Implement a Stack using Queue

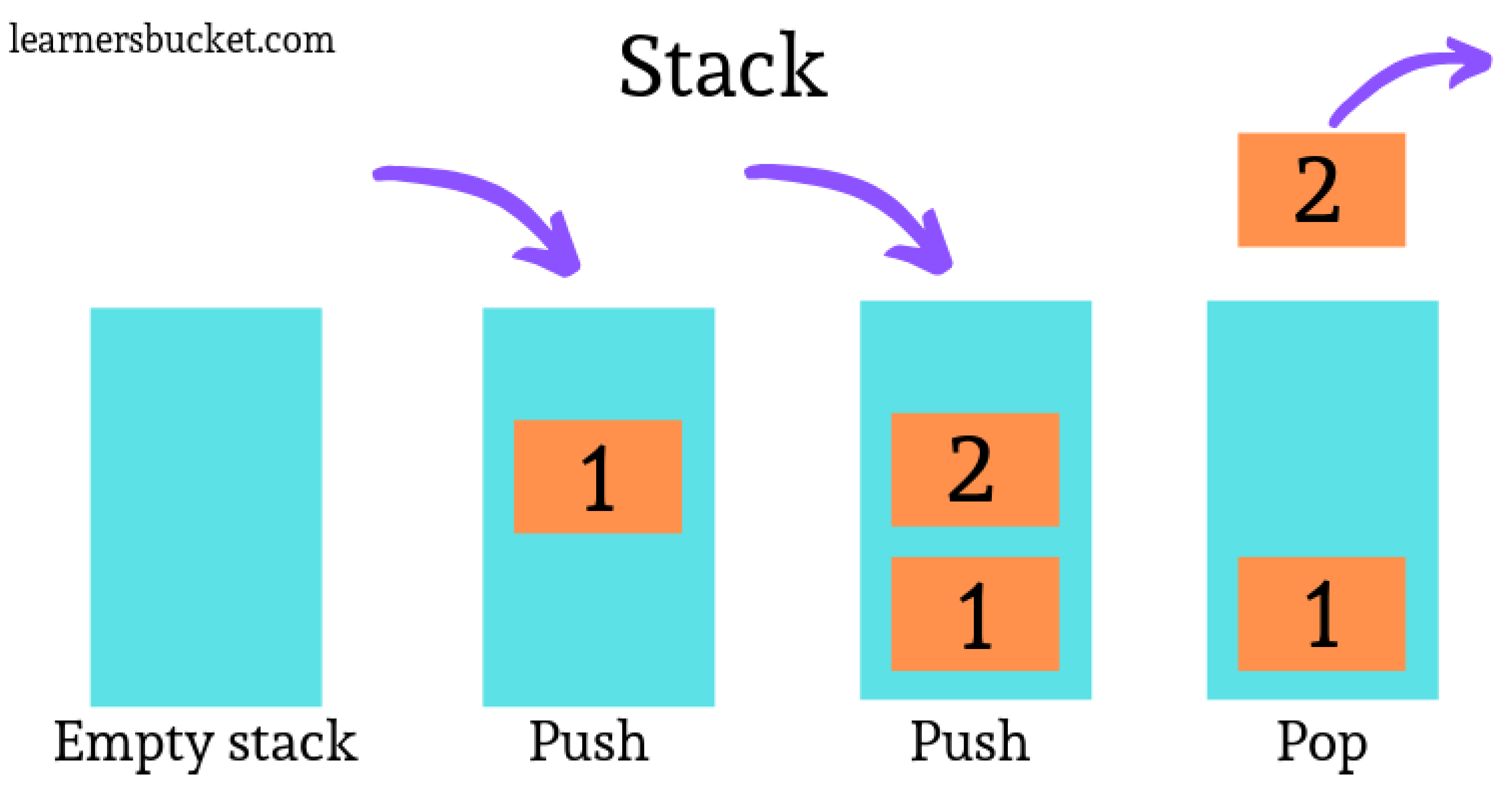
Posted on June 28, 2019 | by Prashant Yadav

Posted in Algorithms, Queue, Stack | Tagged Easy

Practically prepare for your JavaScript interview

JavaScript Revision
JavaScript-Concept Based Problems
Data Structures
Algorithms
Machine Coding
Web Fundamentals

Learn how to implement a stack using a single queue in javascript.



Implementation

- We will be using a single queue for creating the stack.
- So every time we will add a new data to the queue, we will move the existing data to the back of the new data.
- This way we will be able to mimic the stack implementation using the queue operations.

Implementing stack using a single queue

```
function Stack(){
   let queue = new Queue();

   //Other methods go here

}
```

Pushing data in the stack

We will enqueue the data in the queue and move the existing data to the back of the new data.

Copy

```
//Push
  this.push = function(elm){
    let size = queue.size();

    queue.enqueue(elm);

    //Move old data after the new data
    for(let i = 0; i < size; i++){
      let x = queue.dequeue();
      queue.enqueue(x);
    }
  }
```

Pop the data from the stack

Copy

```
//Pop
  this.pop = function(){
    if(queue.isEmpty()){
      return null;
    }

    return queue.dequeue();
  }
```

Peek the data in the stack

Copy

```
//Peek
  this.peek = function(){
    if(queue.isEmpty()){
      return null;
    }

    return queue.front();
  }
```

Size of the stack

Copy

```
//Size
  this.size = function(){
    return queue.size();
  }
```

Check if stack is empty

Copy

```
//IsEmpty
  this.isEmpty = function(){
    return queue.isEmpty();
  }
```

Clear the stack

Copy

```
//Clear
  this.clear = function(){
    queue.clear();
    return true;
  }
```

Convert the stack to an array

Copy

```
//ToArray
  this.toArray = function(){
    return queue.toArray();
  }
```

Complete Code

Copy

```
function Stack() {
  let queue = new Queue();

  //Push
  this.push = function(elm){
    let size = queue.size();

    queue.enqueue(elm);

    //Move old data after the new data
    for(let i = 0; i < size; i++){
      let x = queue.dequeue();
      queue.enqueue(x);
    }
  }

  //Pop
  this.pop = function(){
    if(queue.isEmpty()){
      return null;
    }

    return queue.dequeue();
  }

  //Peek
  this.peek = function(){
    if(queue.isEmpty()){
      return null;
    }

    return queue.front();
  }

  //Size
  this.size = function(){
    return queue.size();
  }

  //IsEmpty
  this.isEmpty = function(){
    return queue.isEmpty();
  }

  //Clear
  this.clear = function(){
    queue.clear();
    return true;
  }

  //ToArray
  this.toArray = function(){
    return queue.toArray();
  }
}
```

```
Input:
let stack = new Stack();   //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());

Output:
3
false
3
3
[2, 1]
2
true
```

We can wrap this inside a closure and IIFE (Immediately Invoked Function Expression) to make all the properties and methods private.

Copy

```
let Stack = (function(){

return function Stack() {
    let queue = new Queue();

    //Push
    this.push = function(elm){
      let size = queue.size();

      queue.enqueue(elm);

      //Move old data after the new data
      for(let i = 0; i < size; i++){
        let x = queue.dequeue();
        queue.enqueue(x);
      }
    }

    //Pop
    this.pop = function(){
      if(queue.isEmpty()){
        return null;
      }

      return queue.dequeue();
    }

    //Peek
    this.peek = function(){
      if(queue.isEmpty()){
        return null;
      }

      return queue.front();
    }

    //Size
    this.size = function(){
      return queue.size();
    }

    //IsEmpty
    this.isEmpty = function(){
      return queue.isEmpty();
    }

    //Clear
    this.clear = function(){
      queue.clear();
      return true;
    }

    //ToArray
    this.toArray = function(){
      return queue.toArray();
    }
  }

}());
```

Time Complexity

#	Access	Search	Insert	Delete
Average	Θ(N)	Θ(N)	Θ(N)	Θ(1)
Worst	O(N)	O(N)	O(N)	O(1)

Because we are copying the data at the end of the queue after adding a new data, the insert operation changes to O(N).

Space Complexity

#	space
Worst	O(N)

Recommended Posts:

- Reverse a stack using recursion.
- FizzBuzz program in javascript
- Learn how to reverse a linked list
- Sort a stack using another stack
- Check if given binary tree is full.
- Program to check if two stacks are equal
- Sorting a linked list
- 3 sum problem algorithm
- Quick sort Iterative
- Merge overlapping intervals

Prev Next