Lesson 02 Demo 12

Implementing Stacks Using a Linked List

Objective: To demonstrate the implementation of a stack using a linked list in JavaScript, covering operations like push, pop, peek, display, and clear to develop your dynamic data structure manipulation skills

Tools required: Visual Studio Code

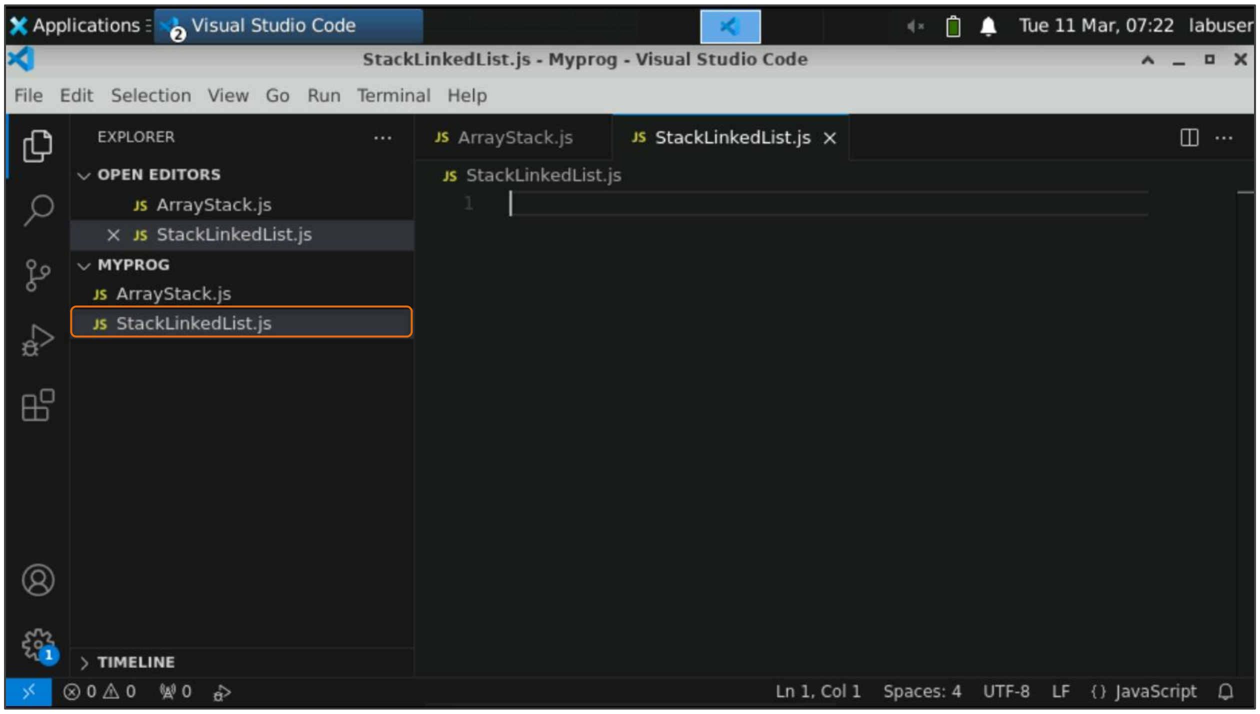
Prerequisites: A basic understanding of linked lists in JavaScript

Steps to be followed:

1. Create a JavaScript file and execute it

Step 1: Create a JavaScript file and execute it

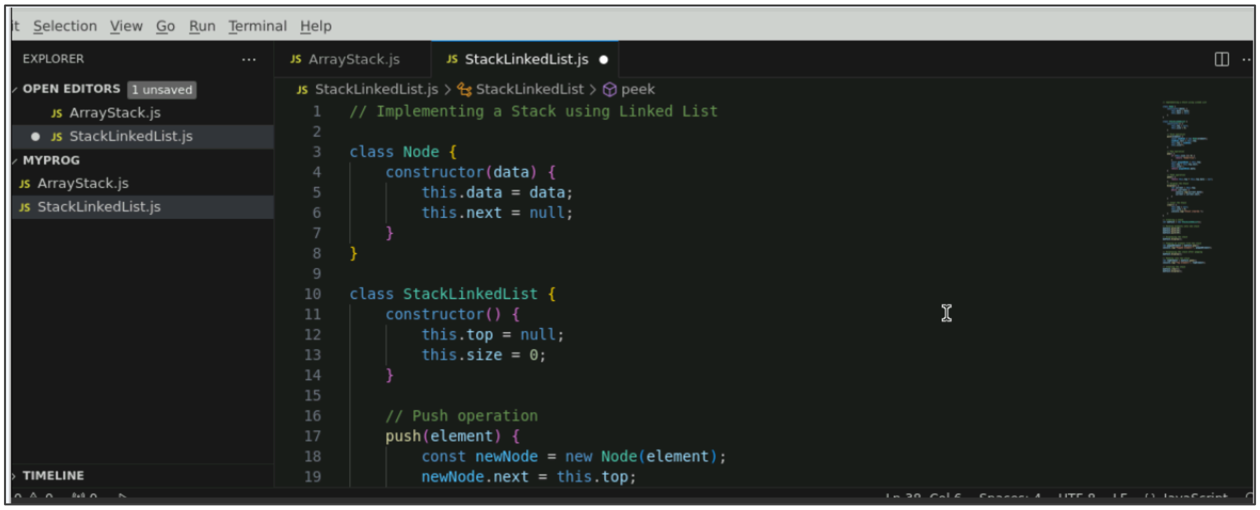
1.1 Open the Visual Studio Code editor and create a JavaScript file named **StackLinkedList.js**



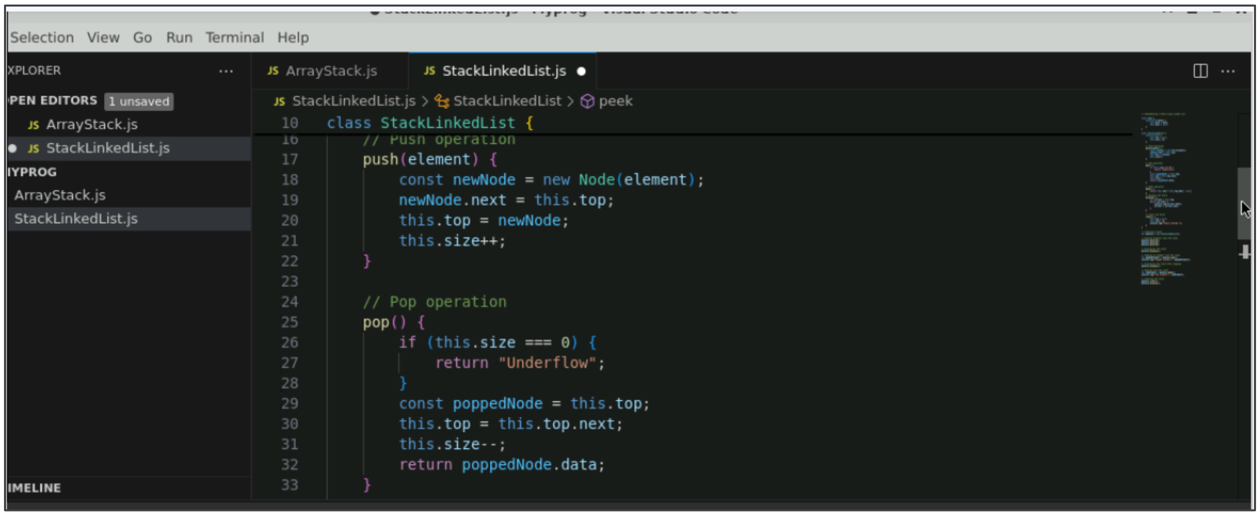
Applications
Visual Studio Code
Tue 11 Mar, 07:22 labuser
StackLinkedList.js - Myprog - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
OPEN EDITORS
ArrayStack.js
StackLinkedList.js
MYPROG
ArrayStack.js
StackLinkedList.js
TIMELINE
Ln 1, Col 1 Spaces: 4 UTF-8 LF {} JavaScript

1.2 Add the following code to the file:

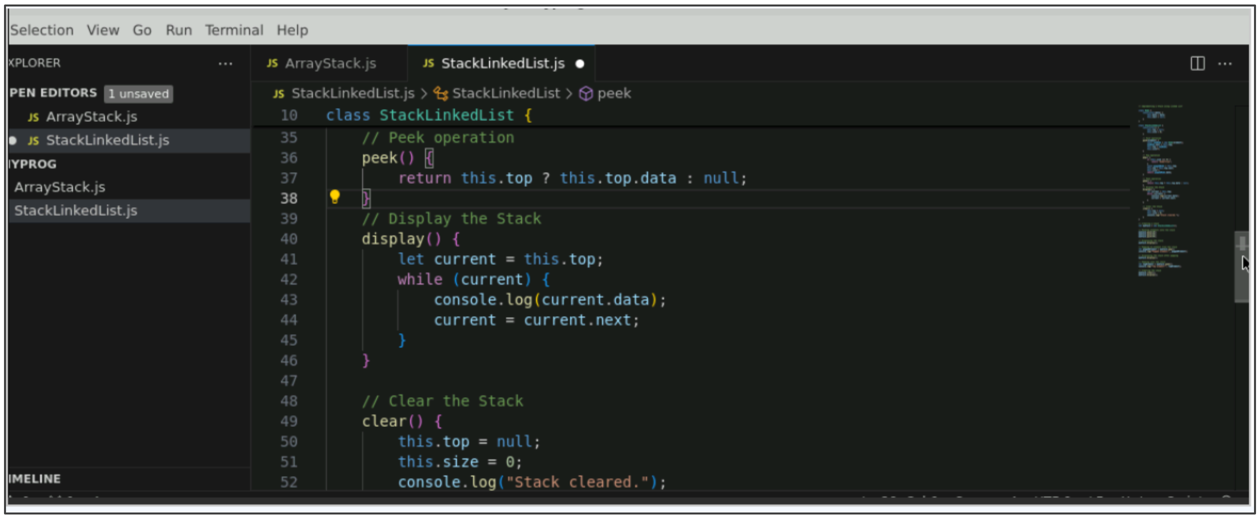
```
// Implementing a Stack using Linked List

class Node {
  constructor(data) {
    this.data = data;
    this.next = null;
  }
}

class StackLinkedList {
  constructor() {
    this.top = null;
    this.size = 0;
  }

  // Push operation
  push(element) {
    const newNode = new Node(element);
    newNode.next = this.top;
    this.top = newNode;
    this.size++;
  }

  // Pop operation
  pop() {
    if (this.size === 0) {
      return "Underflow";
    }
    const poppedNode = this.top;
    this.top = this.top.next;
    this.size--;
    return poppedNode.data;
  }

  // Peek operation
  peek() {
    return this.top ? this.top.data : null;
  }
  // Display the Stack
  display() {
    let current = this.top;
    while (current) {
      console.log(current.data);
      current = current.next;
    }
  }

  // Clear the Stack
  clear() {
    this.top = null;
    this.size = 0;
    console.log("Stack cleared.");
  }
}

// Creating a stack
let myStack = new StackLinkedList();

// Pushing elements onto the stack
myStack.push(10);
myStack.push(20);
myStack.push(30);

// Displaying the stack
myStack.display();

// Popping an element from the stack
let poppedElement = myStack.pop();
console.log("Popped element:", poppedElement);

// Displaying the stack after popping
myStack.display();

// Peeking into the stack
let topElement = myStack.peek();
console.log("Top element:", topElement);

// Clearing the stack
myStack.clear();
myStack.display();
```



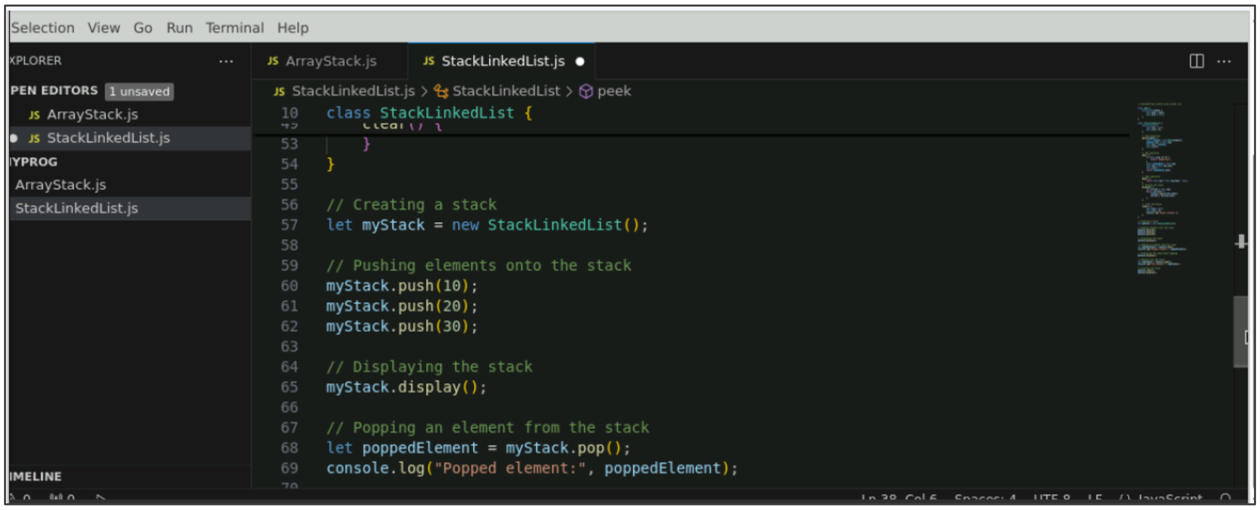
```javascript
// Implementing a Stack using Linked List

class Node {
    constructor(data) {
        this.data = data;
        this.next = null;
    }
}

class StackLinkedList {
    constructor() {
        this.top = null;
        this.size = 0;
    }

    // Push operation
    push(element) {
        const newNode = new Node(element);
        newNode.next = this.top;
```



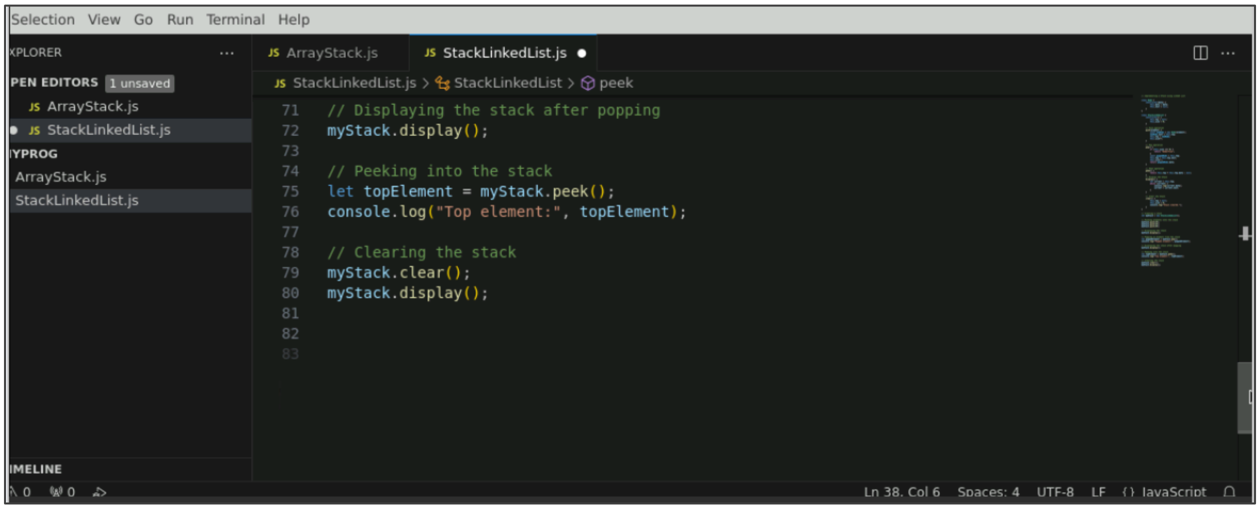
```javascript
class StackLinkedList {
    // Push operation
    push(element) {
        const newNode = new Node(element);
        newNode.next = this.top;
        this.top = newNode;
        this.size++;
    }

    // Pop operation
    pop() {
        if (this.size === 0) {
            return "Underflow";
        }
        const poppedNode = this.top;
        this.top = this.top.next;
        this.size--;
        return poppedNode.data;
    }
```



```
class StackLinkedList {
    // Peek operation
    peek() {
        return this.top ? this.top.data : null;
    }
    // Display the Stack
    display() {
        let current = this.top;
        while (current) {
            console.log(current.data);
            current = current.next;
        }
    }

    // Clear the Stack
    clear() {
        this.top = null;
        this.size = 0;
        console.log("Stack cleared.");
```

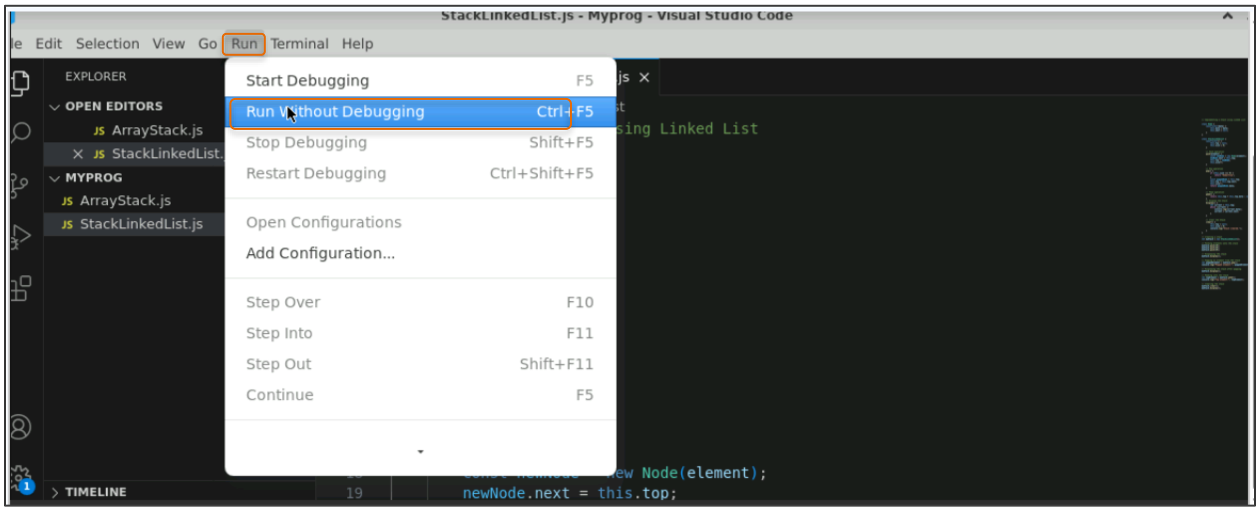

Selection View Go Run Terminal Help
EXPLORER
OPEN EDITORS 1 unsaved
ArrayStack.js
StackLinkedList.js
MYPROG
ArrayStack.js
StackLinkedList.js
TIMELINE
StackLinkedList.js > StackLinkedList > peek
Ln 38, Col 6 Spaces: 4 UTF-8 LF {} JavaScript

```
class StackLinkedList {
    }
}

// Creating a stack
let myStack = new StackLinkedList();

// Pushing elements onto the stack
myStack.push(10);
myStack.push(20);
myStack.push(30);

// Displaying the stack
myStack.display();

// Popping an element from the stack
let poppedElement = myStack.pop();
console.log("Popped element:", poppedElement);
```

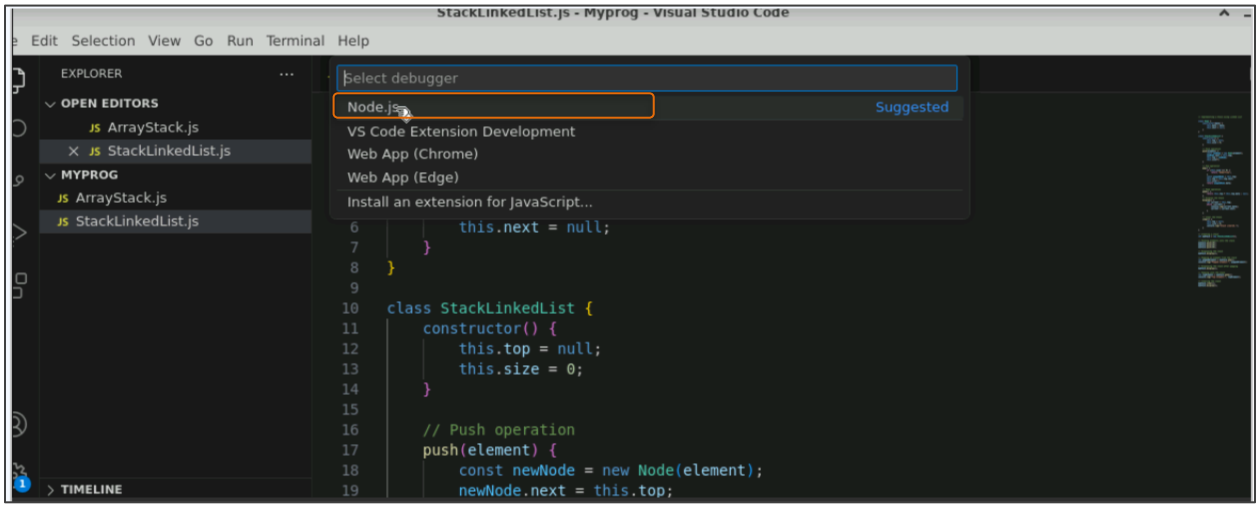

Selection View Go Run Terminal Help
EXPLORER
OPEN EDITORS 1 unsaved
ArrayStack.js
StackLinkedList.js
MYPROG
ArrayStack.js
StackLinkedList.js
TIMELINE
StackLinkedList.js > StackLinkedList > peek
Ln 38, Col 6 Spaces: 4 UTF-8 LF {} JavaScript

```
// Displaying the stack after popping
myStack.display();

// Peeking into the stack
let topElement = myStack.peek();
console.log("Top element:", topElement);

// Clearing the stack
myStack.clear();
myStack.display();
```

1.3 Click **Run** and then **Run Without Debugging**. Select **Node.js** to check the output in the DEBUG CONSOLE.

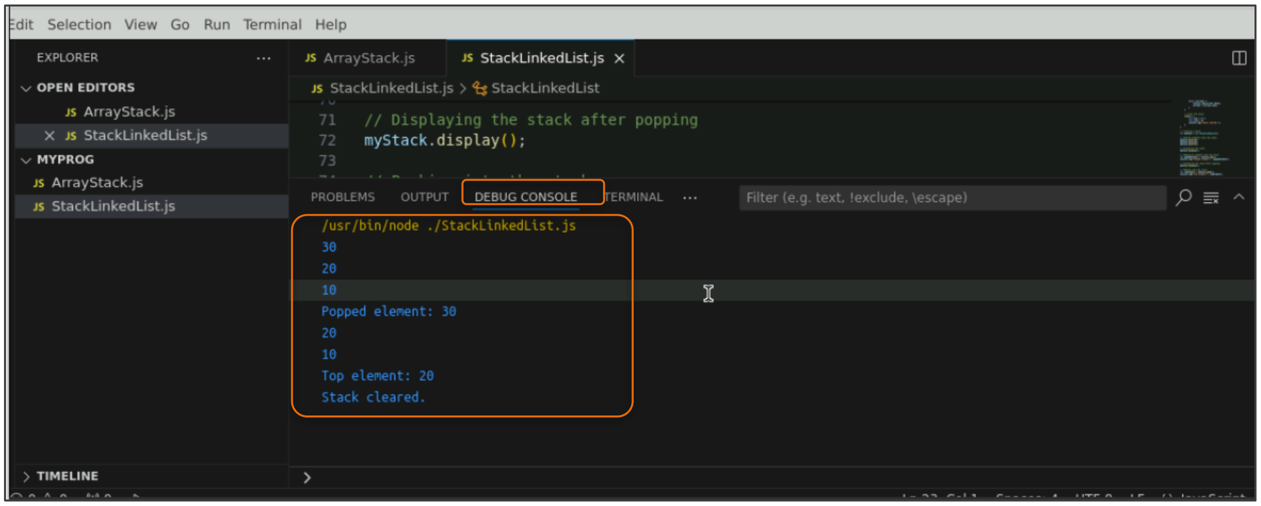


```
sing Linked List
        ew Node(element);
        newNode.next = this.top;
```



```
        this.next = null;
    }
}

class StackLinkedList {
    constructor() {
        this.top = null;
        this.size = 0;
    }

    // Push operation
    push(element) {
        const newNode = new Node(element);
        newNode.next = this.top;
```

1.4 View the output in the **DEBUG CONSOLE** as shown below:



```
// Displaying the stack after popping
myStack.display();
```

```
/usr/bin/node ./StackLinkedList.js
30
20
10
Popped element: 30
20
10
Top element: 20
Stack cleared.
```

Note: This example demonstrates the implementation of a stack using a linked list, including pushing, popping, peeking, displaying, and clearing elements.

By following these steps, you have successfully mastered the implementation and manipulation of a stack using a linked list in JavaScript, covering operations like push, pop, peek, display, and clear. This helps gain valuable insights into dynamic data structure operations to develop your dynamic data structure manipulation skills.