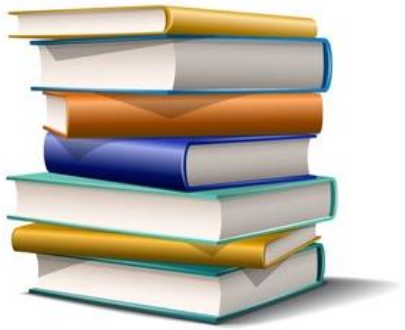
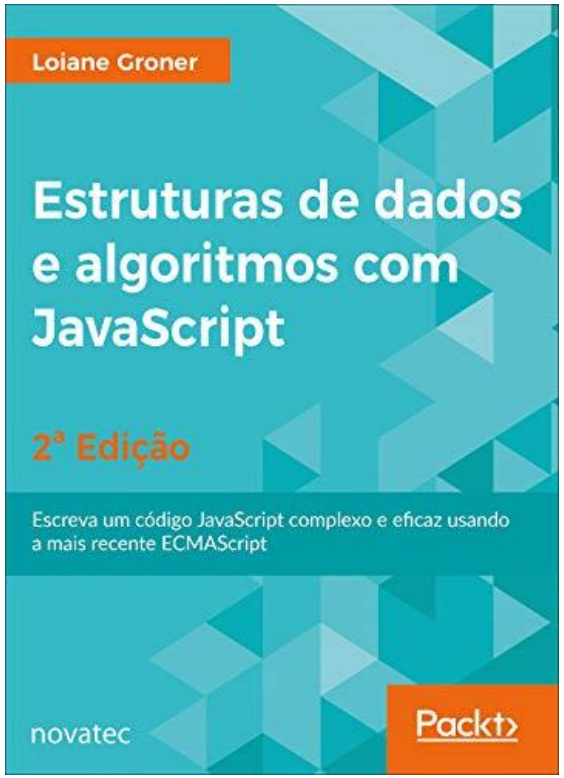
PILHAS

Referências Bibliográficas:

GRONER, Loiane. **Estruturas de dados e algoritmos com JavaScript**: Escreva um código JavaScript complexo e eficaz usando a mais recente ECMAScript. 2ª ed. São Paulo: Novatec, 2019.



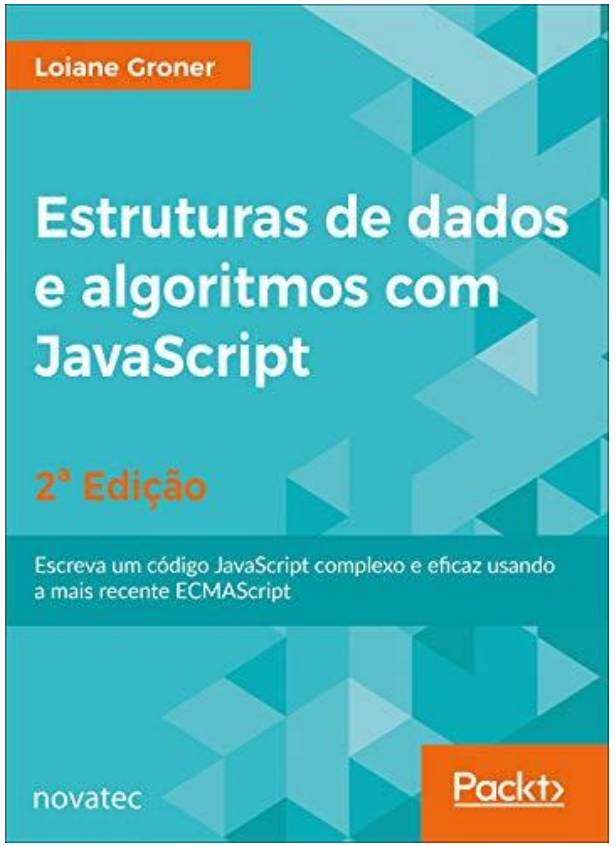


SIF005 - Estrutura de Dados

Prof. Dr. Anderson – anderson.sena@iesb.edu.br

Os seguintes tópicos serão abordados:

- ✓ Criação da nossa própria biblioteca de estrutura de dados JavaScript;
- ✓ A estrutura de dados pilha;
- ✓ Adição de elementos em uma pilha;
- ✓ Remoção (pop) de elementos de uma pilha;
- ✓ Como usar a classe Stack;
- ✓ O problema do decimal para binário.



PILHAS

- Coleção ordenada de itens que segue o princípio LIFO (Last In First Out), ou seja, o último a entrar é o primeiro a sair da pilha.

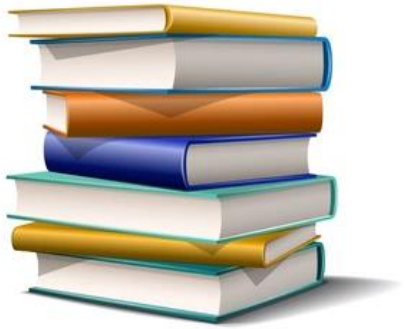
- Temos vários exemplos da vida real que pode nos ajudar a compreender este princípio, um deles é a pilha de livros.

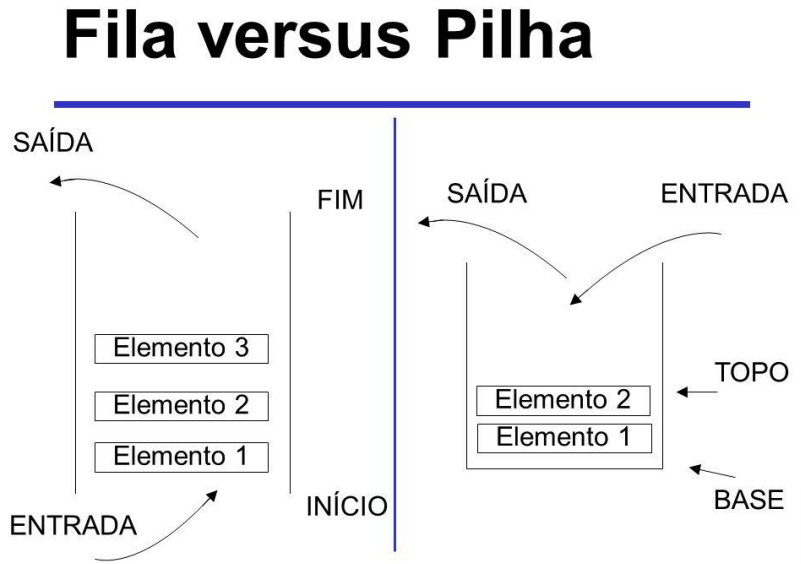
Relembrando conceito da Aula 01

- **Fila e pilha** são estruturas abstratas lineares. A escolha entre as duas está relacionada à ordem de entrada e saída dos elementos:

  ✓ **Fila** é para qualquer situação FIFO - first in, first out (primeiro que entra é o primeiro que sai).
  exemplos: playlist de músicas, pedidos de uma loja, documentos para impressão.

  ✓ **Pilha** é para situações de LIFO - last in, first out (último que entra é o primeiro que sai).
  exemplo: feed de notícias, função de fazer e refazer nos editores de texto.

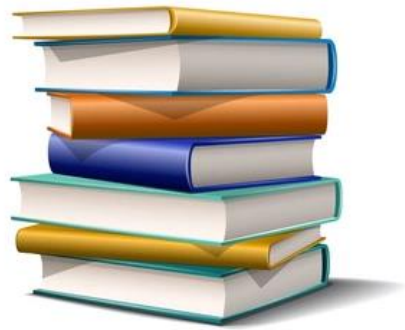




Fonte: Repositório GitHub Fred Gomes <https://github.com/freddgomes/Estruturas-de-Dados-e-Algoritmos>, acesso em 24/01/2021

Pilhas

- Na aula de anterior, vimos como criar e usar **arrays** que são o tipo mais comum de estruturas de dados em ciência da computação.
- Aprendemos que é possível adicionar e remover elementos de um **array** em qualquer índice, porém, às vezes precisamos de alguma forma de estrutura com mais controle de acréscimo e remoção de itens.
- Há duas estruturas de dados que apresentam semelhanças com os **arrays**, mas com mais controle de **adição** e **remoção** dos elementos.
- São as **pilhas** (stacks) e as **filas** (queues).



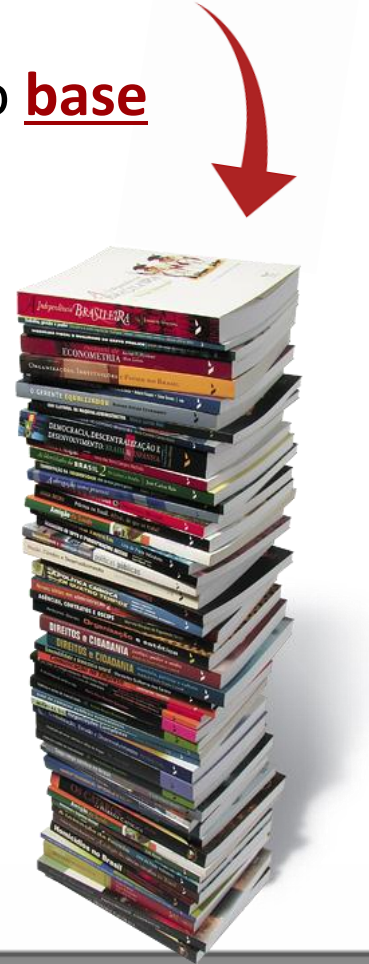
Groner, 2019, pág. 108

Pilhas

- Uma pilha é uma coleção ordenada de itens que obedece ao princípio **LIFO** (Last In First Out, isto é, o último a entrar é o primeiro a sair).
- A adição de novos itens ou a remoção de itens existentes ocorrem **na mesma extremidade**
- O final da P**ILHA** é chamado de **topo**, enquanto o lado oposto é conhecido como **base**
- Os elementos mais novos ficam próximos ao topo, e os elementos mais antigos estão próximos da base.

Exemplos:

  - pilha de livros
  - pilha de pratos em um restaurante,
  - Pilha de janelas abertas no Ms Windows



Criando uma classe Stack baseada em array

- Criaremos a nossa própria classe para representar uma pilha.
- Vamos começar pelo básico criando um arquivo **stack-array.js** e declarando a nossa classe **Stack**:

```
1   class Stack {
2       constructor( ){
3           this.items = [];
4       }
5   }
6
```

- Podemos usar um **array** que armazenará os elementos da pilha.
- Como a **pilha** obedece ao princípio **LIFO**, limitaremos as funcionalidades que estarão disponíveis à inserção e remoção de elementos.

Criando uma classe Stack baseada em array

Os métodos a seguir estarão disponíveis na classe **Stack**:

- **push(**elemento(s)**)**: esse método adiciona um novo elemento (ou vários elementos) no topo da pilha.
- **pop( )** : esse método remove o elemento que está no topo da pilha. Também devolve o elemento removido.
- **peek( )**: esse método devolve o elemento que está no topo da pilha. A pilha não é modificada (o elemento não é removido; ele é devolvido apenas como informação).
- **isEmpty( )**: esse método devolve **true** se a pilha não contiver nenhum elemento e **false** se o tamanho da pilha for maior que 0.
- **clear( )**: esse método remove todos os elementos da pilha.
- **size( )**: esse método devolve o número de elementos contidos na pilha. É semelhante à propriedade lenght de um array.

Métodos a serem implementados na classe Stack


```
1   class Stack {
2       constructor( ){
3           this.items = [];
4       }
5       push(element){
6           //adiciona um novo item à pilha
7       }
8       pop(){
9           // remover o item do topo da pilha
10      }
11      peek() {
12          // devolve o elemento que está no topo da pilha
13          }
14      isEmpty() {
15          // informar se a pilha está vazia ou não
16          }
17      clear(){
18          // limpa a pilha
19      }
20      size() {
21          // informar o tamanho da pilha
22      }
23      print(){
24          // imprime a pilha no console
25      }
26  }
27
```

Push de elementos na pilha

- O primeiro método que implementaremos é o método push( ), responsável pela adição de novos elementos na pilha.

- IMPORTANTE: só podemos adicionar novos itens no topo da pilha (no final).

```
push(element){
    //adiciona um novo item à pilha
    this.items.push(element);
}
```

- Como estamos usando um **array** para armazenar os elementos da pilha, podemos utilizar o método push da classe Array de JavaScript, que conhecemos anteriormente.

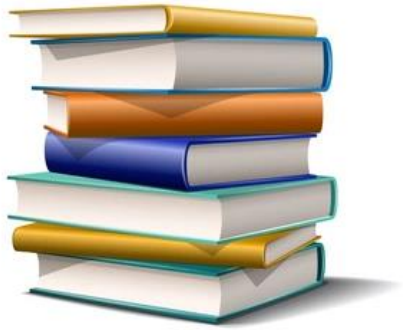
Pop de elementos da pilha

- A seguir, implementaremos o método **pop( )**, responsável pela remoção de itens da pilha.

- Como a pilha utiliza o princípio **LIFO** (*Last in First out*), o último item adicionado é aquele que será removido.

- Por esse motivo, podemos usar o método **pop( )** da classe **Array** de **JavaScript**.

```
pop(){
    // remover o item do topo da pilha
    return this.pop();
}
```



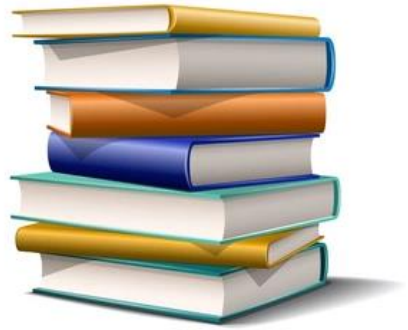
Elemento no topo da pilha

Dando uma espiada no elemento que está no topo da pilha

- Vamos agora implementar métodos auxiliares em nossa classe.
- Se quisermos saber qual foi o último elemento adicionado em nossa pilha, podemos usar o método **peek**.

```
peek() {
    // devolve o elemento que está no topo da pilha
    return this.items[this.items.length - 1];
}
```

- Esse método devolverá o item que está no **topo da pilha**.



Elemento no topo da pilha

Dando uma espiada no elemento que está no topo da pilha

- Vamos agora implementar métodos auxiliares em nossa classe.
- Se quisermos saber qual foi o último elemento adicionado em nossa pilha, podemos usar o método **peek**.

```
peek() {
    // devolve o elemento que está no topo da pilha
    return this.items[this.items.length - 1];
}
```

- Esse método devolverá o item que está no **topo da pilha**.

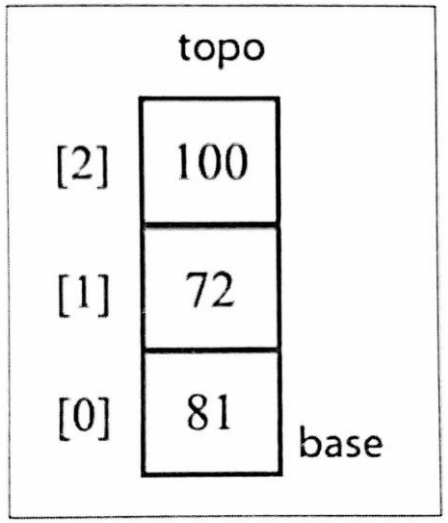


Figura 4.4

Como estamos usando internamente um array, o último item é **lenght - 1**

Groner, 2019, pág. 113

Verificando se a pilha está vazia

- O próximo método que criaremos é isEmpty, que devolverá true se a pilha estiver vazia (nenhum elemento foi adicionado), e false caso contrário.

```
isEmpty() {
    // informar se a pilha está vazia ou não
    return this.items.length === 0;
}
```

- Ao usar o método **isEmpty**, podemos simplesmente verificar se o tamanho do **array** interno é **0**.
- De modo semelhante à propriedade lenght da classe **Array**, também podemos implementar **lenght** em nossa classe **Stack**.

Verificando o tamanho da pilha

- Para coleções, em geral usamos o termo **size** no lugar de **lenght**.

- Novamente, como estamos usando um **array** para armazenar os elementos internamente, basta devolver o valor de **lenght**.

```
size() {
    // informar o tamanho da pilha
    return this.items.length;
}
```

Groner, 2019, pág. 113

Limpando os elementos da pilha

- Implementaremos o método clear, o qual simplesmente esvazia a pilha, removendo todos os seus elementos, sobrescrevendo por um **array** vazio, de forma bem simples.

```
clear(){
    // limpa a pilha
    this.items = [];
}
```

- Por fim, incluiremos um método para imprimir a pilha na console

```
print(){
    // imprime a pilha no console
    console.log(items.toString());
}
```

Groner, 2019, pág. 113

Usando a classe Stack

- Nossa primeira tarefa deve ser instanciar a classe **Stack**.
- Em seguida, podemos verificar se ela está vazia.

```
40  // criando (instancia) um objeto stack (pilha)
41  const stack = new Stack();
42  //verificando se a pilha stack está vazia
43  console.log(stack.isEmpty()); // exibe true
44
```

PROBLEMS 2 OUTPUT TERMINAL DEBUG CONSOLE

```
[Running] node "c:\Users\admin\Desktop\ed-js\js\tempCo
true
```

- A seguir, vamos adicionar alguns elementos e exibir com o método **peek( )** o elemento do topo da pilha:

```
45  // adicionando elementos no topo da pilha
46  stack.push(5);
47  stack.push(8);
48  // exibindo o elemento do topo da pilha
49  console.log(stack.peek()); // exibe 8
50
```

PROBLEMS 2 OUTPUT TERMINAL DEBUG CONSOLE

```
[Running] node "c:\Users\admin\Desktop\ed-js\js\t
8
```

Usando a classe Stack

- ✓ Vamos adicionar outro elemento.
- ✓ Com o método **size( )**, o resultado será o número **3**.
- ✓ Se chamarmos o método **isEmpty( )**, a saída será **false**.
- ✓ Por fim, vamos acrescentar outro elemento.
- ✓ Vamos também apresentar todos os elementos adicionados à pilha.

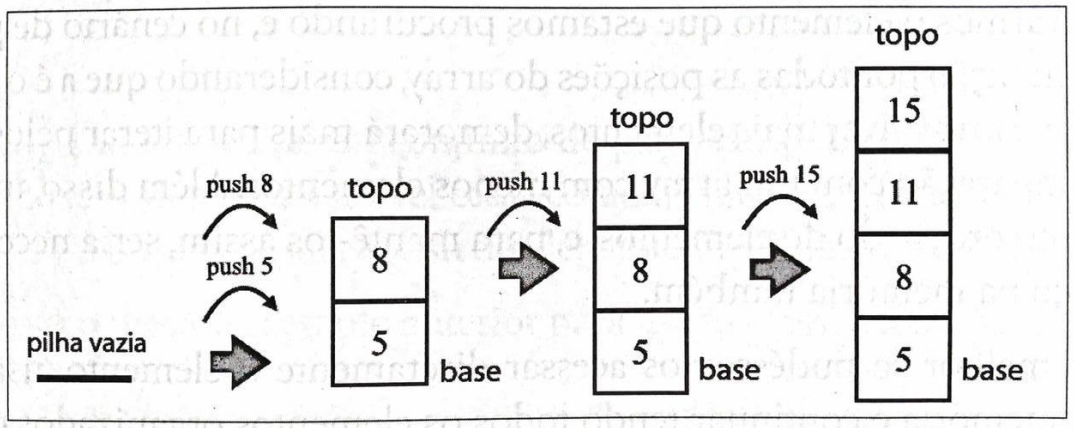


Figura 4.5

```
51  //vamos adicionar outro elemento na pilha
52  stack.push(11);
53  //exibindo o tamanho da pilha
54  console.log(stack.size()); // exibe 3
55  //verificando se a pilha está vazia
56  console.log(stack.isEmpty()); // exibe false
57  // por fim, vamos acrescentar outro elemento
58  stack.push(15);
59  // mostrando todos os elementos da pilha
60  stack.print();
61
```

PROBLEMS 2 OUTPUT TERMINAL DEBUG CONSOLE

```
[Running] node "c:\Users\admin\Desktop\ed-js\js\temp
3
false
5,8,11,15
```

Usando a classe Stack

- ✓ Em seguida, vamos remover dois elementos da pilha chamando o método **pop( )** duas vezes.

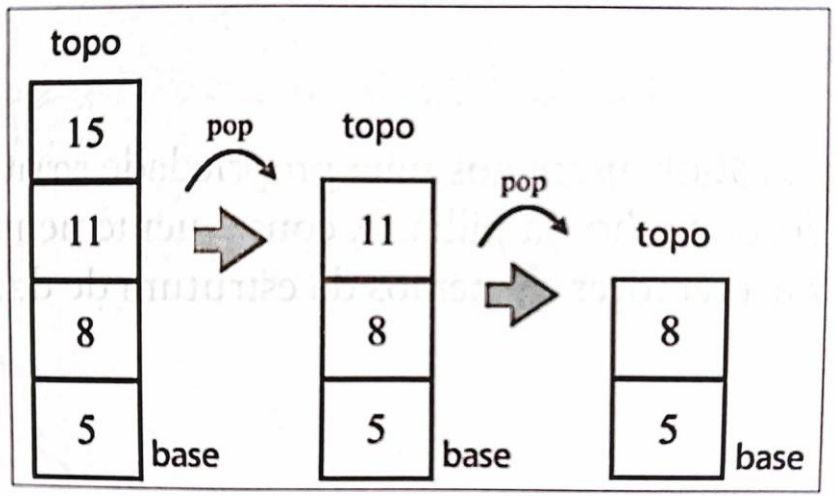


Figura 4.6

```
59    // mostrando todos os elementos da pilha
60    stack.print();
61    // retirando dois elementos do topo da pilha
62    stack.pop();
63    stack.pop();
64    stack.print();
65
```

PROBLEMS 2 OUTPUT TERMINAL DEBUG CONSOLE

```
[Running] node "c:\Users\admin\Desktop\ed-js\js\temp
5,8,11,15
5,8
```

- ✓ Empilhando e desempilhando elementos:

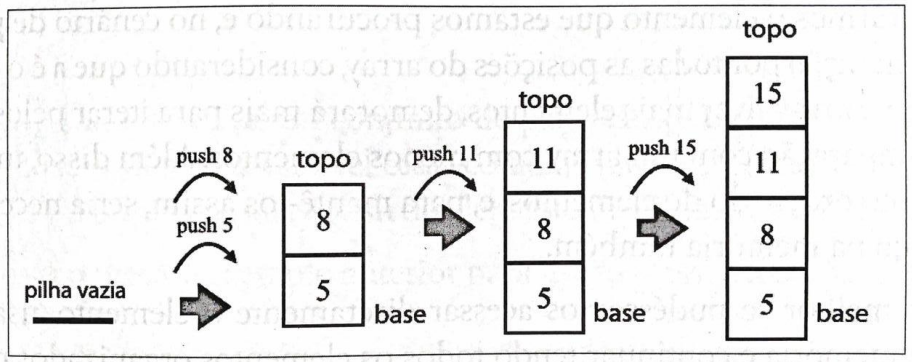


Figura 4.5

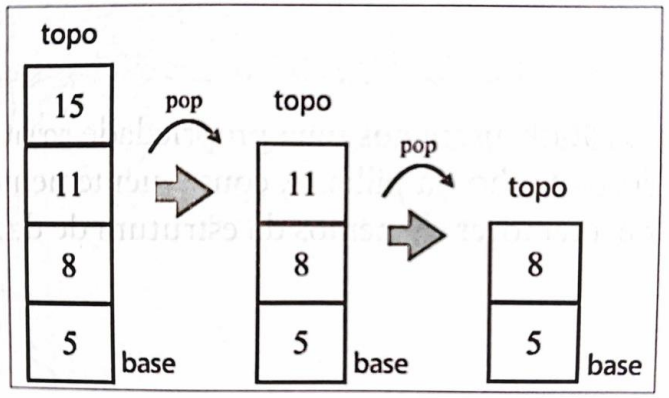


Figura 4.6

```
41  var pilha = new Stack();
42  console.log(pilha.isEmpty());
43  pilha.push(5);
44  pilha.print();
45  pilha.push(8);
46  pilha.print();
47  pilha.push(11);
48  pilha.print();
49  pilha.push(15);
50  pilha.print();
51  pilha.pop();
52  pilha.print();
53  pilha.pop();
54  pilha.print();
55  pilha.pop();
56  pilha.print();
57  pilha.pop();
58  pilha.print();
59  console.log(pilha.isEmpty());
```

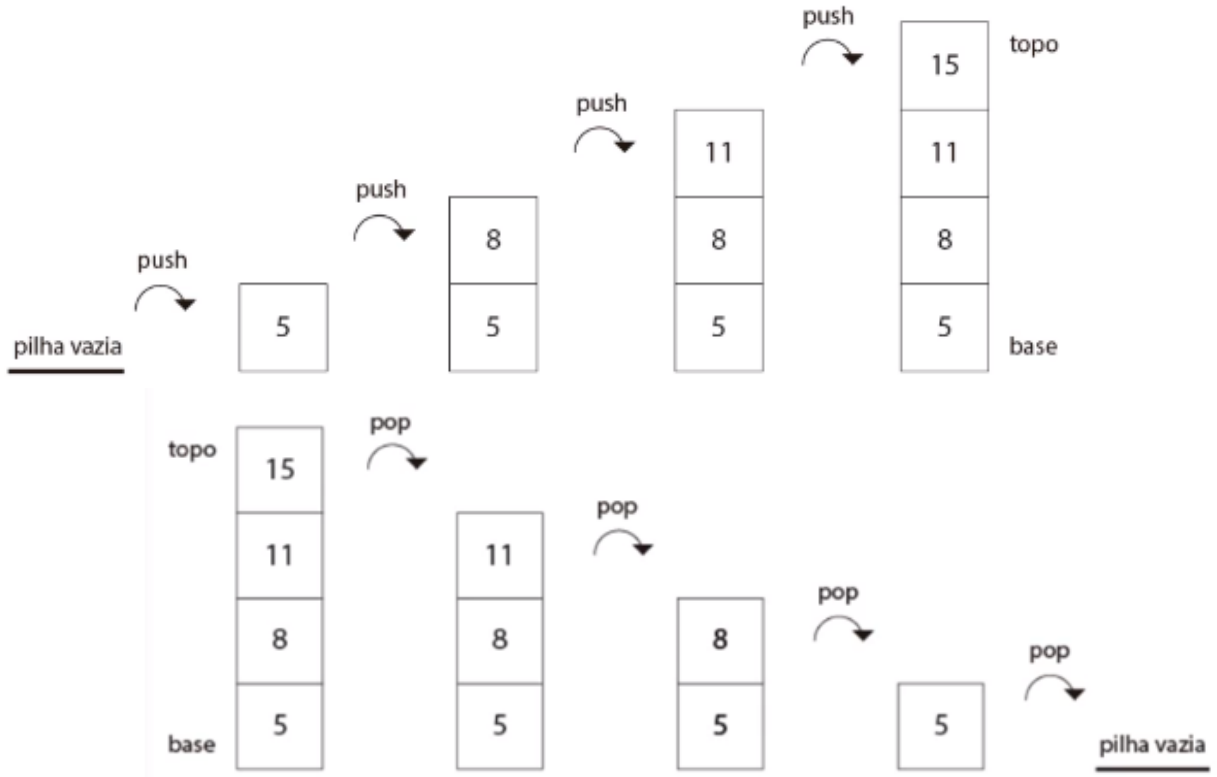
PROBLEMS 2 OUTPUT TERMINAL DEBUG CONSO

```
[Running] node "c:\Users\admin\Desktop\e
true
5
5,8
5,8,11
5,8,11,15
5,8,11
5,8
5

true
```

PILHAS

ANÁLISE GRÁFICA




```
41  var pilha = new Stack();
42  console.log(pilha.isEmpty());
43  pilha.push(5);
44  pilha.print();
45  pilha.push(8);
46  pilha.print();
47  pilha.push(11);
48  pilha.print();
49  pilha.push(15);
50  pilha.print();
51  pilha.pop();
52  pilha.print();
53  pilha.pop();
54  pilha.print();
55  pilha.pop();
56  pilha.print();
57  pilha.pop();
58  pilha.print();
59  console.log(pilha.isEmpty());
```

PROBLEMS 2 | OUTPUT | TERMINAL | DEBUG CONSO

```
[Running] node "c:\Users\admin\Desktop\e
true
5
5,8
5,8,11
5,8,11,15
5,8,11
5,8
5

true
```

Resolvendo problemas usando pilhas

Convertendo números decimais para binários

- Para converter um número decimal e uma representação binária, podemos dividir o número por 2 (binário é um sistema numérico de base 2) até que o resultado divisão seja 0.
- Na **figura 4.7**, converteremos o número 10 em dígitos binários:

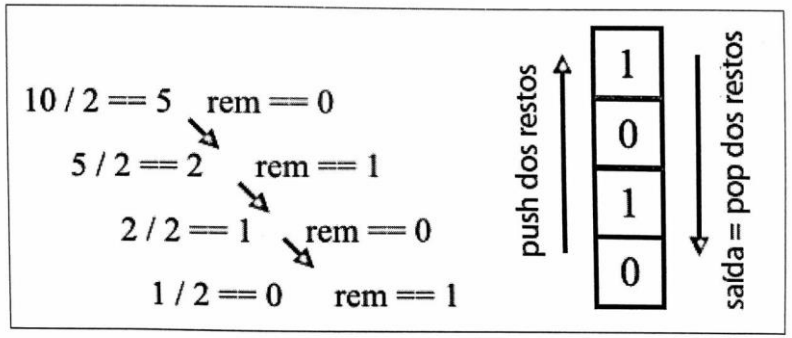


Figura 4.7

Recomendação de site de conversões binárias para conferir:
http://conversordemedidas.info/sistema-binario.php

```
40  function decimalToBinary(decNumber){
41      var restStack = new Stack(),
42      rest,
43      binaryString = '';
44
45      while(decNumber > 0) {
46          //arredonda pra baixo e atribui o resto da divisão por 2
47          rest = Math.floor(decNumber % 2);
48          //acrescenta na pilha
49          restStack.push(rest);
50          //arredonda pra baixo e atribui o resultado da divisão por 2
51          decNumber = Math.floor(decNumber / 2);
52      }
53      while(!restStack.isEmpty()) {
54          //retira o último da pilha e acrescenta à binaryString no formato
55          binaryString += restStack.pop().toString();
56      }
57      return binaryString;
58  }
59  console.log(decimalToBinary(10));
60  console.log(decimalToBinary(25));
61  console.log(decimalToBinary(233));
62  console.log(decimalToBinary(1000));
```

```
PROBLEMS 2   OUTPUT   TERMINAL   DEBUG CONSOLE
[Running] node "c:\Users\admin\Desktop\ed-js\js\tempCodeRunnerFile.js"
1010
11001
11101001
1111101000
```