

## 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

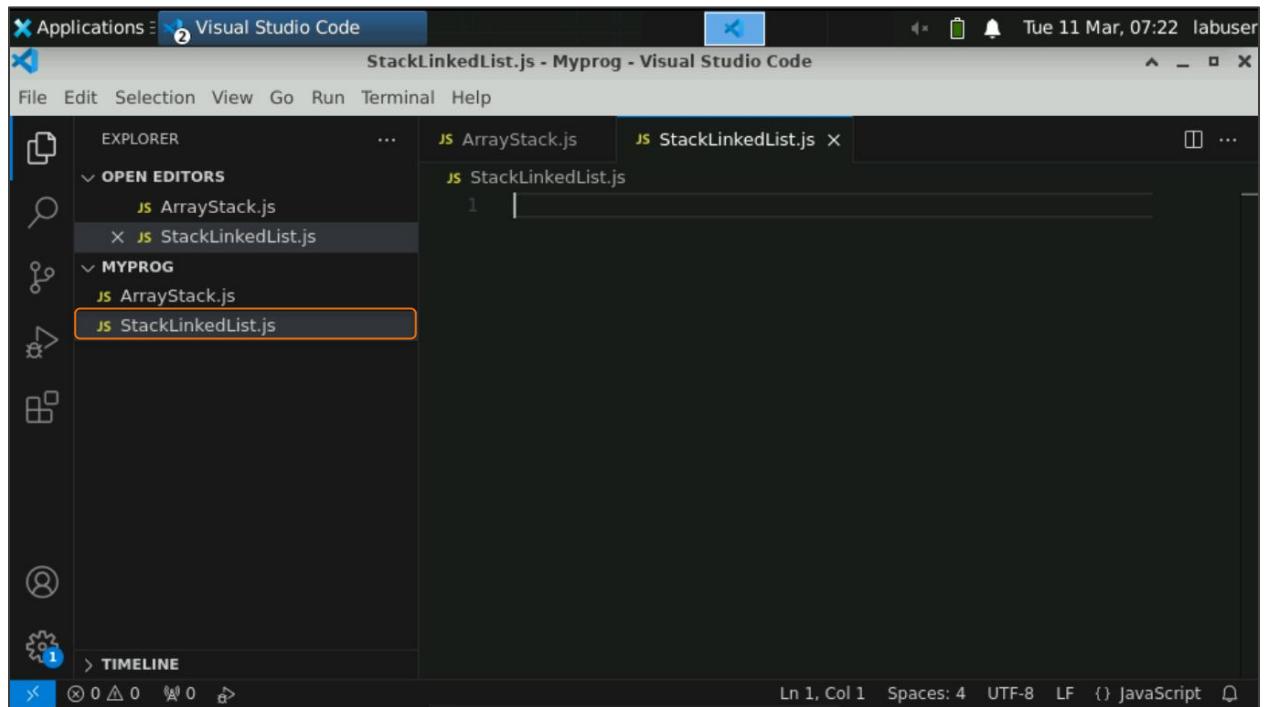
**Prerequisites:** A basic understanding of linked lists in JavaScript

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**



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();
```

A screenshot of a code editor interface. The top bar includes 'Selection', 'View', 'Go', 'Run', 'Terminal', and 'Help'. The left sidebar shows an 'EXPLORER' section with 'OPEN EDITORS' containing 'JS ArrayStack.js' and 'JS StackLinkedList.js'. Below this is a 'MYPROG' section with 'JS ArrayStack.js' and 'JS StackLinkedList.js'. A 'TIMELINE' section is at the bottom. The main area displays the 'JS StackLinkedList.js' 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() {
        if (this.size === 0) {
            return "Underflow";
        }
        return this.top.data;
    }
}
```

A screenshot of a code editor interface, identical to the one above but with more code visible in the 'JS StackLinkedList.js' 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() {
        if (this.size === 0) {
            return "Underflow";
        }
        return this.top.data;
    }
}
```

Selection View Go Run Terminal Help

KPLOKER ... JS ArrayStack.js JS StackLinkedList.js ●

PEN EDITORS 1 unsaved JS StackLinkedList.js > ⚙ StackLinkedList > ⚡ peek

JS StackLinkedList.js  
10 class StackLinkedList {  
35 // Peek operation  
36 peek() {  
37 return this.top ? this.top.data : null;  
38 }  
39 // Display the Stack  
40 display() {  
41 let current = this.top;  
42 while (current) {  
43 console.log(current.data);  
44 current = current.next;  
45 }  
46 }  
47 // Clear the Stack  
48 clear() {  
49 this.top = null;  
50 this.size = 0;  
51 console.log("Stack cleared.");  
52 }

IMELINE

Selection View Go Run Terminal Help

KPLOKER ... JS ArrayStack.js JS StackLinkedList.js ●

PEN EDITORS 1 unsaved JS StackLinkedList.js > ⚙ StackLinkedList > ⚡ peek

JS StackLinkedList.js  
10 class StackLinkedList {  
42 clear() {  
53 }  
54 }  
55 // Creating a stack  
56 let myStack = new StackLinkedList();  
57  
59 // Pushing elements onto the stack  
60 myStack.push(10);  
61 myStack.push(20);  
62 myStack.push(30);  
63  
64 // Displaying the stack  
65 myStack.display();  
66  
67 // Popping an element from the stack  
68 let poppedElement = myStack.pop();  
69 console.log("Popped element:", poppedElement);  
70 }

IMELINE

Selection View Go Run Terminal Help

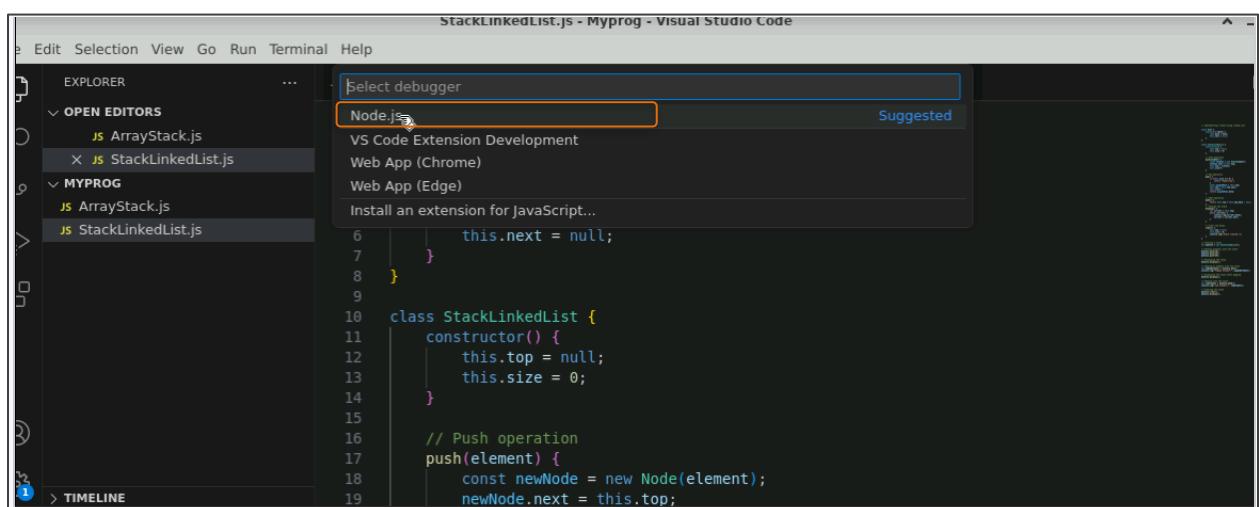
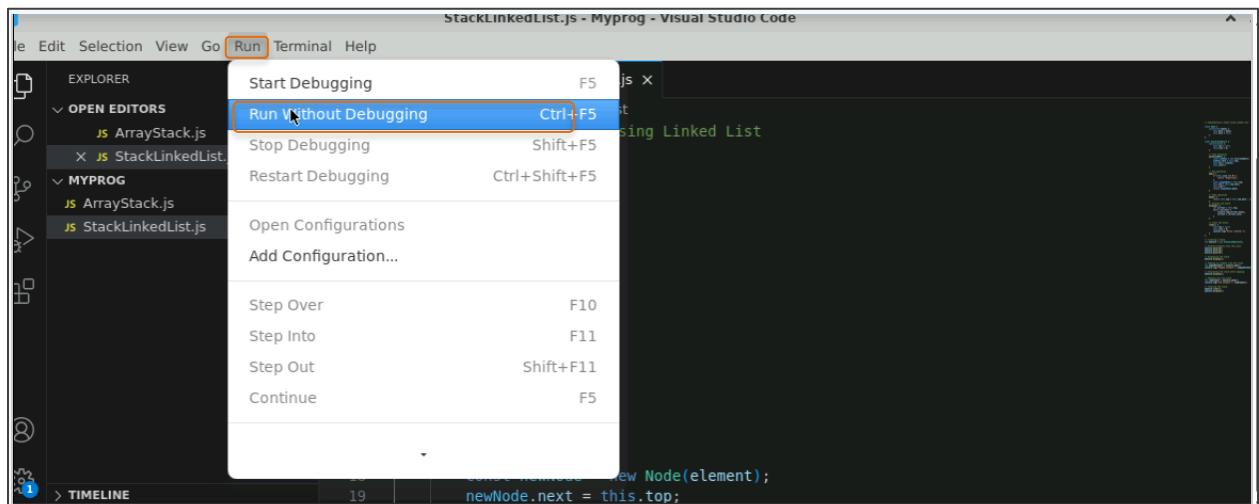
KPLOKER ... JS ArrayStack.js JS StackLinkedList.js ●

PEN EDITORS 1 unsaved JS StackLinkedList.js > ⚙ StackLinkedList > ⚡ peek

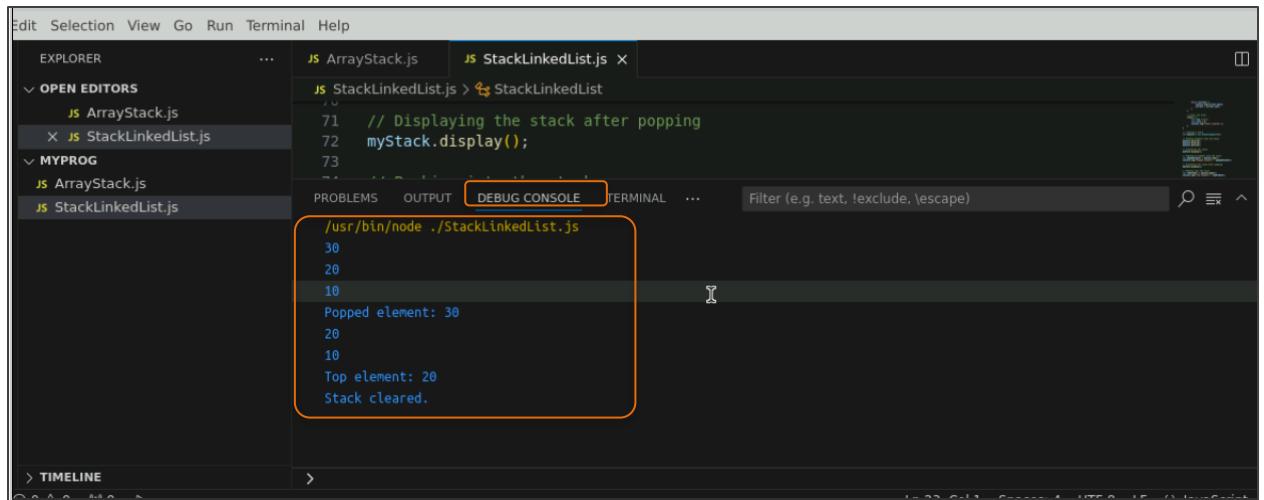
JS StackLinkedList.js  
71 // Displaying the stack after popping  
72 myStack.display();  
73  
74 // Peeking into the stack  
75 let topElement = myStack.peek();  
76 console.log("Top element:", topElement);  
77  
78 // Clearing the stack  
79 myStack.clear();  
80 myStack.display();  
81  
82  
83

IMELINE

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



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



The screenshot shows the VS Code interface with the DEBUG CONSOLE tab selected. The console output is highlighted with an orange box and contains the following text:

```
/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.