

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

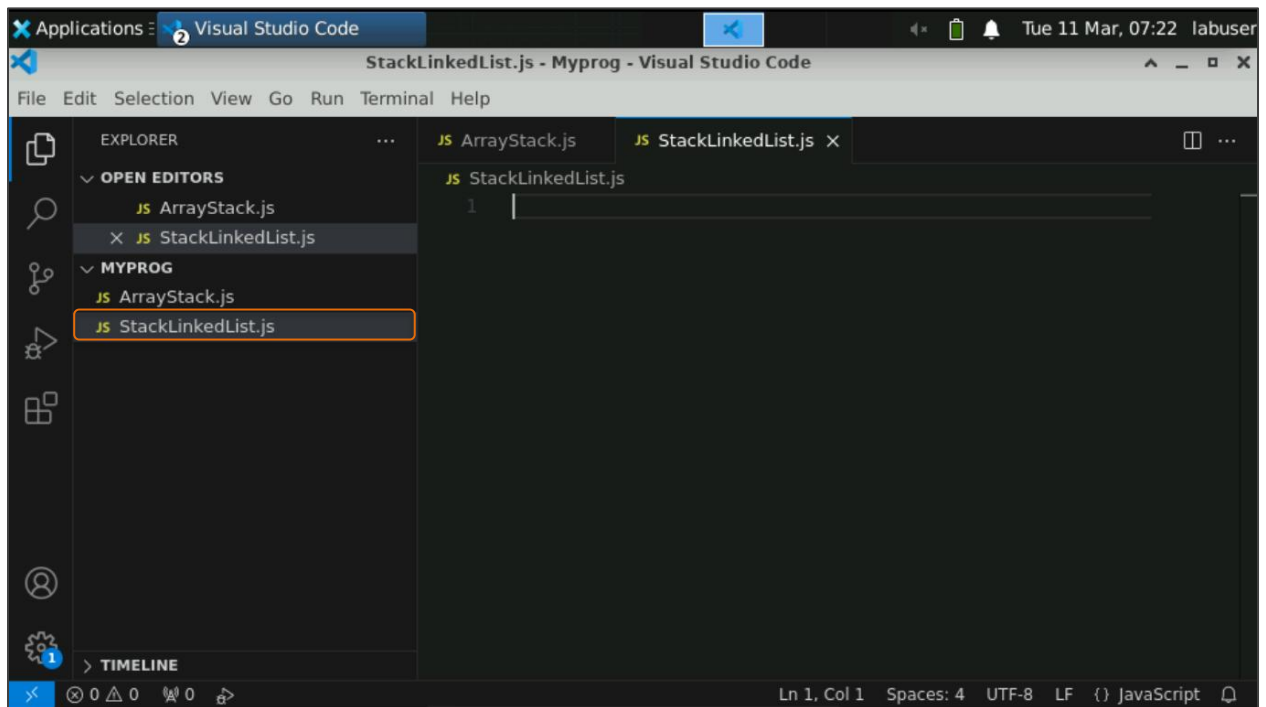
Prerequisites: A basic understanding of linked lists in JavaScript

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();
```

```
1 // Implementing a Stack using Linked List
2
3 class Node {
4   constructor(data) {
5     this.data = data;
6     this.next = null;
7   }
8 }
9
10 class StackLinkedList {
11   constructor() {
12     this.top = null;
13     this.size = 0;
14   }
15
16   // Push operation
17   push(element) {
18     const newNode = new Node(element);
19     newNode.next = this.top;
```

```
10 class StackLinkedList {
11   // Push operation
12   push(element) {
13     const newNode = new Node(element);
14     newNode.next = this.top;
15     this.top = newNode;
16     this.size++;
17   }
18
19   // Pop operation
20   pop() {
21     if (this.size === 0) {
22       return "Underflow";
23     }
24     const poppedNode = this.top;
25     this.top = this.top.next;
26     this.size--;
27     return poppedNode.data;
28   }
29 }
30
```

```
Selection View Go Run Terminal Help
XPLORER ... JS ArrayStack.js JS StackLinkedList.js
PEN EDITORS 1 unsaved
JS ArrayStack.js
JS StackLinkedList.js
YPROG
ArrayStack.js
StackLinkedList.js
IMELINE

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 }
```

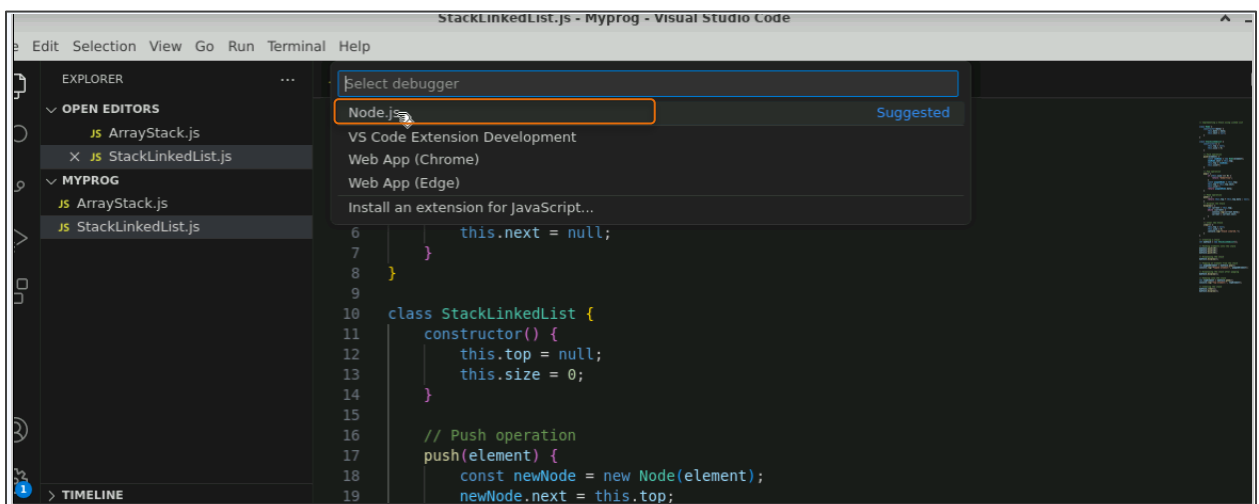
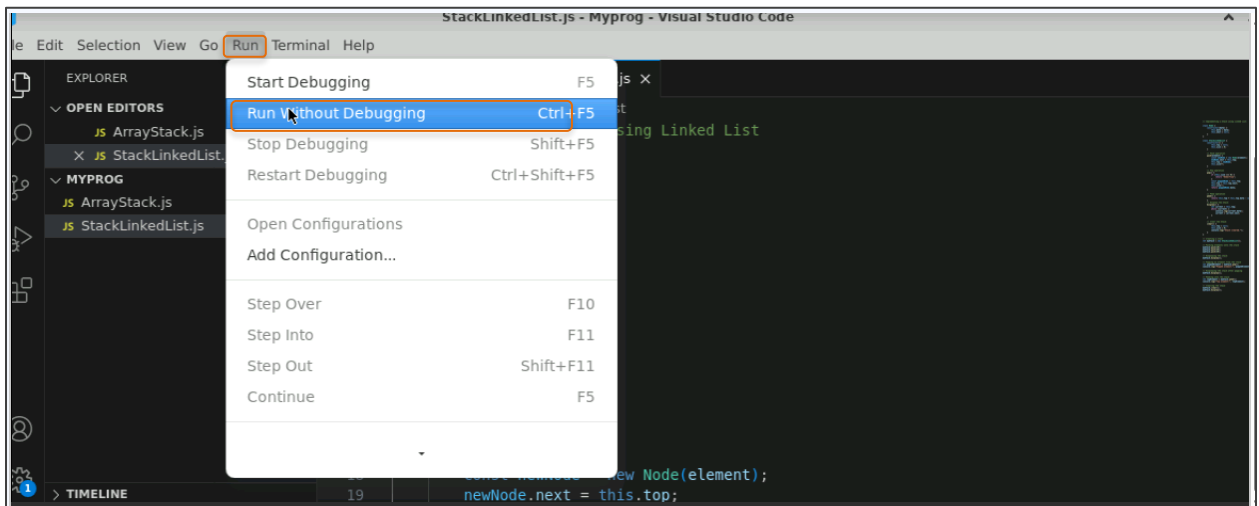
```
Selection View Go Run Terminal Help
XPLORER ... JS ArrayStack.js JS StackLinkedList.js
PEN EDITORS 1 unsaved
JS ArrayStack.js
JS StackLinkedList.js
YPROG
ArrayStack.js
StackLinkedList.js
IMELINE

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

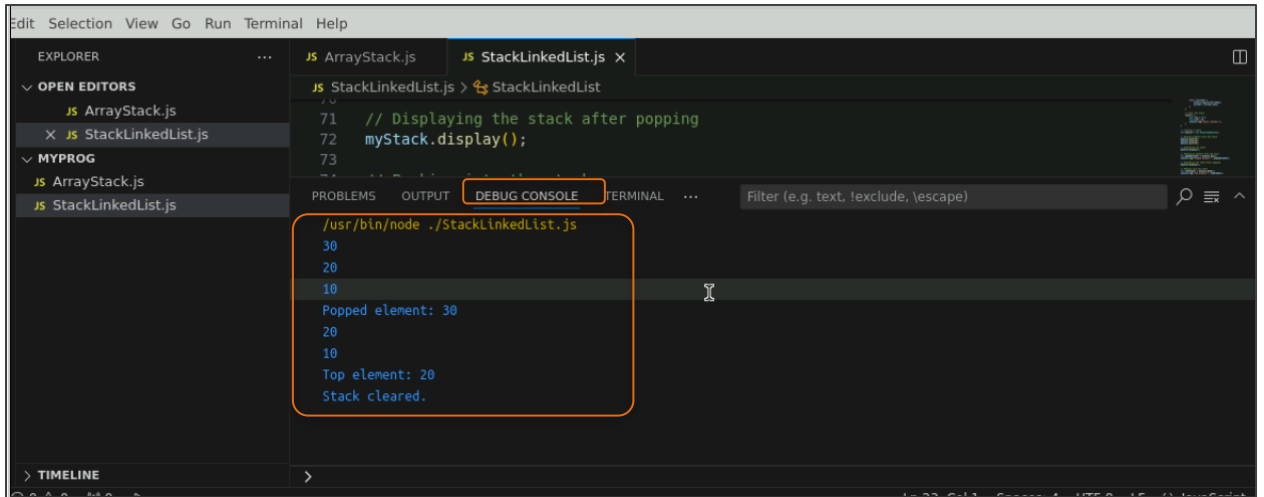
```
Selection View Go Run Terminal Help
XPLORER ... JS ArrayStack.js JS StackLinkedList.js
PEN EDITORS 1 unsaved
JS ArrayStack.js
JS StackLinkedList.js
YPROG
ArrayStack.js
StackLinkedList.js
IMELINE

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

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 Visual Studio Code interface. The Explorer panel on the left shows the file structure with 'StackLinkedList.js' selected. The Editor panel shows the code in 'StackLinkedList.js' with lines 71, 72, and 73 visible. The Debug Console panel at the bottom shows the output of the command `/usr/bin/node ./StackLinkedList.js`. The output is as follows:

```
/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.