**Data Structure and Algorithm Practicals**

5. Implement Stack using Linked List

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<script src="stacklist.js"></script>

<title>Document</title>

</head>

<body>

</body>

</html>

//Stack using linkedlist

function stackUsingLL(){

//Node

let Node = function(elm){

this.element = elm;

this.next = null;

}

//To keep track of the size

let length = 0;

//To keep track of the list

let head = null;

//Push data in the stack

this.push = function(elm){

//Create a new node

let node = new Node(elm),

current;

//Add the new node at the top

current = head;

node.next = current;

head = node;

length++;

}

//Pop the item from the stack

this.pop = function(){

let current = head;

//If there is item then remove it

//and make the next element as the first

if(current){

let elm = current.element;

current = current.next;

head = current;

length--;

return elm;

}

return null;

}

//Return the first element in the stack

this.peek = function(){

if(head){

return head.element;

}

return null;

}

//Convert the stack to an array

this.toArray = function(){

let arr = [];

let current = head;

while(current){

arr.push(current.element);

current = current.next;

}

return arr;

}

//Check if stack is empty

this.isEmpty = function(){

return length === 0;

}

//Return the size of the stack

this.size = function(){

return length;

}

//Clear the stack

this.clear = function(){

head = null;

length = 0;

}

}

let stack = new stackUsingLL(); //creating new instance of Stack

stack.push(1);

stack.push(2);

stack.push(3);

console.log(stack.peek());

console.log(stack.isEmpty());

console.log(stack.size());

console.log(stack.pop());

console.log(stack.toArray());

console.log(stack.size());

stack.clear(); //clear the stack

console.log(stack.isEmpty());