 -> NULL
```

11. A program to implement MALLOC().



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

int main() {
    int *arr;
    int n, i;

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

    arr = (int *)malloc(n * sizeof(int));

    if (arr == NULL) {
        printf("Memory allocation failed!\n");
        return 1; // Exit the program if malloc failed
    }

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

    // Display the elements
    printf("The entered elements are:\n");
    for (i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    // Free the allocated memory
    free(arr);
    printf("Memory has been freed.\n");

    return 0;
}
```

Output :



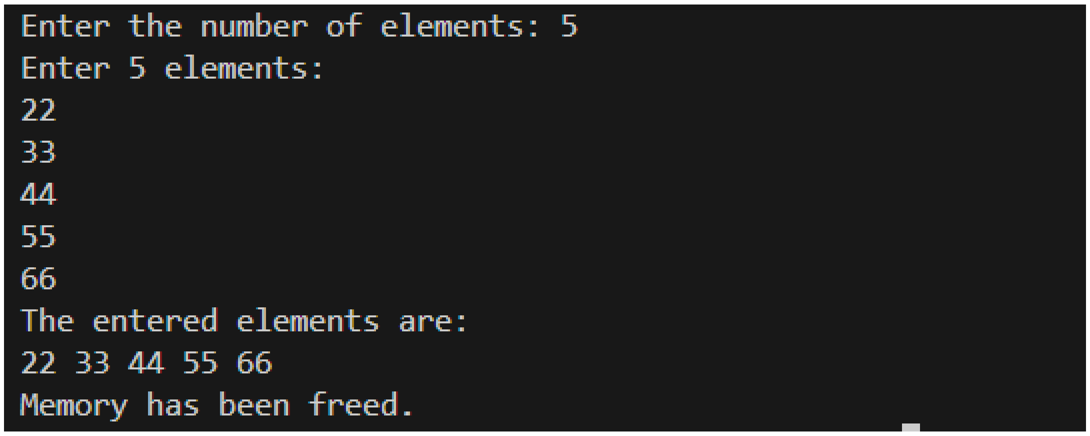
```
Enter the number of elements: 5
Enter 5 elements:
22
33
44
55
66
The entered elements are:
22 33 44 55 66
Memory has been freed.
```

12. A program to implement CALLOC().

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

int main() {
    int *arr;
    int n, i;

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

    // Allocate memory using calloc()
    arr = (int *)calloc(n, sizeof(int));

    if (arr == NULL) {
        printf("Memory allocation failed!\n");
```

```c
        return 1; // Exit the program if calloc failed
    }

    printf("Initial values in the array (set by calloc to 0):\n");
    for (i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

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

    printf("The entered elements are:\n");
    for (i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    free(arr);
    printf("Memory has been freed.\n");

    return 0;
}
```

Output :

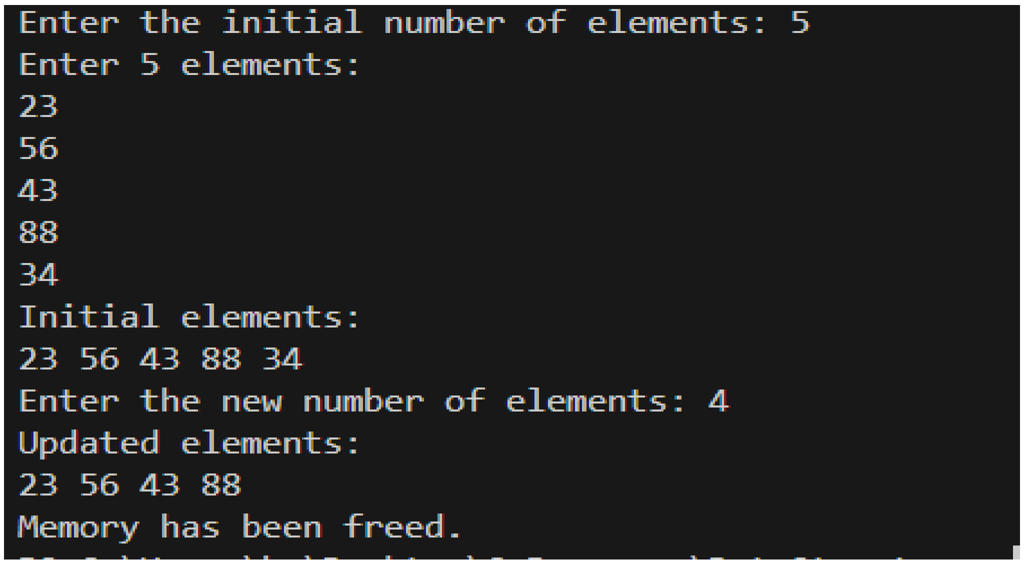


```
Enter the number of elements: 5
Enter 5 elements:
22
33
44
55
66
The entered elements are:
22 33 44 55 66
Memory has been freed.
```

13. A program to implement REALLOC() and FREE().

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

int main() {
    int *arr;
    int n, new_size, i;

    // Initial memory allocation
    printf("Enter the initial number of elements: ");
    scanf("%d", &n);

    arr = (int *)malloc(n * sizeof(int)); // Allocate memory using malloc

    // Check if memory allocation was successful
    if (arr == NULL) {
        printf("Memory allocation failed!\n");
        return 1;
    }

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

    printf("Initial elements:\n");
    for (i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    // Reallocate memory
    printf("Enter the new number of elements: ");
    scanf("%d", &new_size);

    arr = (int *)realloc(arr, new_size * sizeof(int)); // Resize the memory block

    // Check if memory reallocation was successful
    if (arr == NULL) {
        printf("Memory reallocation failed!\n");
        return 1;
    }

    // Initialize new elements (if size increased)
    if (new_size > n) {
        printf("Enter %d new elements:\n", new_size - n);
        for (i = n; i < new_size; i++) {
            scanf("%d", &arr[i]);
        }
    }

    printf("Updated elements:\n");
    for (i = 0; i < new_size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    free(arr);
    printf("Memory has been freed.\n");

    return 0;
}
```

Output :



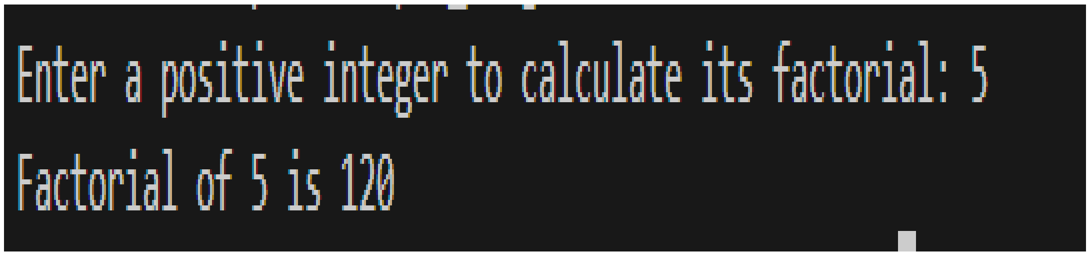
```
Enter the initial number of elements: 5
Enter 5 elements:
23
56
43
88
34
Initial elements:
23 56 43 88 34
Enter the new number of elements: 4
Updated elements:
23 56 43 88
Memory has been freed.
```

14. Factorial Using Recursion.

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

```
// Function to calculate factorial using recursion
int factorial(int n) {
    if (n == 0 || n == 1) { // Base case
        return 1;
    } else {
        return n * factorial(n - 1); // Recursive call
    }
}

int main() {
    int num;

    printf("Enter a positive integer to calculate its factorial: ");
    scanf("%d", &num);

    if (num < 0) {
        printf("Factorial is not defined for negative numbers.\n");
    } else {
        printf("Factorial of %d is %d\n", num, factorial(num));
    }

    return 0;
}
```

Output :



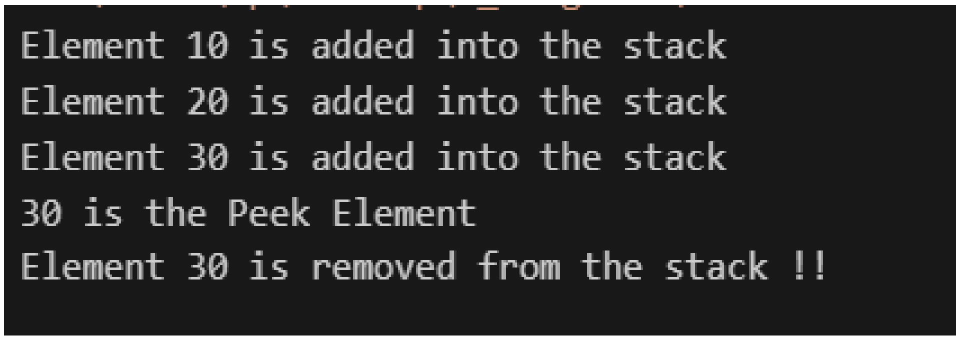
```
Enter a positive integer to calculate its factorial: 5
Factorial of 5 is 120
```

15. A program to implement stack operations using array.

```
#include <stdio.h>
#define MAX 5
```

```c
int Stack[MAX];

int top = -1;

int isFull()
{
    return top == MAX - 1;
}

int isEmpty()
{
    return top == -1;
}

void push(int value)
{
    if (isFull())
    {
        printf("Stack is Full !! \n");
    }
    else
    {
        top++;
        Stack[top] = value;
        printf("Element %d is added into the stack \n", value);
    }
}

void pop()
{
    if (isEmpty())
    {
        printf("Stack is Empty !! \n");
    }
    else
    {
        int value = Stack[top];
        top--;
        printf("Element %d is removed from the stack !! \n", value);
    }
}

int peek()
{
    return Stack[top];
}

int main()
{
    push(10);
    push(20);
    push(30);

    printf("%d is the Peek Element \n", peek());

    pop();
}
```

Output :



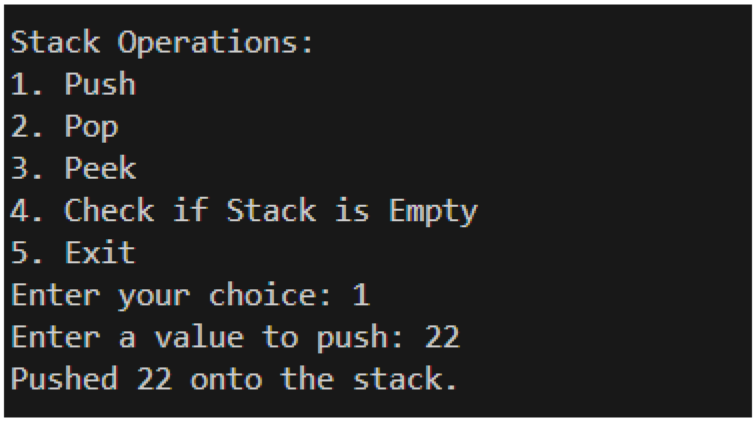
```
Element 10 is added into the stack
Element 20 is added into the stack
Element 30 is added into the stack
30 is the Peek Element
Element 30 is removed from the stack !!
```

16. A program to implement stack operations using Linked List.

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

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

void push(struct Node** top, int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Stack overflow!\n");
```

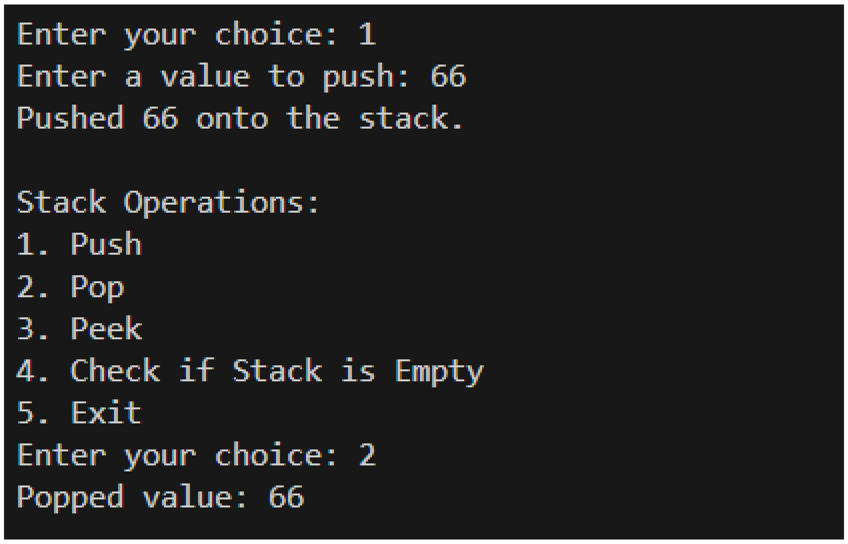
```c
        return;
    }
    newNode->data = value;
    newNode->next = *top;
    *top = newNode;
    printf("Pushed %d onto the stack.\n", value);
}

int pop(struct Node** top) {
    if (*top == NULL) {
        printf("Stack underflow!\n");
        return -1;
    }
    int poppedValue = (*top)->data;
    struct Node* temp = *top;
    *top = (*top)->next;
    free(temp);
    return poppedValue;
}

int peek(struct Node* top) {
    if (top == NULL) {
        printf("Stack is empty!\n");
        return -1;
    }
    return top->data;
}

int isEmpty(struct Node* top) {
    return top == NULL;
}

// Main function
int main() {
    struct Node* stack = NULL;
    int choice, value;

    do {
        printf("\nStack Operations:\n");
        printf("1. Push\n");
        printf("2. Pop\n");
        printf("3. Peek\n");
        printf("4. Check if Stack is Empty\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter a value to push: ");
                scanf("%d", &value);
                push(&stack, value);
                break;
            case 2:
                value = pop(&stack);
                if (value != -1) {
                    printf("Popped value: %d\n", value);
                }
                break;
            case 3:
                value = peek(stack);
                if (value != -1) {
                    printf("Top value: %d\n", value);
                }
                break;
            case 4:
                if (isEmpty(stack)) {
                    printf("Stack is empty.\n");
                } else {
                    printf("Stack is not empty.\n");
                }
                break;
            case 5:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Try again.\n");
        }
    } while (choice != 5);

    // Free any remaining nodes
    while (!isEmpty(stack)) {
        pop(&stack);
    }

    return 0;
}
```

Output for Push :



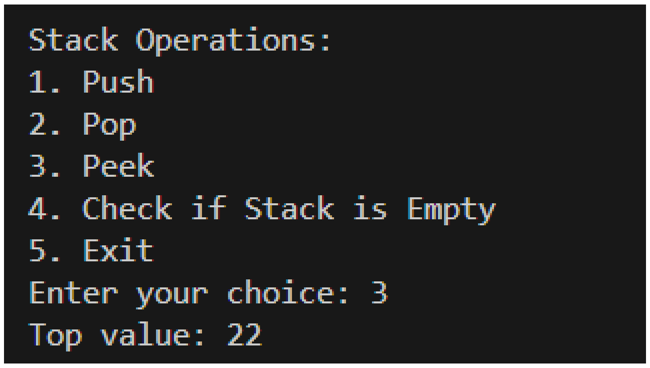
```
Stack Operations:
1. Push
2. Pop
3. Peek
4. Check if Stack is Empty
5. Exit
Enter your choice: 1
Enter a value to push: 22
Pushed 22 onto the stack.
```

Output for Pop :



```
Enter your choice: 1
Enter a value to push: 66
Pushed 66 onto the stack.

Stack Operations:
1. Push
2. Pop
3. Peek
4. Check if Stack is Empty
5. Exit
Enter your choice: 2
Popped value: 66
```

Output for peek :



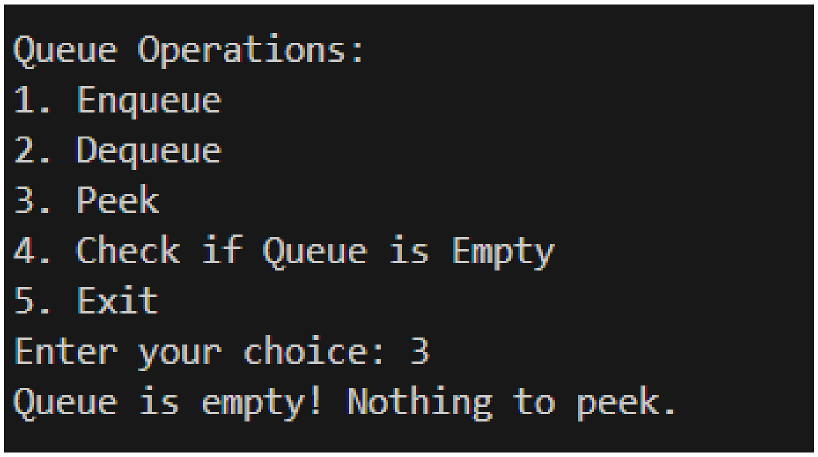
```
Stack Operations:
1. Push
2. Pop
3. Peek
4. Check if Stack is Empty
5. Exit
Enter your choice: 3
Top value: 22
```

17. A program to implement queue operations.

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

#define SIZE 5

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

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

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


int isFull(Queue* q) {
    return (q->rear + 1) % SIZE == q->front;
}

void enqueue(Queue* q, int value) {
    if (isFull(q)) {
        printf("Queue overflow! Cannot add %d.\n", value);
        return;
    }

    if (isEmpty(q)) {
        q->front = 0;
    }
    q->rear = (q->rear + 1) % SIZE;
    q->items[q->rear] = value;
    printf("Enqueued %d.\n", value);
}

int dequeue(Queue* q) {
    if (isEmpty(q)) {
        printf("Queue underflow! Nothing to dequeue.\n");
        return -1;
    }
```

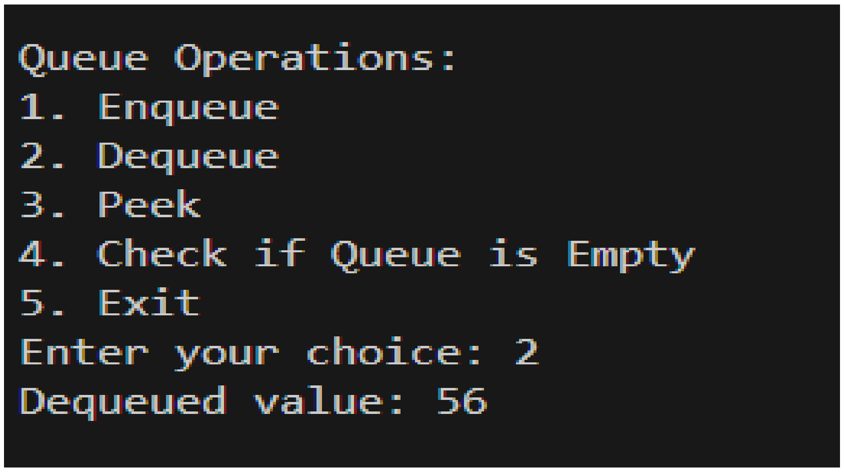
```c
    int dequeuedValue = q->items[q->front];

    if (q->front == q->rear) {

        q->front = q->rear = -1;
    } else {
        q->front = (q->front + 1) % SIZE;
    }

    return dequeuedValue;
}

int peek(Queue* q) {
    if (isEmpty(q)) {
        printf("Queue is empty! Nothing to peek.\n");
        return -1;
    }
    return q->items[q->front];
}

// Main function
int main() {
    Queue q;
    initializeQueue(&q);

    int choice, value;

    do {
        printf("\nQueue Operations:\n");
        printf("1. Enqueue\n");
        printf("2. Dequeue\n");
        printf("3. Peek\n");
        printf("4. Check if Queue is Empty\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter a value to enqueue: ");
                scanf("%d", &value);
                enqueue(&q, value);
                break;
            case 2:
                value = dequeue(&q);
                if (value != -1) {
                    printf("Dequeued value: %d\n", value);
                }
                break;
            case 3:
                value = peek(&q);
                if (value != -1) {
                    printf("Front value: %d\n", value);
                }
                break;
            case 4:
                if (isEmpty(&q)) {
                    printf("Queue is empty.\n");
                } else {
                    printf("Queue is not empty.\n");
                }
                break;
            case 5:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Try again.\n");
        }
    } while (choice != 5);

    return 0;
}
```

Output for Enqueue :



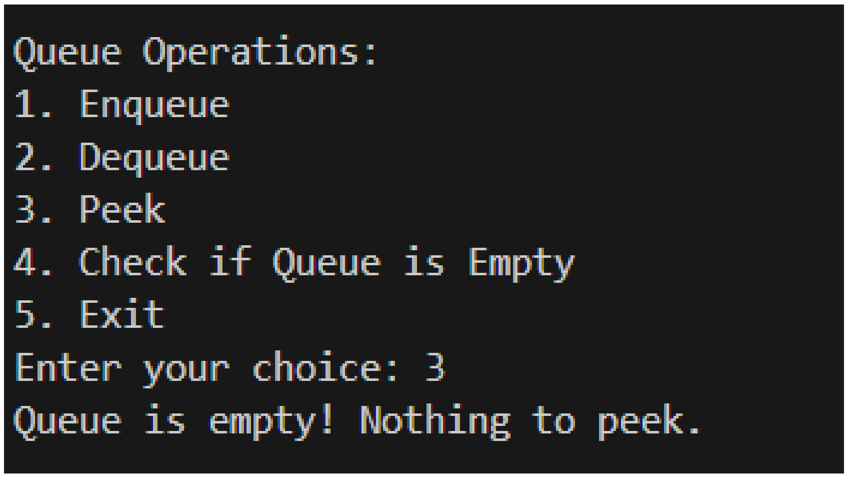
```
Queue Operations:
1. Enqueue
2. Dequeue
3. Peek
4. Check if Queue is Empty
5. Exit
Enter your choice: 3
Queue is empty! Nothing to peek.
```

Output for Dequeue :



```
Queue Operations:
1. Enqueue
2. Dequeue
3. Peek
4. Check if Queue is Empty
5. Exit
Enter your choice: 2
Dequeued value: 56
```

Output for peek :



```
Queue Operations:
1. Enqueue
2. Dequeue
3. Peek
4. Check if Queue is Empty
5. Exit
Enter your choice: 3
Queue is empty! Nothing to peek.
```

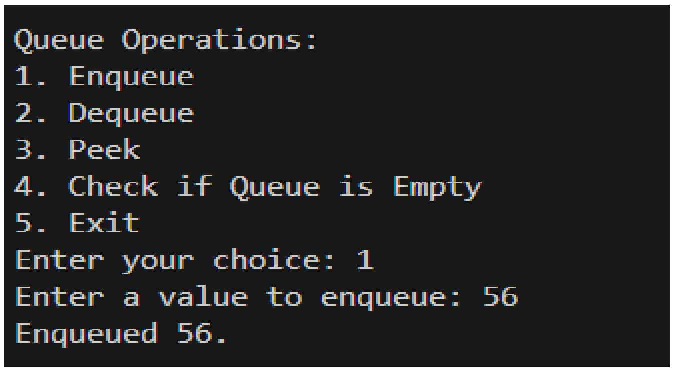
18. A program to implement queue operations using Linked List.

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

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

```c
typedef struct Queue {
    struct Node* front;
    struct Node* rear;
} Queue;

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

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

void enqueue(Queue* q, int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed! Cannot enqueue %d.\n", value);
        return;
    }
    newNode->data = value;
    newNode->next = NULL;

    if (q->rear == NULL) {
        // If the queue is empty, front and rear both point to the new node
        q->front = q->rear = newNode;
    } else {
        // Otherwise, add the new node at the rear and update the rear pointer
        q->rear->next = newNode;
        q->rear = newNode;
    }
    printf("Enqueued %d.\n", value);
}

int dequeue(Queue* q) {
    if (isEmpty(q)) {
        printf("Queue underflow! Nothing to dequeue.\n");
        return -1;
    }

    int dequeuedValue = q->front->data;
    struct Node* temp = q->front;

    q->front = q->front->next;
    // If the queue becomes empty, set rear to NULL as well
    if (q->front == NULL) {
        q->rear = NULL;
    }

    free(temp); // Free the memory of the dequeued node
    return dequeuedValue;
}

int peek(Queue* q) {
    if (isEmpty(q)) {
        printf("Queue is empty! Nothing to peek.\n");
        return -1;
    }
    return q->front->data;
}

int main() {
    Queue q;
    initializeQueue(&q);

    int choice, value;

    do {
        printf("\nQueue Operations:\n");
        printf("1. Enqueue\n");
        printf("2. Dequeue\n");
        printf("3. Peek\n");
        printf("4. Check if Queue is Empty\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter a value to enqueue: ");
                scanf("%d", &value);
                enqueue(&q, value);
                break;
            case 2:
                value = dequeue(&q);
                if (value != -1) {
                    printf("Dequeued value: %d\n", value);
                }
```

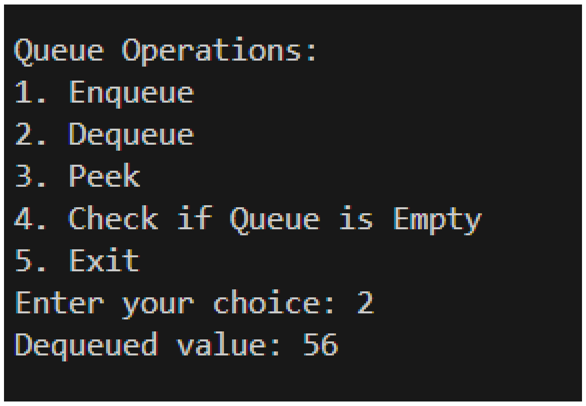
```c
                break;
            case 3:
                value = peek(&q);
                if (value != -1) {
                    printf("Front value: %d\n", value);
                }
                break;
            case 4:
                if (isEmpty(&q)) {
                    printf("Queue is empty.\n");
                } else {
                    printf("Queue is not empty.\n");
                }
                break;
            case 5:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Try again.\n");
        }
    } while (choice != 5);

    // Free remaining nodes
    while (!isEmpty(&q)) {
        dequeue(&q);
    }

    return 0;
}
```

Output for Enqueue :



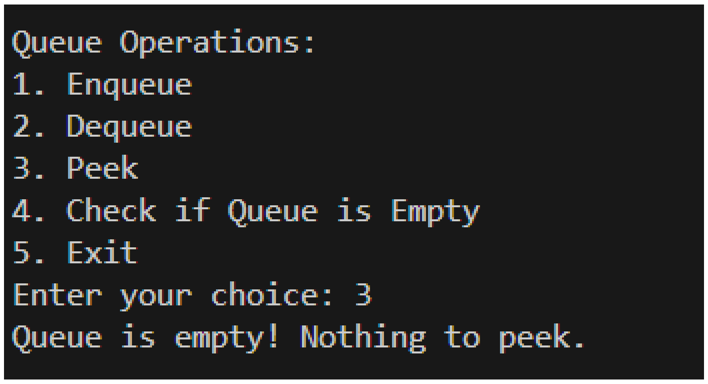
```
Queue Operations:
1. Enqueue
2. Dequeue
3. Peek
4. Check if Queue is Empty
5. Exit
Enter your choice: 1
Enter a value to enqueue: 56
Enqueued 56.
```

Output for Dequeue :



```
Queue Operations:
1. Enqueue
2. Dequeue
3. Peek
4. Check if Queue is Empty
5. Exit
Enter your choice: 2
Dequeued value: 56
```

Output for Peek :

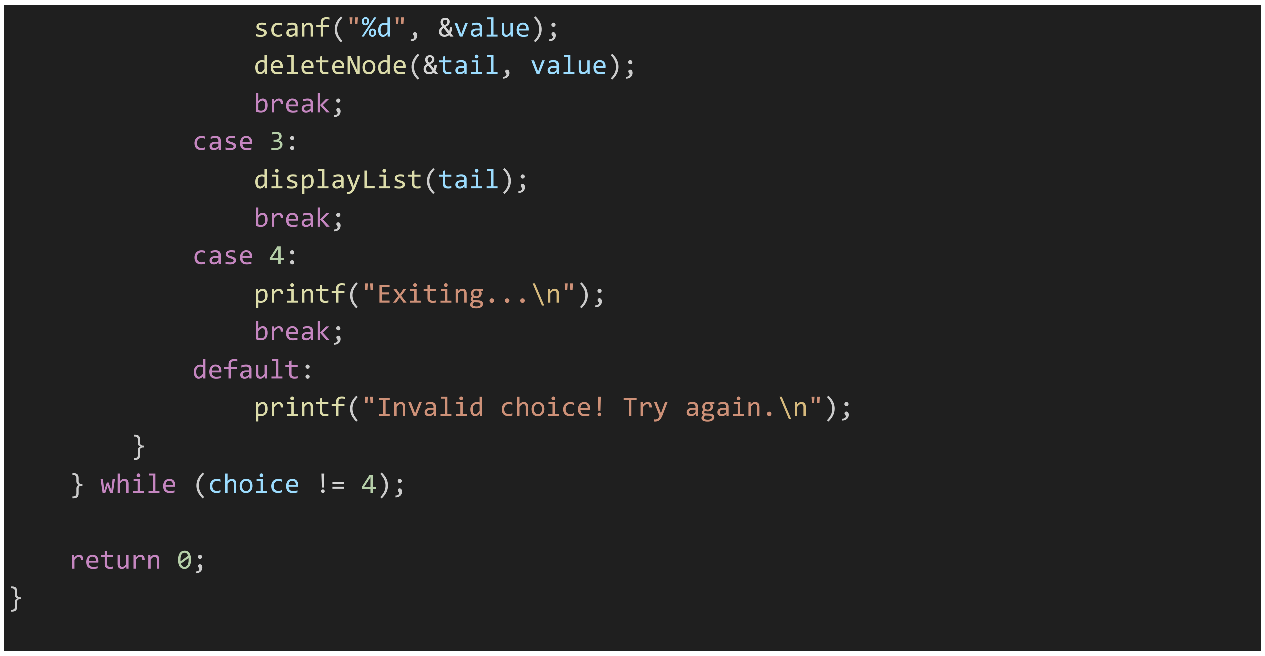


```
Queue Operations:
1. Enqueue
2. Dequeue
3. Peek
4. Check if Queue is Empty
5. Exit
Enter your choice: 3
Queue is empty! Nothing to peek.
```

19. A program to implement circular Linked list.

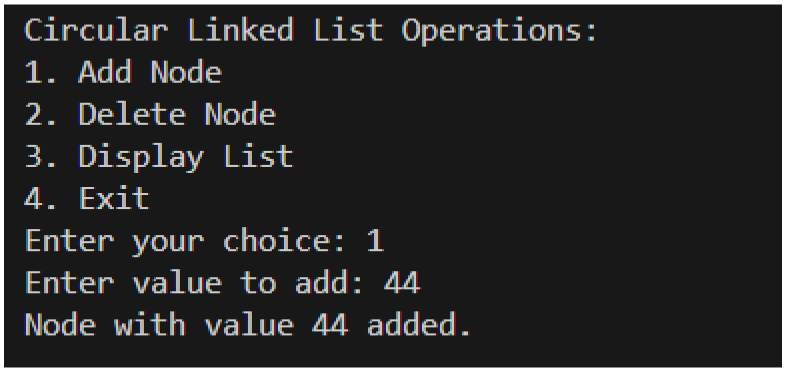
```c
#include <stdio.h>
#include <stdlib.h>

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

struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (!newNode) {
        printf("Memory allocation failed!\n");
        exit(1);
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

void addNode(struct Node** tail, int data) {
    struct Node* newNode = createNode(data);
    if (*tail == NULL) {
        *tail = newNode;
        (*tail)->next = newNode;
    } else {
        newNode->next = (*tail)->next;
        (*tail)->next = newNode;
        *tail = newNode;
    }
    printf("Node with value %d added.\n", data);
}

void deleteNode(struct Node** tail, int key) {
    if (*tail == NULL) {
        printf("List is empty! Nothing to delete.\n");
        return;
    }

    struct Node *current = (*tail)->next, *prev = *tail;

    if ((*tail)->next == *tail && (*tail)->data == key) {
        free(*tail);
        *tail = NULL;
        printf("Node with value %d deleted. List is now empty.\n", key);
        return;
    }

    do {
        if (current->data == key) {
            if (current == *tail) {
                *tail = prev;
            }
            prev->next = current->next;
            free(current);
            printf("Node with value %d deleted.\n", key);
```

```c
            return;
        }
        prev = current;
        current = current->next;
    } while (current != (*tail)->next);

    printf("Node with value %d not found!\n", key);
}

void displayList(struct Node* tail) {
    if (tail == NULL) {
        printf("List is empty!\n");
        return;
    }

    struct Node* current = tail->next;
    printf("Circular Linked List: ");
    do {
        printf("%d -> ", current->data);
        current = current->next;
    } while (current != tail->next);
    printf("(head)\n");
}

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

    do {
        printf("\nCircular Linked List Operations:\n");
        printf("1. Add Node\n");
        printf("2. Delete Node\n");
        printf("3. Display List\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to add: ");
                scanf("%d", &value);
                addNode(&tail, value);
                break;
            case 2:
                printf("Enter value to delete: ");
```



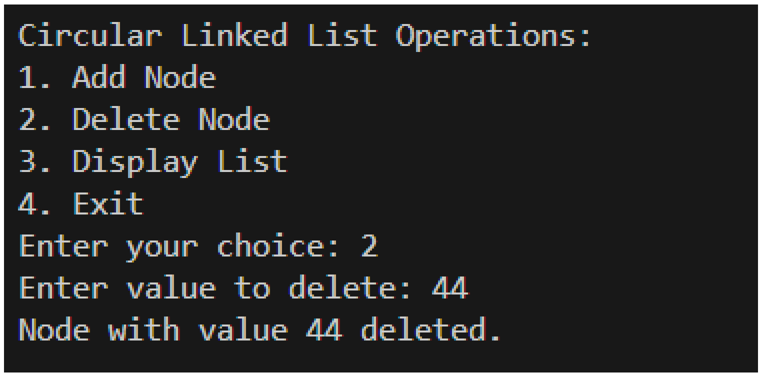
```c
                scanf("%d", &value);
                deleteNode(&tail, value);
                break;
            case 3:
                displayList(tail);
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice! Try again.\n");
        }
    } while (choice != 4);

    return 0;
}
```

Output for Add :



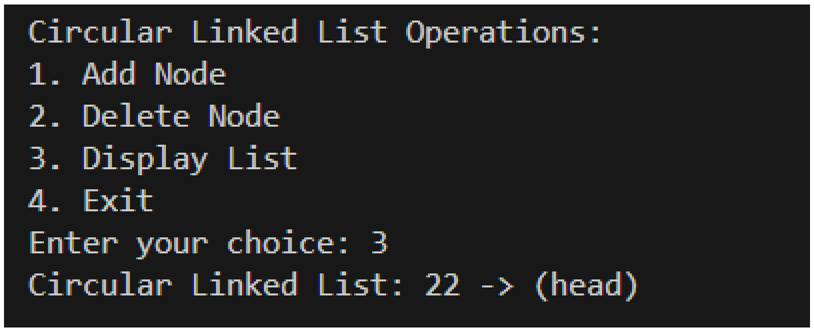
```
Circular Linked List Operations:
1. Add Node
2. Delete Node
3. Display List
4. Exit
Enter your choice: 1
Enter value to add: 44
Node with value 44 added.
```

Output for Delete :



```
Circular Linked List Operations:
1. Add Node
2. Delete Node
3. Display List
4. Exit
Enter your choice: 2
Enter value to delete: 44
Node with value 44 deleted.
```

Output for Display :

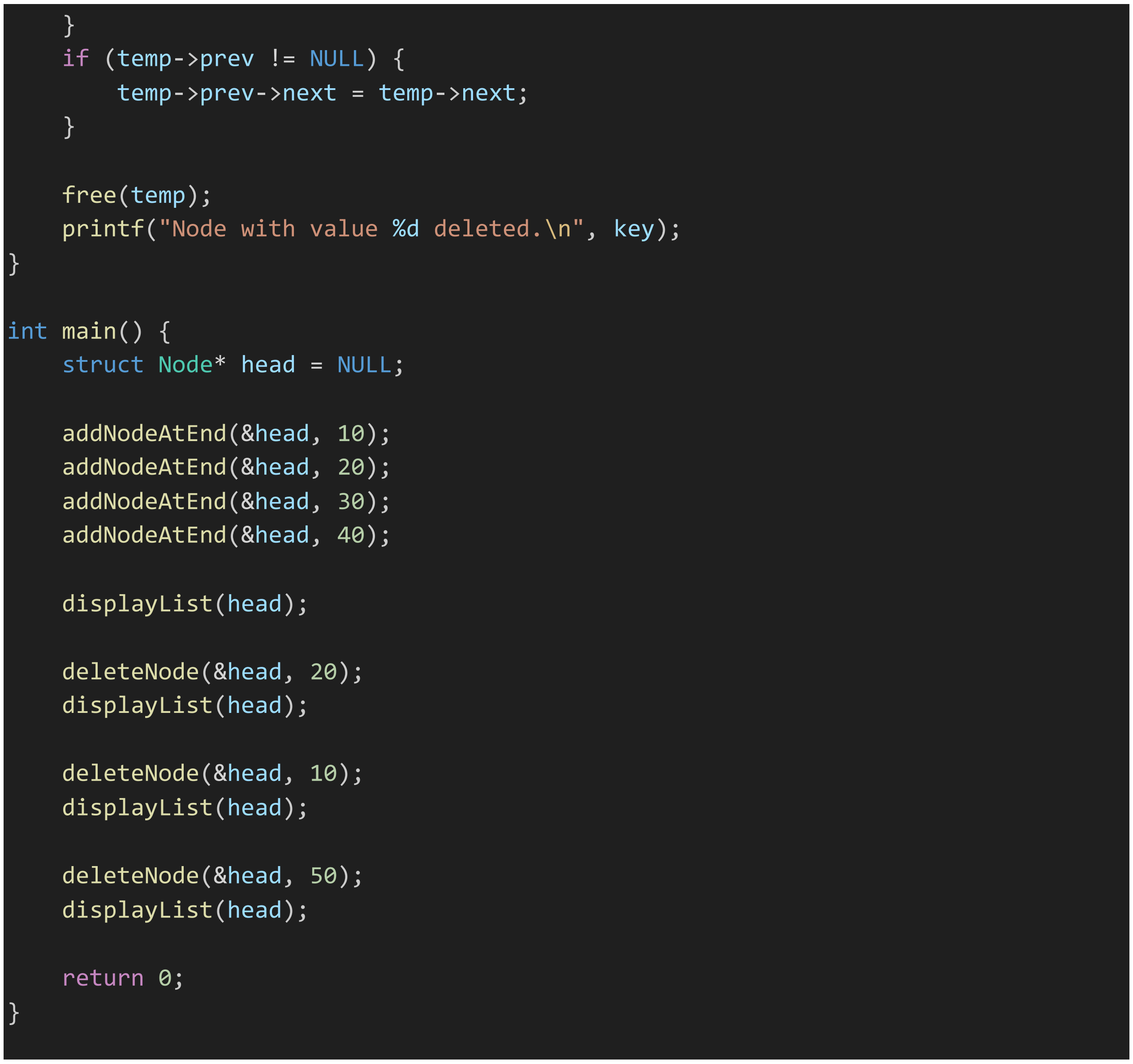


```
Circular Linked List Operations:
1. Add Node
2. Delete Node
3. Display List
4. Exit
Enter your choice: 3
Circular Linked List: 22 -> (head)
```

20. A program to implement double Linked list.

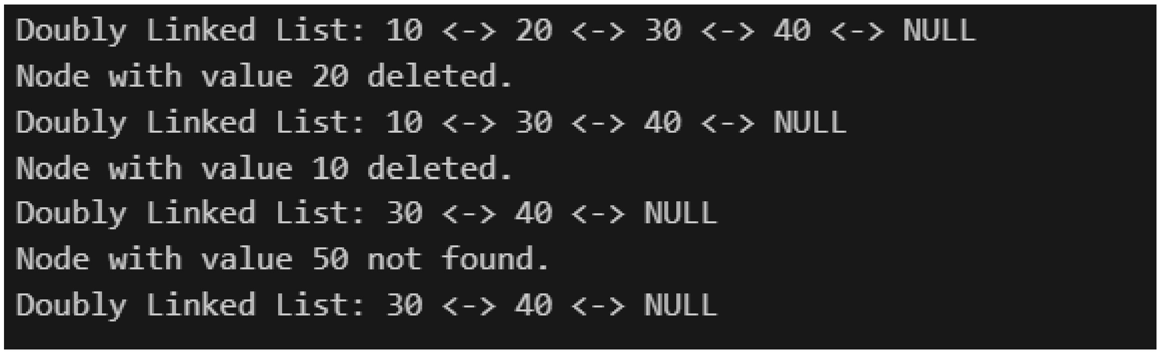
```c
#include <stdio.h>
#include <stdlib.h>

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

void addNodeAtEnd(struct Node** head, int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    struct Node* temp = *head;

    newNode->data = data;
    newNode->next = NULL;
    newNode->prev = NULL;

    if (*head == NULL) {
        *head = newNode;
        return;
    }

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

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

```c
}

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

void deleteNode(struct Node** head, int key) {
    if (*head == NULL) {
        printf("List is empty! Nothing to delete.\n");
        return;
    }

    struct Node* temp = *head;

    if ((*head)->data == key) {
        *head = (*head)->next;
        if (*head != NULL) {
            (*head)->prev = NULL;
        }
        free(temp);
        printf("Node with value %d deleted.\n", key);
        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;
```



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

    free(temp);
    printf("Node with value %d deleted.\n", key);
}

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

    addNodeAtEnd(&head, 10);
    addNodeAtEnd(&head, 20);
    addNodeAtEnd(&head, 30);
    addNodeAtEnd(&head, 40);

    displayList(head);

    deleteNode(&head, 20);
    displayList(head);

    deleteNode(&head, 10);
    displayList(head);

    deleteNode(&head, 50);
    displayList(head);

    return 0;
}
```

Output :



```
Doubly Linked List: 10 <-> 20 <-> 30 <-> 40 <-> NULL
Node with value 20 deleted.
Doubly Linked List: 10 <-> 30 <-> 40 <-> NULL
Node with value 10 deleted.
Doubly Linked List: 30 <-> 40 <-> NULL
Node with value 50 not found.
Doubly Linked List: 30 <-> 40 <-> NULL
```