Dr. B.C ROY ENGINEERING COLLEGE

LAB FILE :Data Structure & Algorithms

Paper Name: Data Structure & Algorithms

Paper Code: PCC-CS 301

Department: CSE (Data Science)

Semester: 3rd Semester

Academic Year: 2023-24

Student Name: MADHAV RAJ

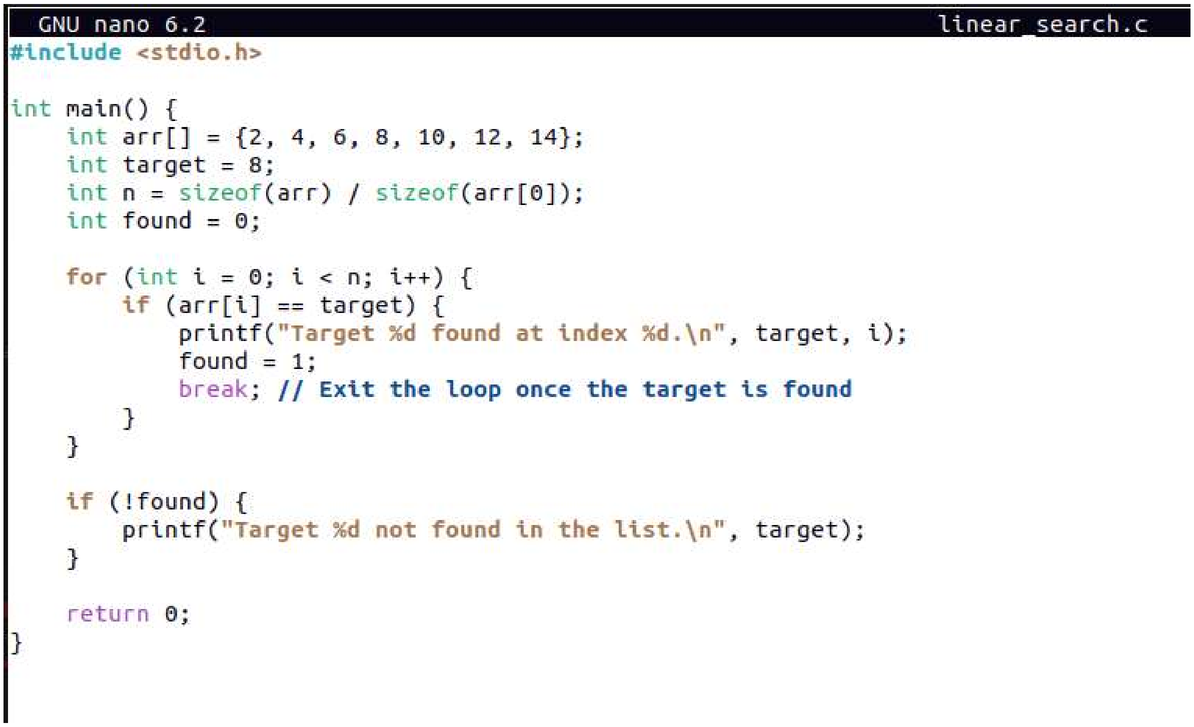
Student University Roll Number: 12030522033

Assigned Professor for the paper: Dr. Chandan Bandyopadhyay

Designation: Associate Professor and HoD, CSE(DS)

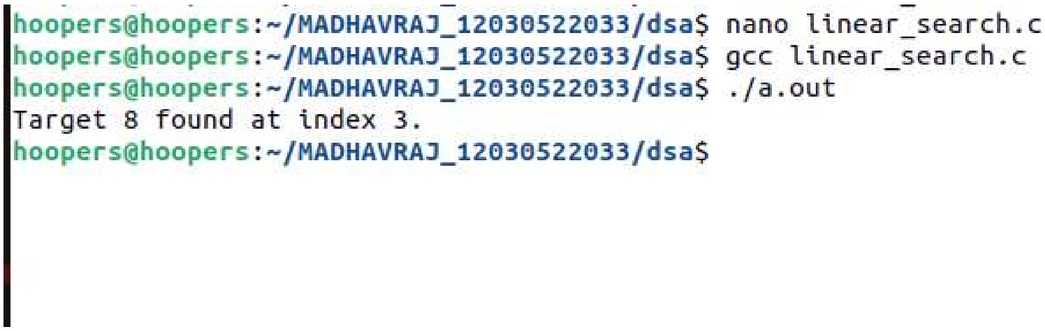
Program1: WAP to execute Linear Search:

INPUT:



```c
#include <stdio.h>

int main() {
    int arr[] = {2, 4, 6, 8, 10, 12, 14};
    int target = 8;
    int n = sizeof(arr) / sizeof(arr[0]);
    int found = 0;

    for (int i = 0; i < n; i++) {
        if (arr[i] == target) {
            printf("Target %d found at index %d.\n", target, i);
            found = 1;
            break; // Exit the loop once the target is found
        }
    }

    if (!found) {
        printf("Target %d not found in the list.\n", target);
    }

    return 0;
}
```

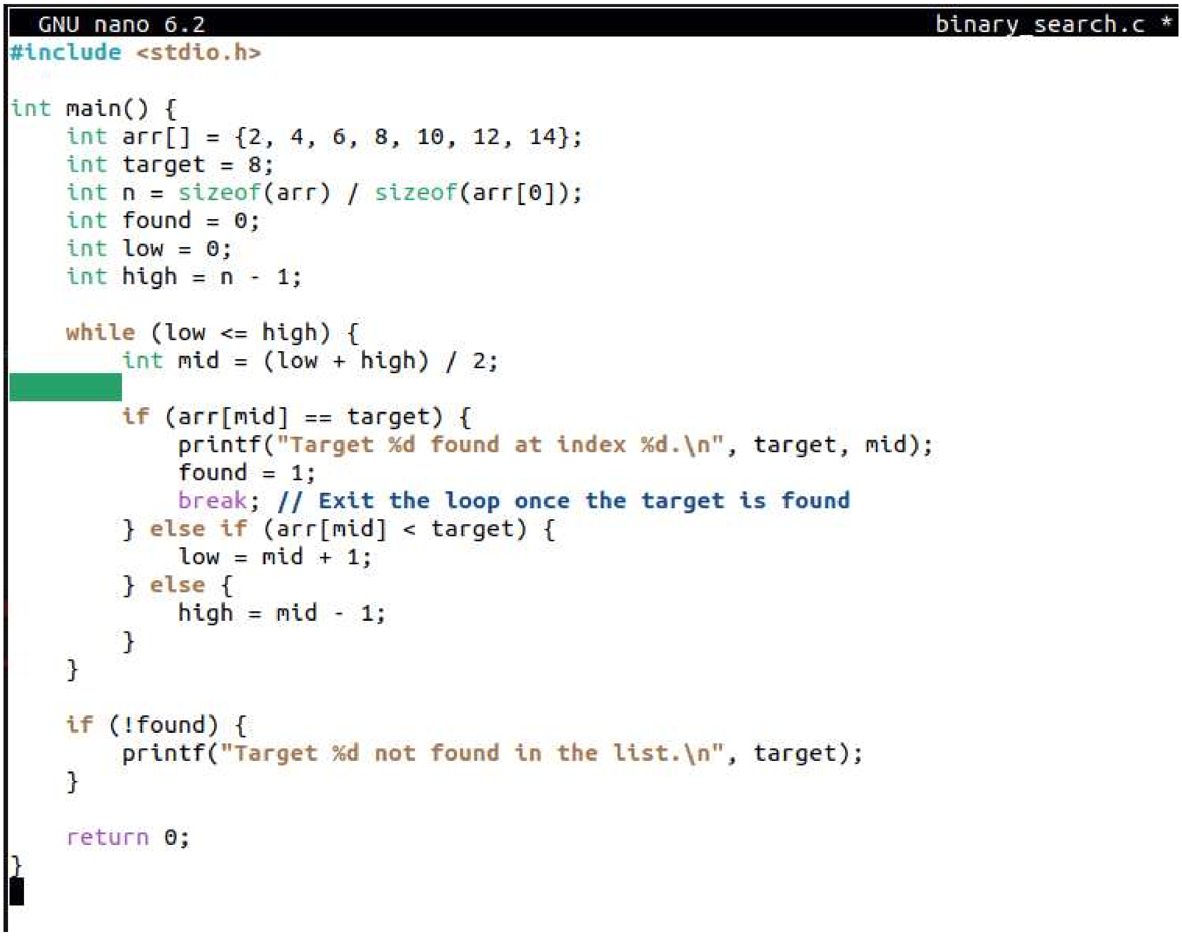
OUTPUT:



```
hoopers@hoopers:~/MADHAVRAJ_12030522033/dsa$ nano linear_search.c
hoopers@hoopers:~/MADHAVRAJ_12030522033/dsa$ gcc linear_search.c
hoopers@hoopers:~/MADHAVRAJ_12030522033/dsa$ ./a.out
Target 8 found at index 3.
hoopers@hoopers:~/MADHAVRAJ_12030522033/dsa$
```

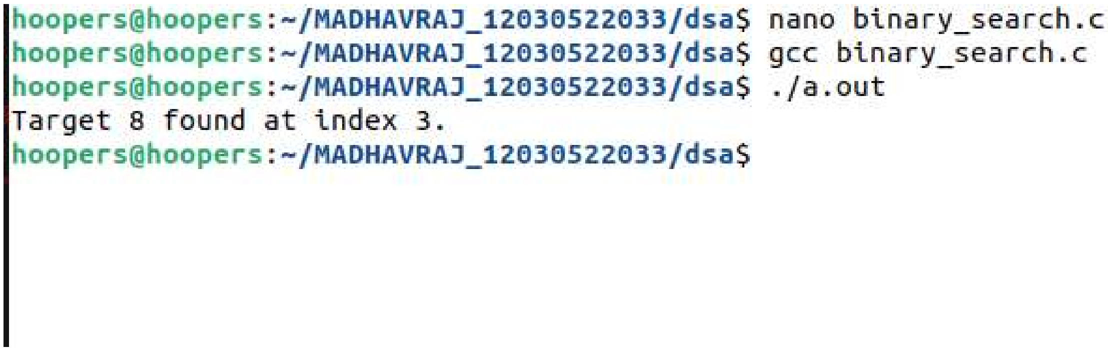
Program2: WAP to execute Binary Search:

INPUT:



```
#include <stdio.h>

int main() {
    int arr[] = {2, 4, 6, 8, 10, 12, 14};
    int target = 8;
    int n = sizeof(arr) / sizeof(arr[0]);
    int found = 0;
    int low = 0;
    int high = n - 1;

    while (low <= high) {
        int mid = (low + high) / 2;

        if (arr[mid] == target) {
            printf("Target %d found at index %d.\n", target, mid);
            found = 1;
            break; // Exit the loop once the target is found
        } else if (arr[mid] < target) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }

    if (!found) {
        printf("Target %d not found in the list.\n", target);
    }

    return 0;
}
```

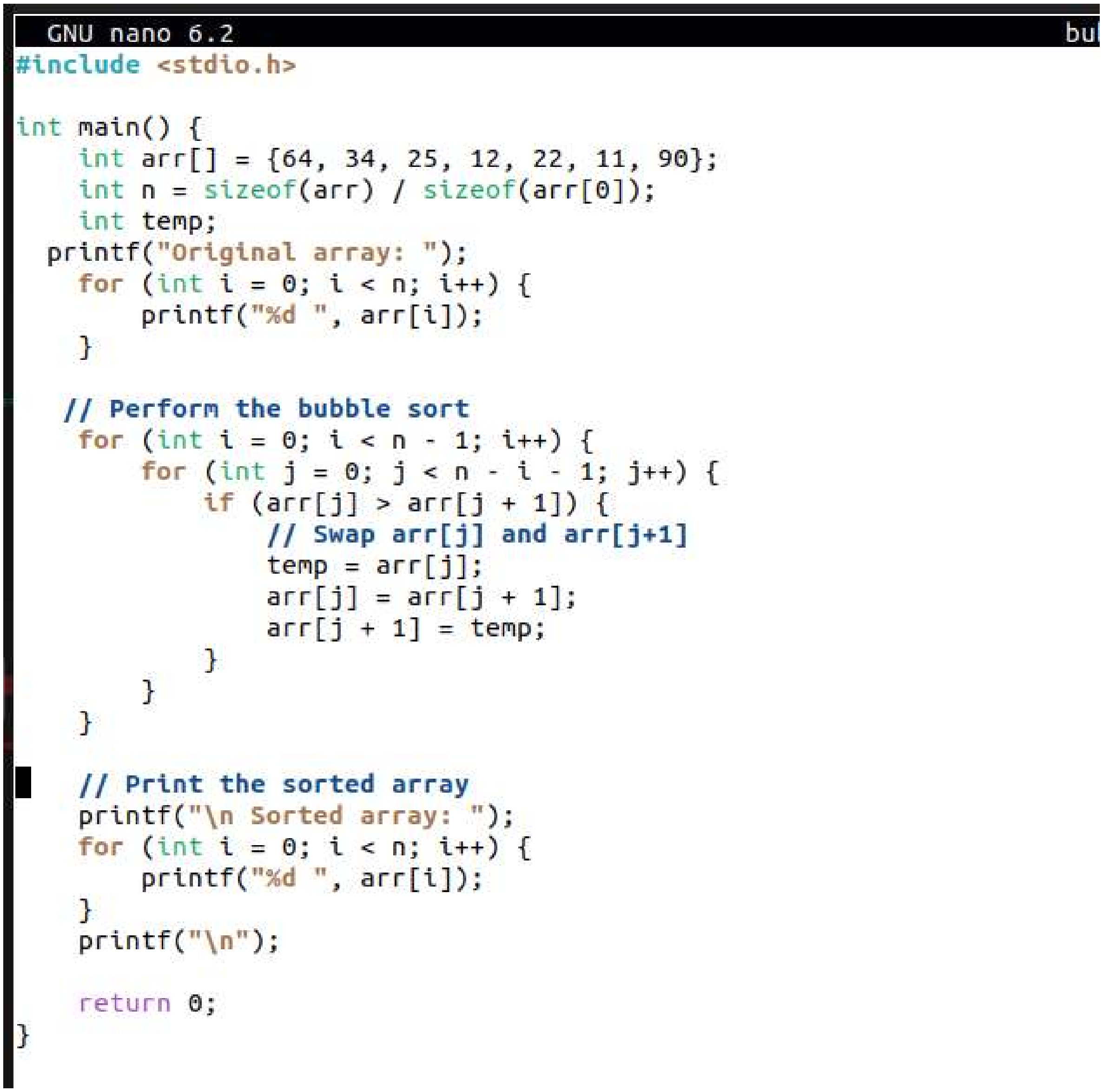
OUTPUT:



```
hoopers@hoopers:~/MADHAVRAJ_12030522033/dsa$ nano binary_search.c
hoopers@hoopers:~/MADHAVRAJ_12030522033/dsa$ gcc binary_search.c
hoopers@hoopers:~/MADHAVRAJ_12030522033/dsa$ ./a.out
Target 8 found at index 3.
hoopers@hoopers:~/MADHAVRAJ_12030522033/dsa$
```

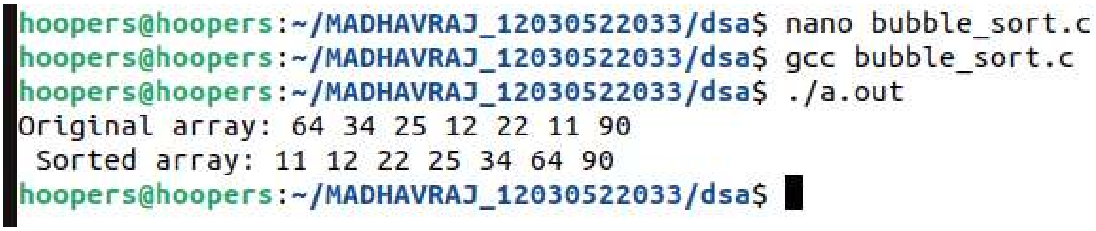
Program3: WAP to execute Bubble Sort:

INPUT:



```
#include <stdio.h>

int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr) / sizeof(arr[0]);
    int temp;
  printf("Original array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }

   // Perform the bubble sort
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                // Swap arr[j] and arr[j+1]
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }

    // Print the sorted array
    printf("\n Sorted array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    return 0;
}
```

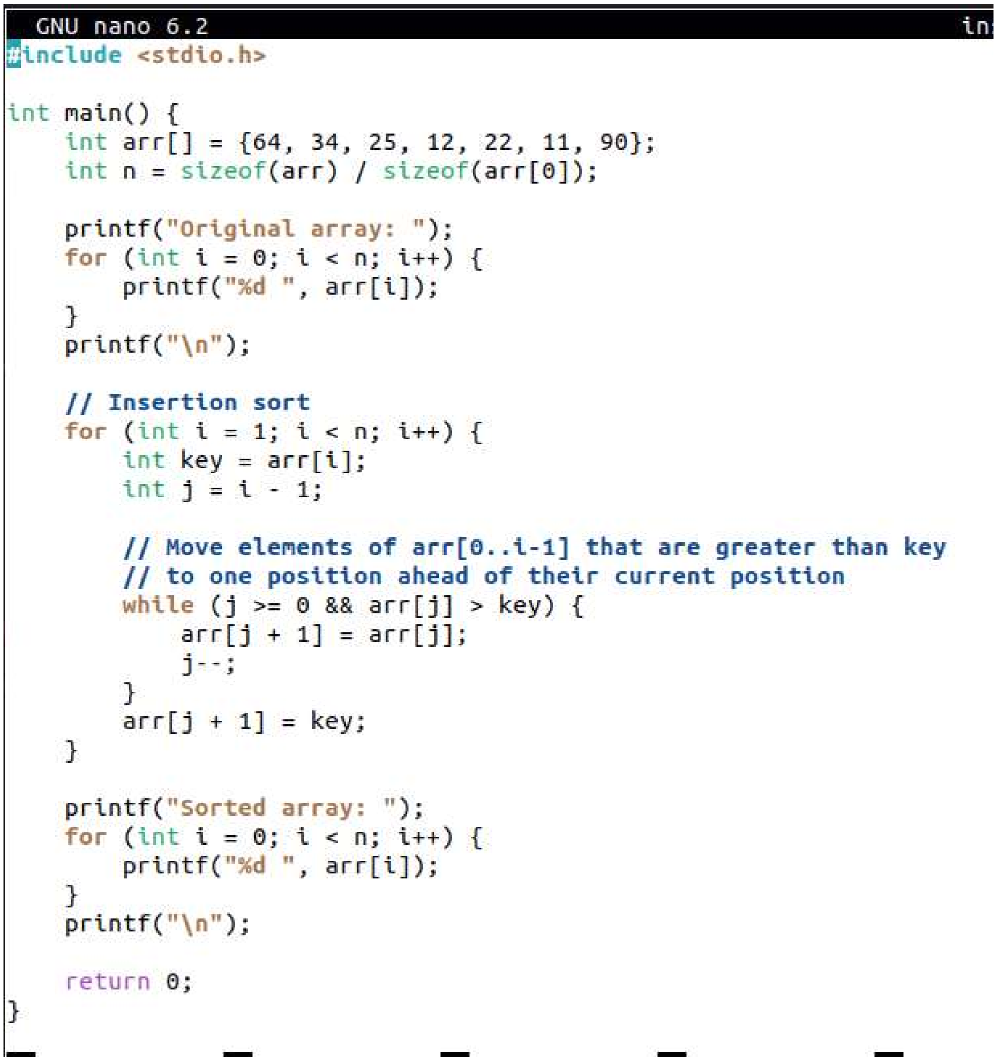
OUTPUT:



```
hoopers@hoopers:~/MADHAVRAJ_12030522033/dsa$ nano bubble_sort.c
hoopers@hoopers:~/MADHAVRAJ_12030522033/dsa$ gcc bubble_sort.c
hoopers@hoopers:~/MADHAVRAJ_12030522033/dsa$ ./a.out
Original array: 64 34 25 12 22 11 90
 Sorted array: 11 12 22 25 34 64 90
hoopers@hoopers:~/MADHAVRAJ_12030522033/dsa$
```

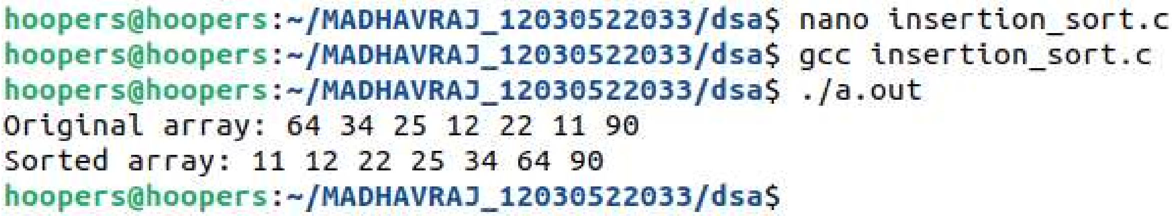
Program4: WAP to execute Insertion Sort:

INPUT:



```c
#include <stdio.h>

int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr) / sizeof(arr[0]);

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

    // Insertion sort
    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;

        // Move elements of arr[0..i-1] that are greater than key
        // to one position ahead of their current position
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }

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

    return 0;
}
```

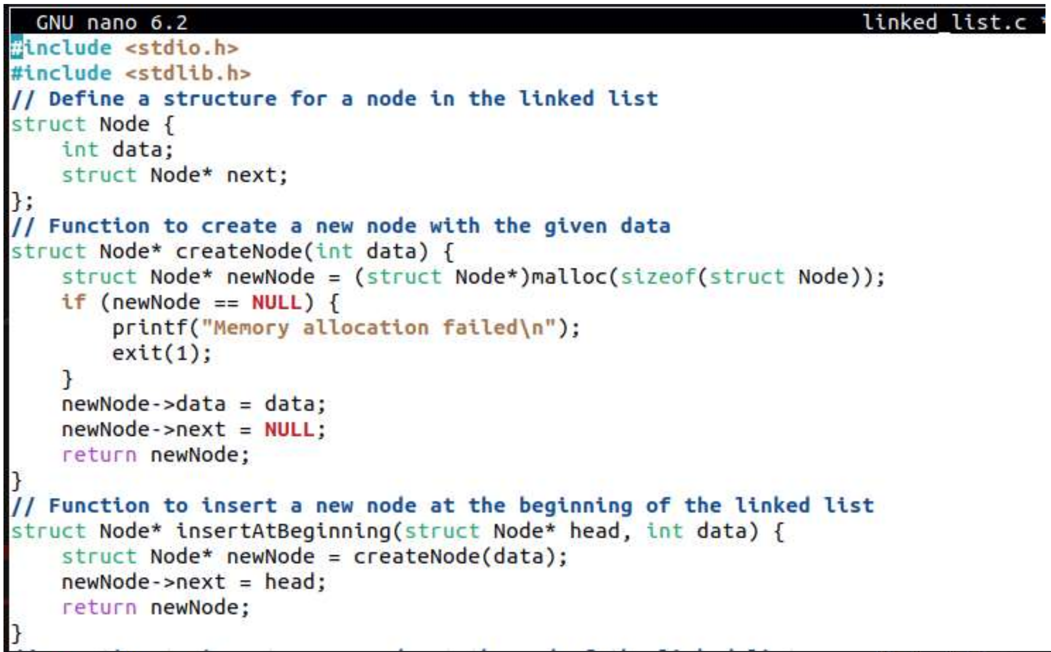
OUTPUT:



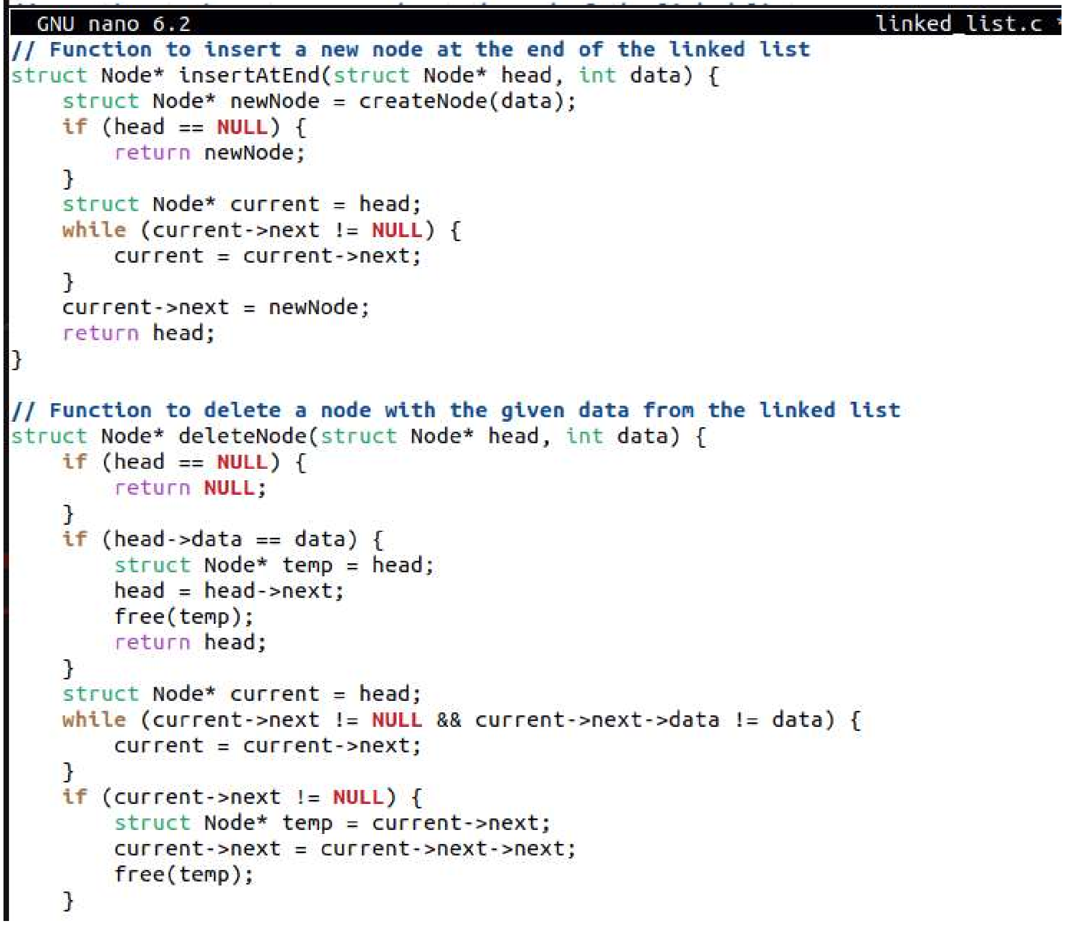
```
hoopers@hoopers:~/MADHAVRAJ_12030522033/dsa$ nano insertion_sort.c
hoopers@hoopers:~/MADHAVRAJ_12030522033/dsa$ gcc insertion_sort.c
hoopers@hoopers:~/MADHAVRAJ_12030522033/dsa$ ./a.out
Original array: 64 34 25 12 22 11 90
Sorted array: 11 12 22 25 34 64 90
hoopers@hoopers:~/MADHAVRAJ_12030522033/dsa$
```

Program5: WAP to execute Linked List (traversal, insertion & deletion):

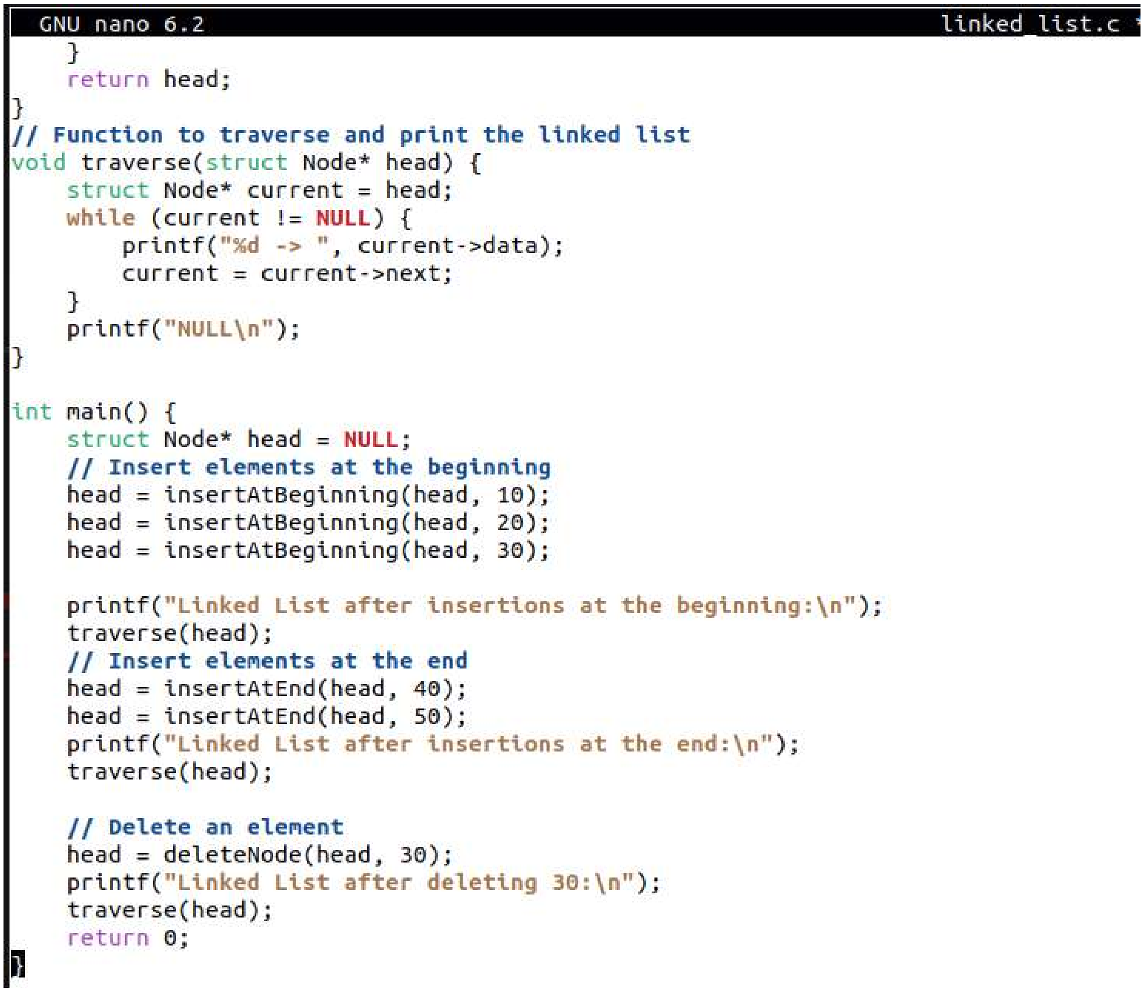
INPUT:



```c
#include <stdio.h>
#include <stdlib.h>
// Define a structure for a node in the linked list
struct Node {
    int data;
    struct Node* next;
};
// Function to create a new node with the given data
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}
// Function to insert a new node at the beginning of the linked list
struct Node* insertAtBeginning(struct Node* head, int data) {
    struct Node* newNode = createNode(data);
    newNode->next = head;
    return newNode;
}
```

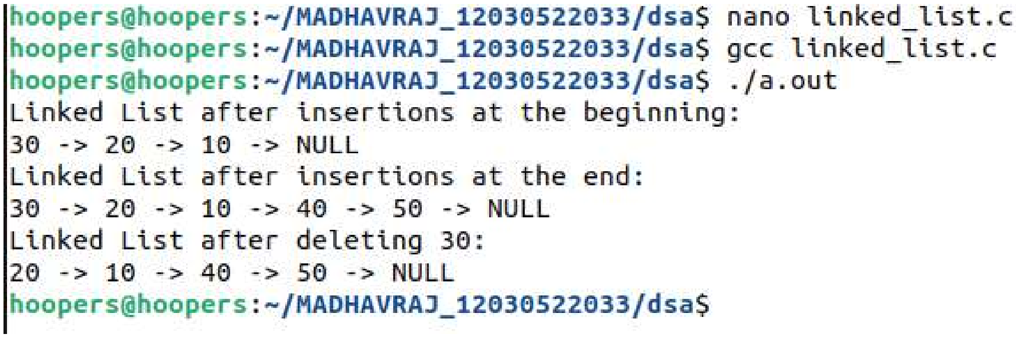


```c
// Function to insert a new node at the end of the linked list
struct Node* insertAtEnd(struct Node* head, int data) {
    struct Node* newNode = createNode(data);
    if (head == NULL) {
        return newNode;
    }
    struct Node* current = head;
    while (current->next != NULL) {
        current = current->next;
    }
    current->next = newNode;
    return head;
}

// Function to delete a node with the given data from the linked list
struct Node* deleteNode(struct Node* head, int data) {
    if (head == NULL) {
        return NULL;
    }
    if (head->data == data) {
        struct Node* temp = head;
        head = head->next;
        free(temp);
        return head;
    }
    struct Node* current = head;
    while (current->next != NULL && current->next->data != data) {
        current = current->next;
    }
    if (current->next != NULL) {
        struct Node* temp = current->next;
        current->next = current->next->next;
        free(temp);
    }
```



```
    }
    return head;
}
// Function to traverse and print the linked list
void traverse(struct Node* head) {
    struct Node* current = head;
    while (current != NULL) {
        printf("%d -> ", current->data);
        current = current->next;
    }
    printf("NULL\n");
}

int main() {
    struct Node* head = NULL;
    // Insert elements at the beginning
    head = insertAtBeginning(head, 10);
    head = insertAtBeginning(head, 20);
    head = insertAtBeginning(head, 30);

    printf("Linked List after insertions at the beginning:\n");
    traverse(head);
    // Insert elements at the end
    head = insertAtEnd(head, 40);
    head = insertAtEnd(head, 50);
    printf("Linked List after insertions at the end:\n");
    traverse(head);

    // Delete an element
    head = deleteNode(head, 30);
    printf("Linked List after deleting 30:\n");
    traverse(head);
    return 0;
}
```

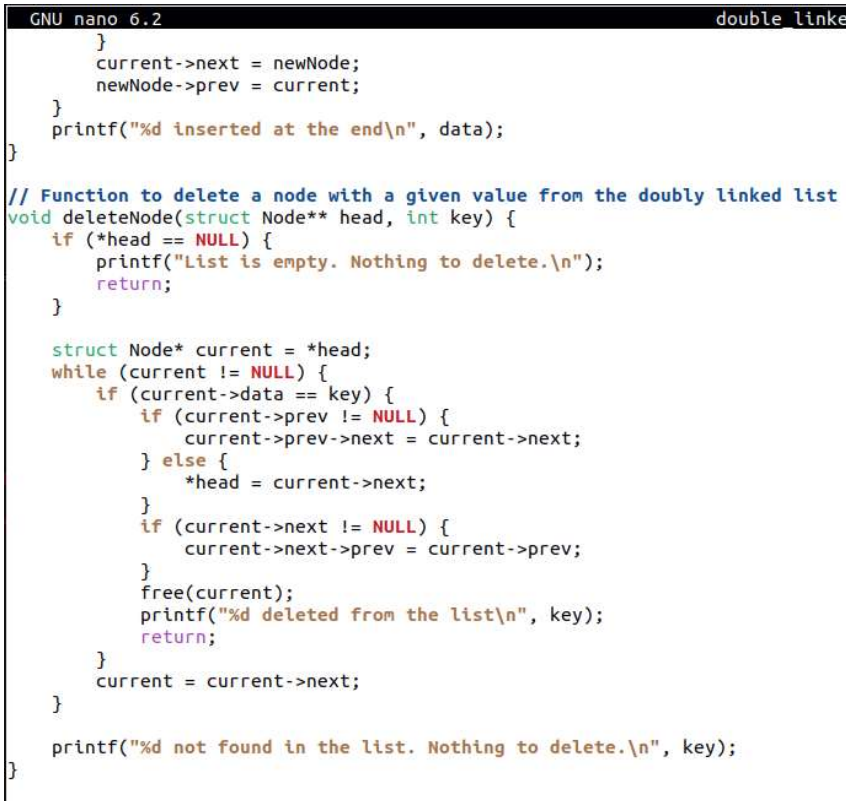
OUTPUT:



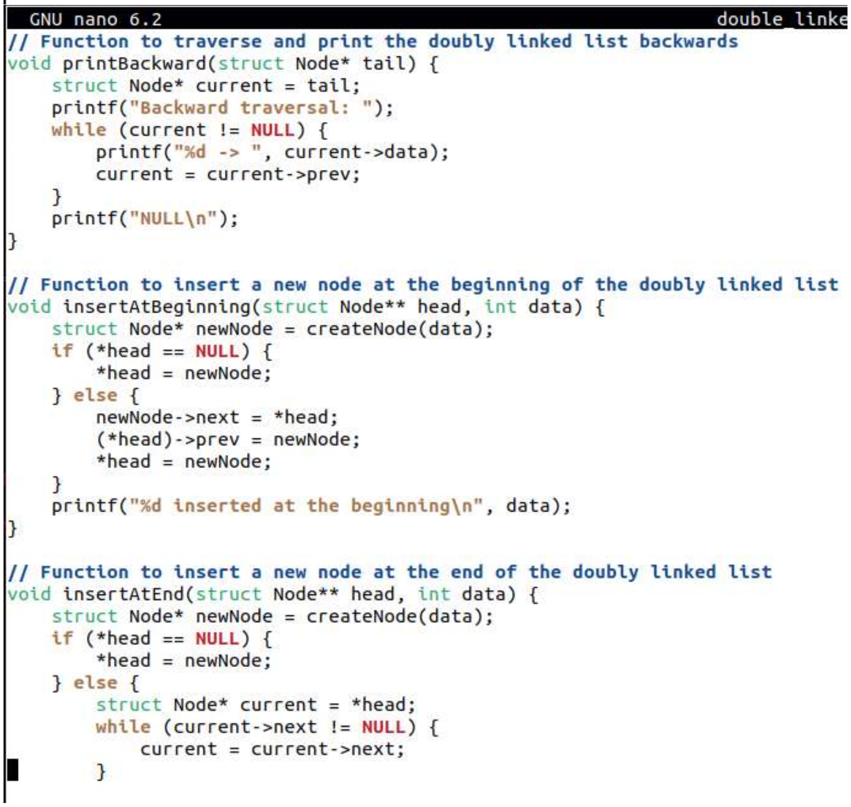
```
hoopers@hoopers:~/MADHAVRAJ_12030522033/dsa$ nano linked_list.c
hoopers@hoopers:~/MADHAVRAJ_12030522033/dsa$ gcc linked_list.c
hoopers@hoopers:~/MADHAVRAJ_12030522033/dsa$ ./a.out
Linked List after insertions at the beginning:
30 -> 20 -> 10 -> NULL
Linked List after insertions at the end:
30 -> 20 -> 10 -> 40 -> 50 -> NULL
Linked List after deleting 30:
20 -> 10 -> 40 -> 50 -> NULL
hoopers@hoopers:~/MADHAVRAJ_12030522033/dsa$
```

Program6: WAP to execute Double Linked List (traversal, insertion & deletion):

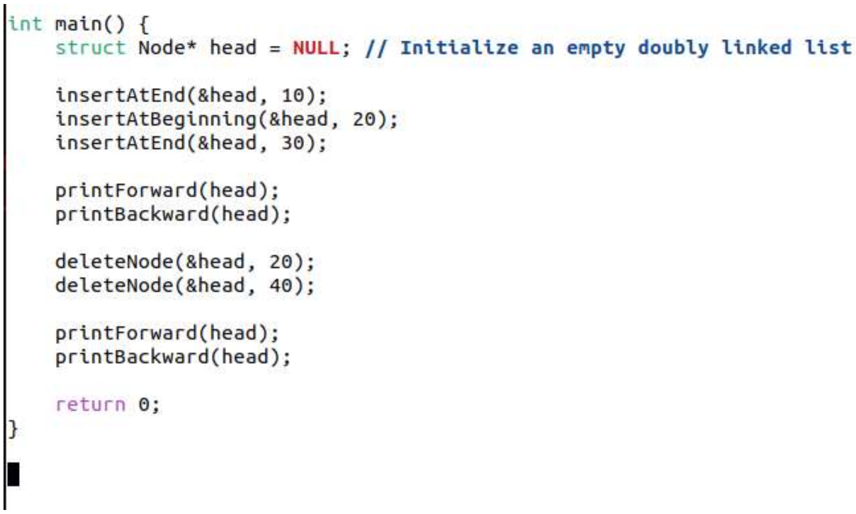
INPUT:



```
        }
        current->next = newNode;
        newNode->prev = current;
    }
    printf("%d inserted at the end\n", data);
}

// Function to delete a node with a given value from the doubly linked list
void deleteNode(struct Node** head, int key) {
    if (*head == NULL) {
        printf("List is empty. Nothing to delete.\n");
        return;
    }

    struct Node* current = *head;
    while (current != NULL) {
        if (current->data == key) {
            if (current->prev != NULL) {
                current->prev->next = current->next;
            } else {
                *head = current->next;
            }
            if (current->next != NULL) {
                current->next->prev = current->prev;
            }
            free(current);
            printf("%d deleted from the list\n", key);
            return;
        }
        current = current->next;
    }

    printf("%d not found in the list. Nothing to delete.\n", key);
}
```

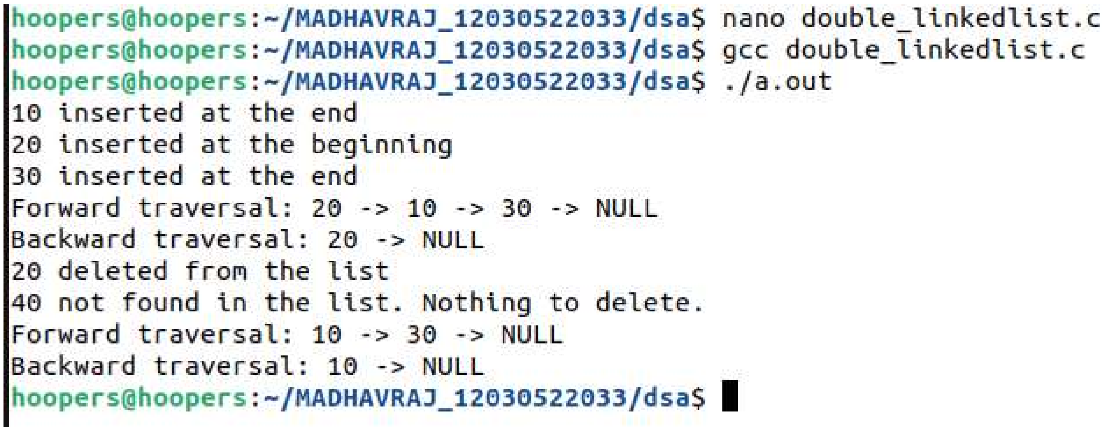


```
// Function to traverse and print the doubly linked list backwards
void printBackward(struct Node* tail) {
    struct Node* current = tail;
    printf("Backward traversal: ");
    while (current != NULL) {
        printf("%d -> ", current->data);
        current = current->prev;
    }
    printf("NULL\n");
}

// Function to insert a new node at the beginning of the doubly linked list
void insertAtBeginning(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
    } else {
        newNode->next = *head;
        (*head)->prev = newNode;
        *head = newNode;
    }
    printf("%d inserted at the beginning\n", data);
}

// Function to insert a new node at the end of the doubly linked list
void insertAtEnd(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
    } else {
        struct Node* current = *head;
        while (current->next != NULL) {
            current = current->next;
        }
```



```c
int main() {
    struct Node* head = NULL; // Initialize an empty doubly linked list

    insertAtEnd(&head, 10);
    insertAtBeginning(&head, 20);
    insertAtEnd(&head, 30);

    printForward(head);
    printBackward(head);

    deleteNode(&head, 20);
    deleteNode(&head, 40);

    printForward(head);
    printBackward(head);

    return 0;
}
```

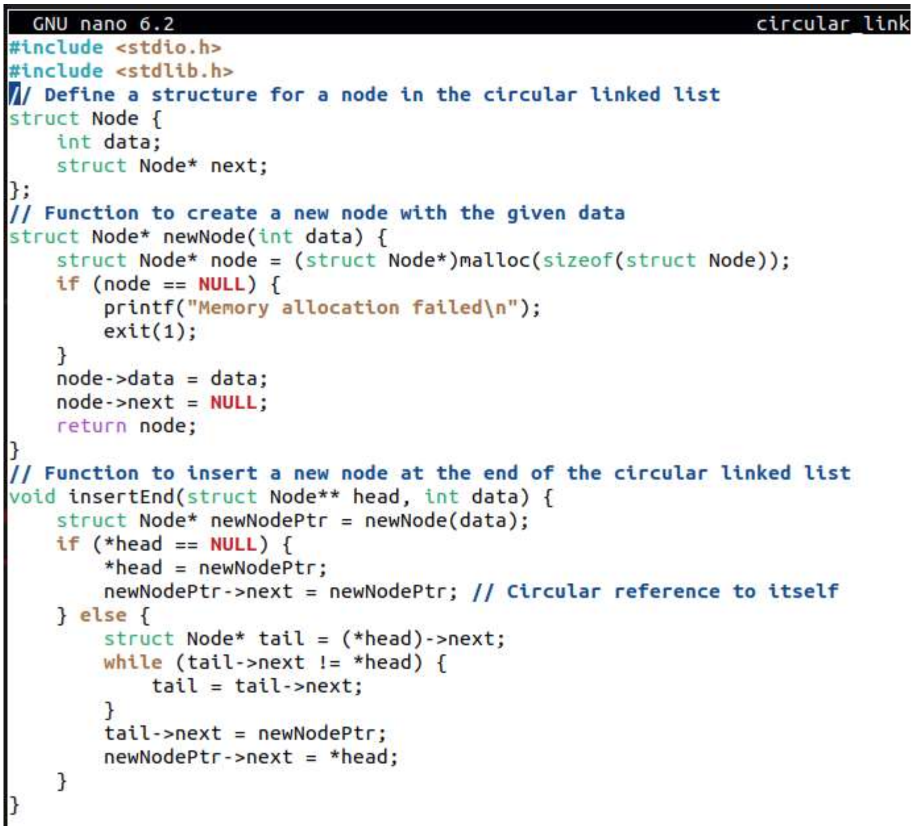
OUTPUT:



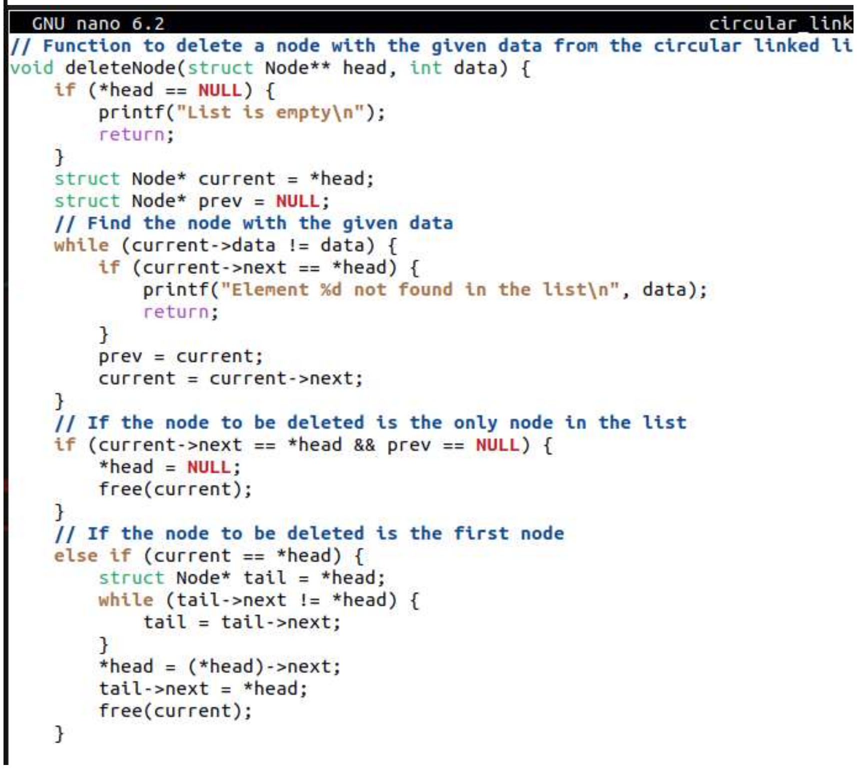
```
hoopers@hoopers:~/MADHAVRAJ_12030522033/dsa$ nano double_linkedlist.c
hoopers@hoopers:~/MADHAVRAJ_12030522033/dsa$ gcc double_linkedlist.c
hoopers@hoopers:~/MADHAVRAJ_12030522033/dsa$ ./a.out
10 inserted at the end
20 inserted at the beginning
30 inserted at the end
Forward traversal: 20 -> 10 -> 30 -> NULL
Backward traversal: 20 -> NULL
20 deleted from the list
40 not found in the list. Nothing to delete.
Forward traversal: 10 -> 30 -> NULL
Backward traversal: 10 -> NULL
hoopers@hoopers:~/MADHAVRAJ_12030522033/dsa$
```

Program7: WAP to execute Circular Linked List (traversal, insertion & deletion):

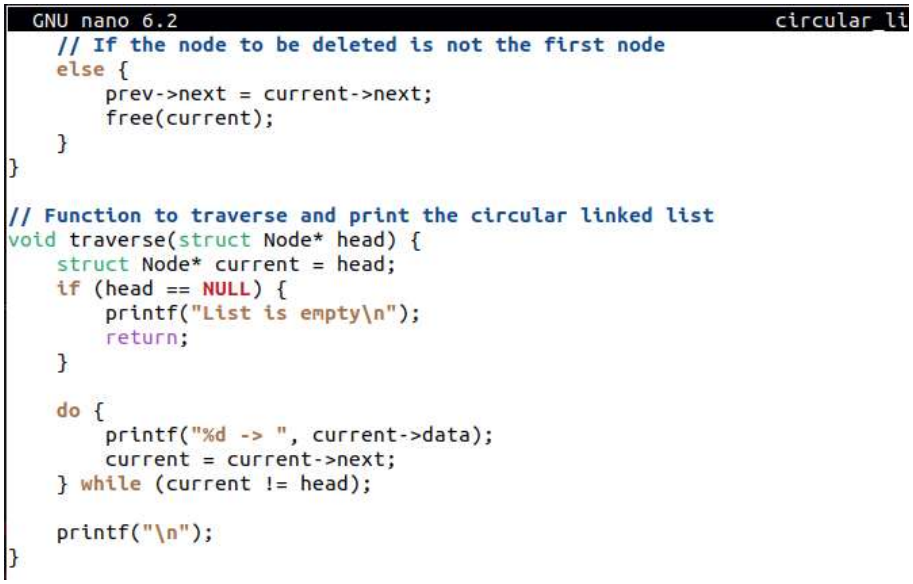
INPUT:



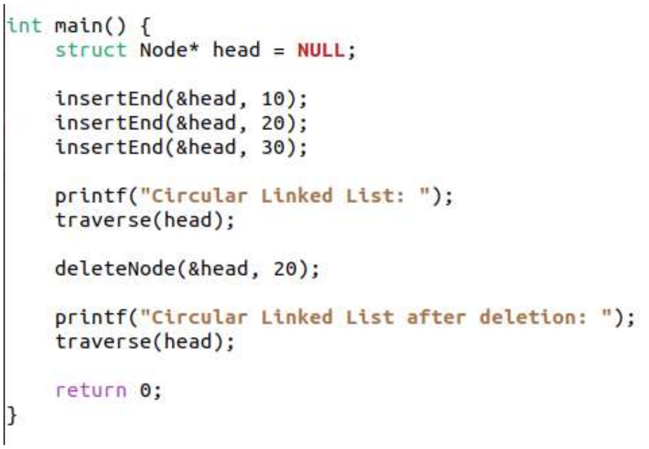
```
#include <stdio.h>
#include <stdlib.h>
// Define a structure for a node in the circular linked list
struct Node {
    int data;
    struct Node* next;
};
// Function to create a new node with the given data
struct Node* newNode(int data) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    if (node == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }
    node->data = data;
    node->next = NULL;
    return node;
}
// Function to insert a new node at the end of the circular linked list
void insertEnd(struct Node** head, int data) {
    struct Node* newNodePtr = newNode(data);
    if (*head == NULL) {
        *head = newNodePtr;
        newNodePtr->next = newNodePtr; // Circular reference to itself
    } else {
        struct Node* tail = (*head)->next;
        while (tail->next != *head) {
            tail = tail->next;
        }
        tail->next = newNodePtr;
        newNodePtr->next = *head;
    }
}
```



```
// Function to delete a node with the given data from the circular linked li
void deleteNode(struct Node** head, int data) {
    if (*head == NULL) {
        printf("List is empty\n");
        return;
    }
    struct Node* current = *head;
    struct Node* prev = NULL;
    // Find the node with the given data
    while (current->data != data) {
        if (current->next == *head) {
            printf("Element %d not found in the list\n", data);
            return;
        }
        prev = current;
        current = current->next;
    }
    // If the node to be deleted is the only node in the list
    if (current->next == *head && prev == NULL) {
        *head = NULL;
        free(current);
    }
    // If the node to be deleted is the first node
    else if (current == *head) {
        struct Node* tail = *head;
        while (tail->next != *head) {
            tail = tail->next;
        }
        *head = (*head)->next;
        tail->next = *head;
        free(current);
    }
```

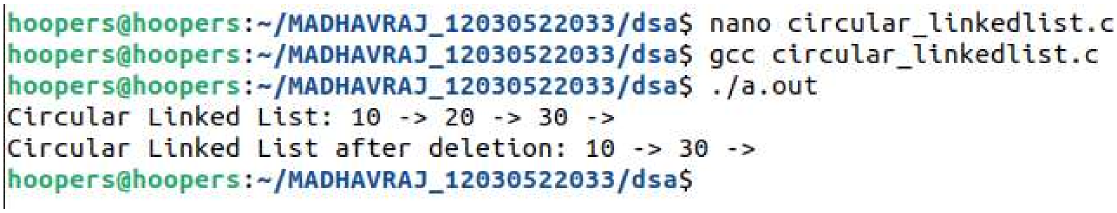


```
    // If the node to be deleted is not the first node
    else {
        prev->next = current->next;
        free(current);
    }
}

// Function to traverse and print the circular linked list
void traverse(struct Node* head) {
    struct Node* current = head;
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }

    do {
        printf("%d -> ", current->data);
        current = current->next;
    } while (current != head);

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



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

    insertEnd(&head, 10);
    insertEnd(&head, 20);
    insertEnd(&head, 30);

    printf("Circular Linked List: ");
    traverse(head);

    deleteNode(&head, 20);

    printf("Circular Linked List after deletion: ");
    traverse(head);

    return 0;
}
```

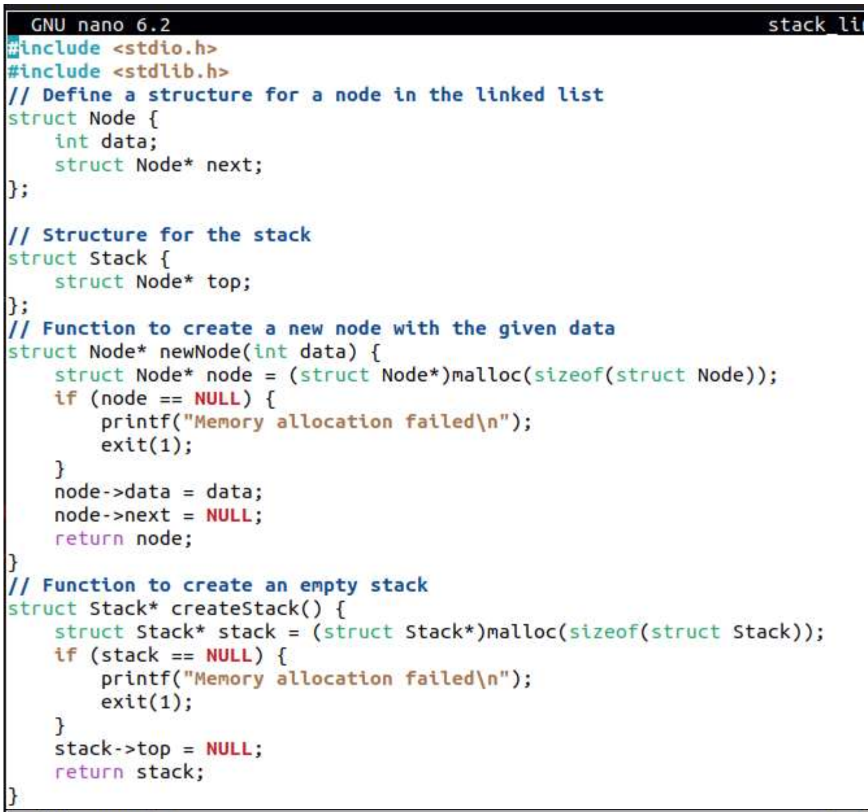
OUTPUT:



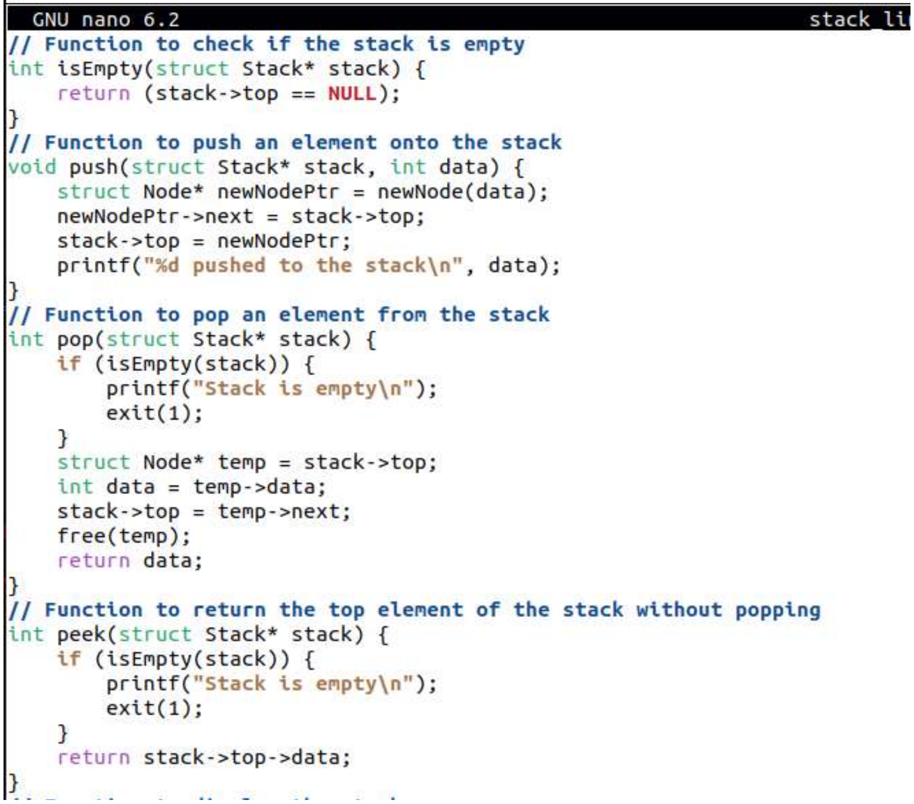
```
hoopers@hoopers:~/MADHAVRAJ_12030522033/dsa$ nano circular_linkedlist.c
hoopers@hoopers:~/MADHAVRAJ_12030522033/dsa$ gcc circular_linkedlist.c
hoopers@hoopers:~/MADHAVRAJ_12030522033/dsa$ ./a.out
Circular Linked List: 10 -> 20 -> 30 ->
Circular Linked List after deletion: 10 -> 30 ->
hoopers@hoopers:~/MADHAVRAJ_12030522033/dsa$
```

Program8: WAP to execute Stack using Linked List:

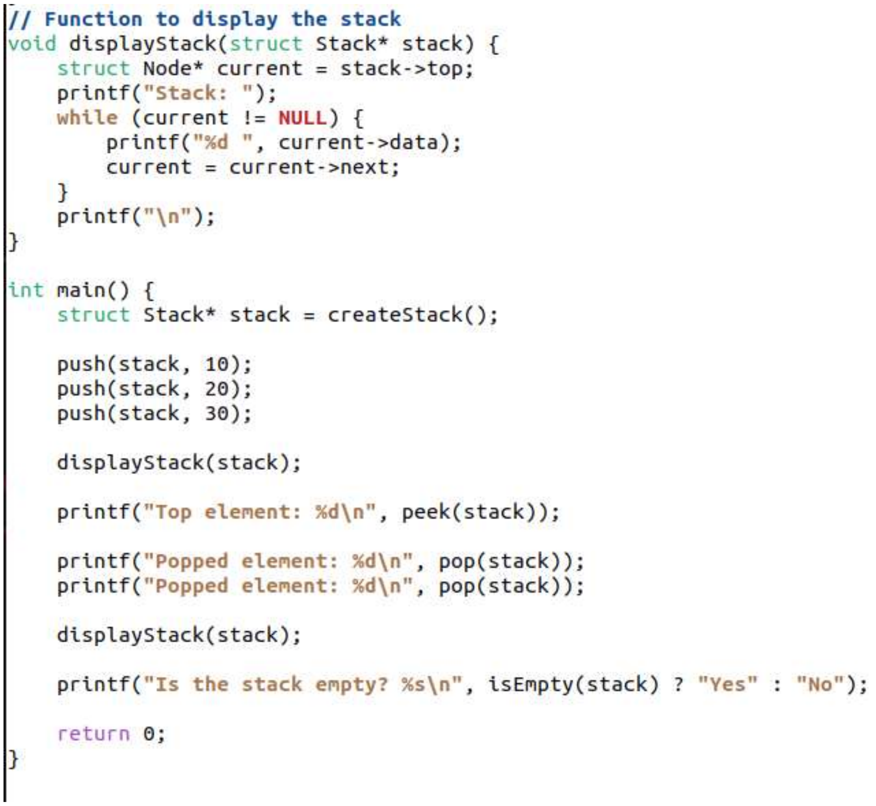
INPUT:



```
#include <stdio.h>
#include <stdlib.h>
// Define a structure for a node in the linked list
struct Node {
    int data;
    struct Node* next;
};

// Structure for the stack
struct Stack {
    struct Node* top;
};
// Function to create a new node with the given data
struct Node* newNode(int data) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    if (node == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }
    node->data = data;
    node->next = NULL;
    return node;
}
// Function to create an empty stack
struct Stack* createStack() {
    struct Stack* stack = (struct Stack*)malloc(sizeof(struct Stack));
    if (stack == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }
    stack->top = NULL;
    return stack;
}
```

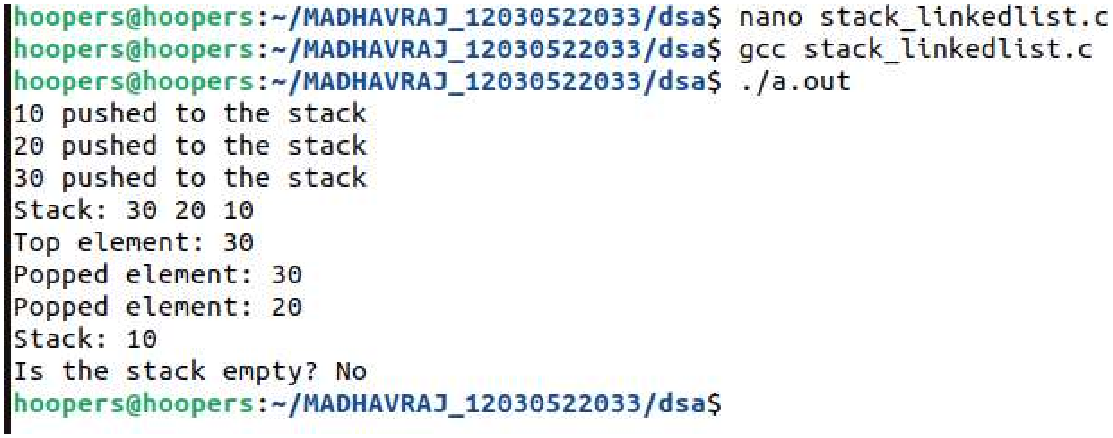


```
// Function to check if the stack is empty
int isEmpty(struct Stack* stack) {
    return (stack->top == NULL);
}
// Function to push an element onto the stack
void push(struct Stack* stack, int data) {
    struct Node* newNodePtr = newNode(data);
    newNodePtr->next = stack->top;
    stack->top = newNodePtr;
    printf("%d pushed to the stack\n", data);
}
// Function to pop an element from the stack
int pop(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty\n");
        exit(1);
    }
    struct Node* temp = stack->top;
    int data = temp->data;
    stack->top = temp->next;
    free(temp);
    return data;
}
// Function to return the top element of the stack without popping
int peek(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty\n");
        exit(1);
    }
    return stack->top->data;
}
```



```c
// Function to display the stack
void displayStack(struct Stack* stack) {
    struct Node* current = stack->top;
    printf("Stack: ");
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
}

int main() {
    struct Stack* stack = createStack();

    push(stack, 10);
    push(stack, 20);
    push(stack, 30);

    displayStack(stack);

    printf("Top element: %d\n", peek(stack));

    printf("Popped element: %d\n", pop(stack));
    printf("Popped element: %d\n", pop(stack));

    displayStack(stack);

    printf("Is the stack empty? %s\n", isEmpty(stack) ? "Yes" : "No");

    return 0;
}
```

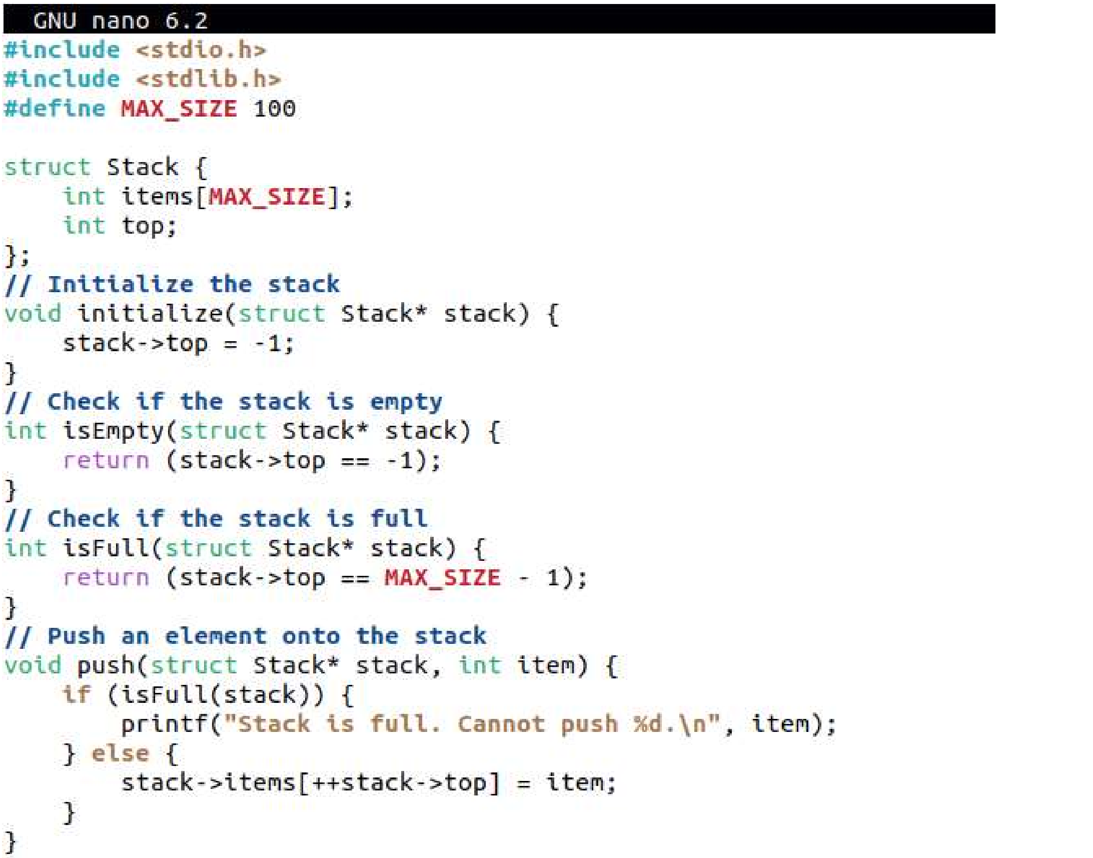
OUTPUT:



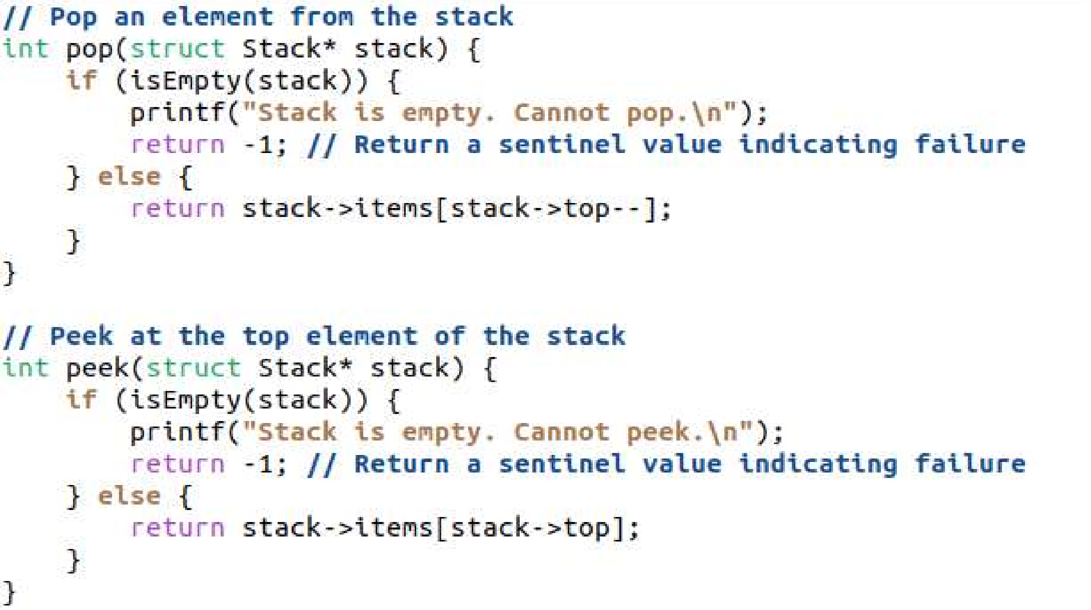
```
hoopers@hoopers:~/MADHAVRAJ_12030522033/dsa$ nano stack_linkedlist.c
hoopers@hoopers:~/MADHAVRAJ_12030522033/dsa$ gcc stack_linkedlist.c
hoopers@hoopers:~/MADHAVRAJ_12030522033/dsa$ ./a.out
10 pushed to the stack
20 pushed to the stack
30 pushed to the stack
Stack: 30 20 10
Top element: 30
Popped element: 30
Popped element: 20
Stack: 10
Is the stack empty? No
hoopers@hoopers:~/MADHAVRAJ_12030522033/dsa$
```

Program9: WAP to execute Stack:

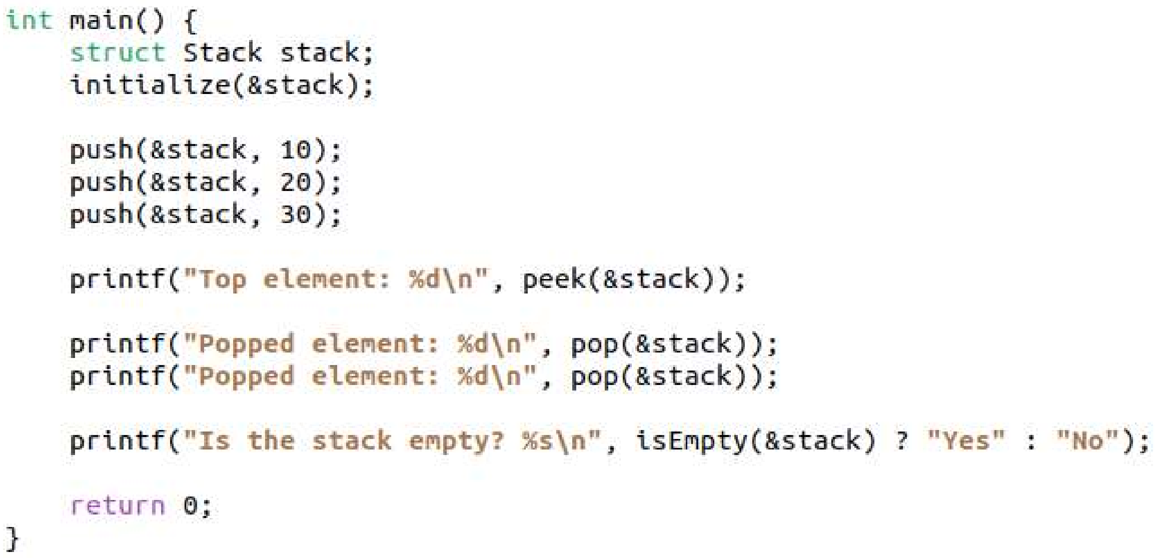
INPUT:



```c
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 100

struct Stack {
    int items[MAX_SIZE];
    int top;
};
// Initialize the stack
void initialize(struct Stack* stack) {
    stack->top = -1;
}
// Check if the stack is empty
int isEmpty(struct Stack* stack) {
    return (stack->top == -1);
}
// Check if the stack is full
int isFull(struct Stack* stack) {
    return (stack->top == MAX_SIZE - 1);
}
// Push an element onto the stack
void push(struct Stack* stack, int item) {
    if (isFull(stack)) {
        printf("Stack is full. Cannot push %d.\n", item);
    } else {
        stack->items[++stack->top] = item;
    }
}
```

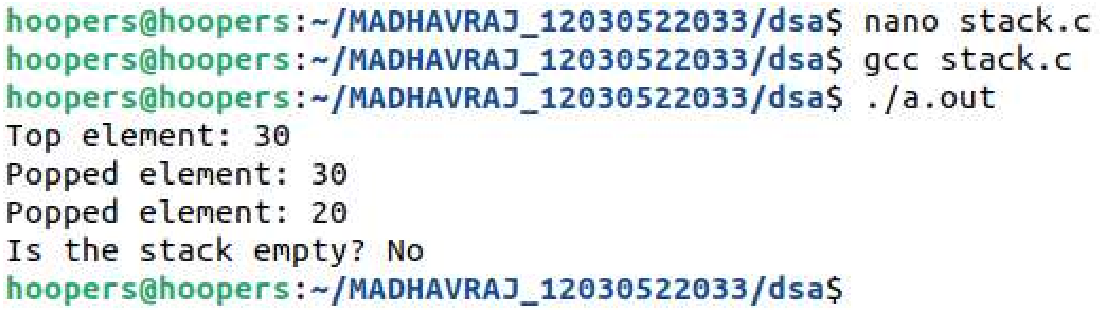


```c
// Pop an element from the stack
int pop(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty. Cannot pop.\n");
        return -1; // Return a sentinel value indicating failure
    } else {
        return stack->items[stack->top--];
    }
}

// Peek at the top element of the stack
int peek(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("Stack is empty. Cannot peek.\n");
        return -1; // Return a sentinel value indicating failure
    } else {
        return stack->items[stack->top];
    }
}
```



```c
int main() {
    struct Stack stack;
    initialize(&stack);

    push(&stack, 10);
    push(&stack, 20);
    push(&stack, 30);

    printf("Top element: %d\n", peek(&stack));

    printf("Popped element: %d\n", pop(&stack));
    printf("Popped element: %d\n", pop(&stack));

    printf("Is the stack empty? %s\n", isEmpty(&stack) ? "Yes" : "No");

    return 0;
}
```

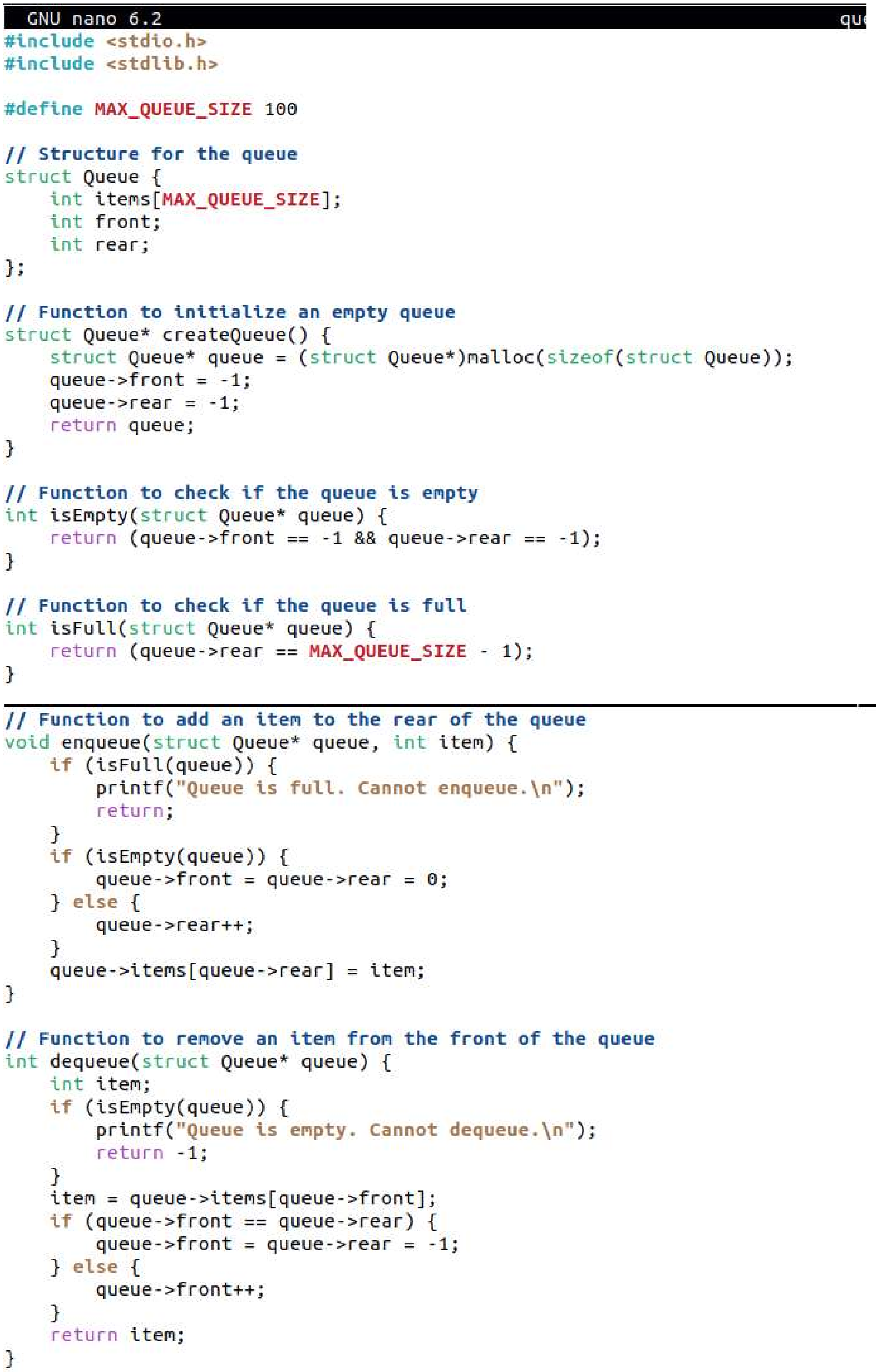
OUTPUT:



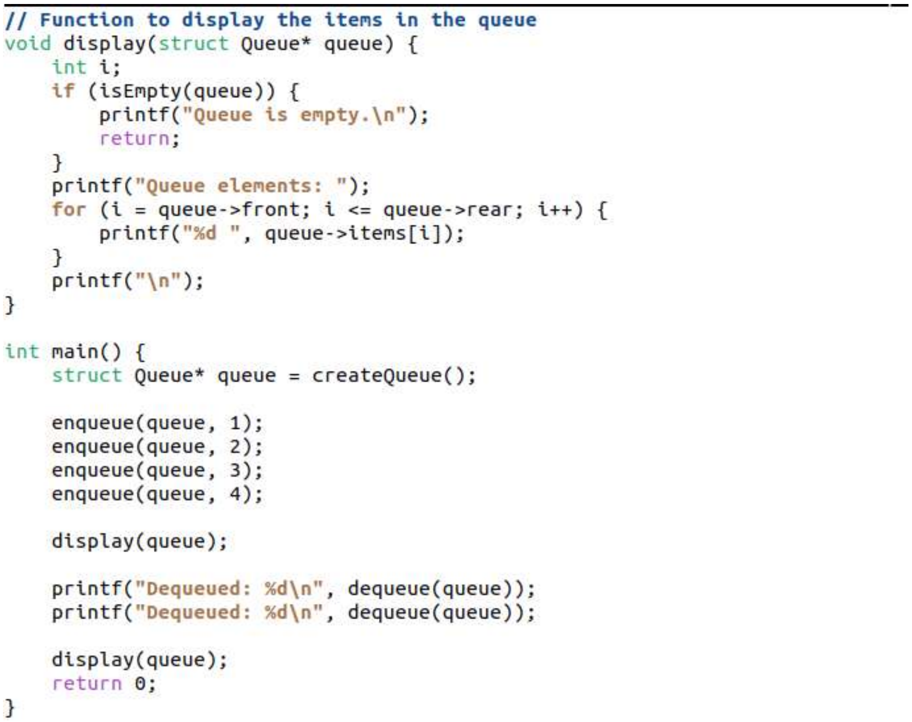
```
hoopers@hoopers:~/MADHAVRAJ_12030522033/dsa$ nano stack.c
hoopers@hoopers:~/MADHAVRAJ_12030522033/dsa$ gcc stack.c
hoopers@hoopers:~/MADHAVRAJ_12030522033/dsa$ ./a.out
Top element: 30
Popped element: 30
Popped element: 20
Is the stack empty? No
hoopers@hoopers:~/MADHAVRAJ_12030522033/dsa$
```

Program10: WAP to execute Queue:

INPUT:

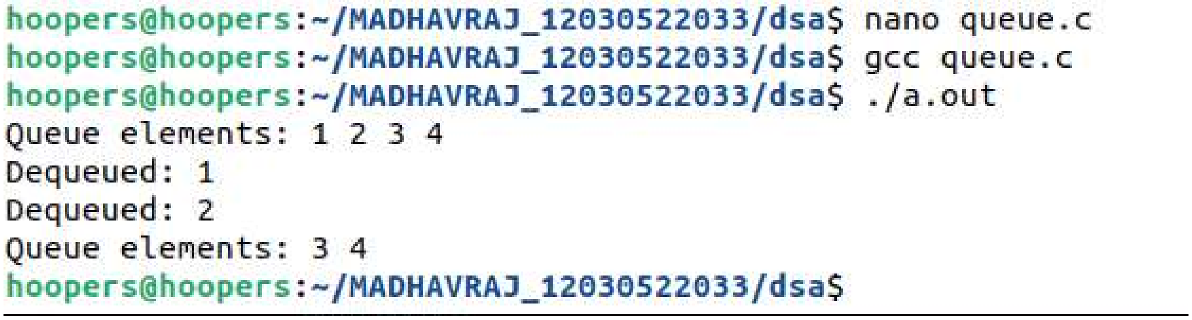


```
  GNU nano 6.2                                                      qu
#include <stdio.h>
#include <stdlib.h>

#define MAX_QUEUE_SIZE 100

// Structure for the queue
struct Queue {
    int items[MAX_QUEUE_SIZE];
    int front;
    int rear;
};

// Function to initialize an empty queue
struct Queue* createQueue() {
    struct Queue* queue = (struct Queue*)malloc(sizeof(struct Queue));
    queue->front = -1;
    queue->rear = -1;
    return queue;
}

// Function to check if the queue is empty
int isEmpty(struct Queue* queue) {
    return (queue->front == -1 && queue->rear == -1);
}

// Function to check if the queue is full
int isFull(struct Queue* queue) {
    return (queue->rear == MAX_QUEUE_SIZE - 1);
}

// Function to add an item to the rear of the queue
void enqueue(struct Queue* queue, int item) {
    if (isFull(queue)) {
        printf("Queue is full. Cannot enqueue.\n");
        return;
    }
    if (isEmpty(queue)) {
        queue->front = queue->rear = 0;
    } else {
        queue->rear++;
    }
    queue->items[queue->rear] = item;
}

// Function to remove an item from the front of the queue
int dequeue(struct Queue* queue) {
    int item;
    if (isEmpty(queue)) {
        printf("Queue is empty. Cannot dequeue.\n");
        return -1;
    }
    item = queue->items[queue->front];
    if (queue->front == queue->rear) {
        queue->front = queue->rear = -1;
    } else {
        queue->front++;
    }
    return item;
}
```



```c
// Function to display the items in the queue
void display(struct Queue* queue) {
    int i;
    if (isEmpty(queue)) {
        printf("Queue is empty.\n");
        return;
    }
    printf("Queue elements: ");
    for (i = queue->front; i <= queue->rear; i++) {
        printf("%d ", queue->items[i]);
    }
    printf("\n");
}

int main() {
    struct Queue* queue = createQueue();

    enqueue(queue, 1);
    enqueue(queue, 2);
    enqueue(queue, 3);
    enqueue(queue, 4);

    display(queue);

    printf("Dequeued: %d\n", dequeue(queue));
    printf("Dequeued: %d\n", dequeue(queue));

    display(queue);
    return 0;
}
```

OUTPUT:



```
hoopers@hoopers:~/MADHAVRAJ_12030522033/dsa$ nano queue.c
hoopers@hoopers:~/MADHAVRAJ_12030522033/dsa$ gcc queue.c
hoopers@hoopers:~/MADHAVRAJ_12030522033/dsa$ ./a.out
Queue elements: 1 2 3 4
Dequeued: 1
Dequeued: 2
Queue elements: 3 4
hoopers@hoopers:~/MADHAVRAJ_12030522033/dsa$
```