Estruturas de Dados 1

Módulo 2 - Pilhas

Prof. Dr. Daniel A. Furtado - FACOM/UFU

Conteúdo protegido por direito autoral, nos termos da Lei nº 9 610/98
A cópia, reprodução ou apropriação deste material, total ou parcialmente, é proibida pelo autor

Conteúdo da Aula

1. Tipo Abstrato de Dado (TAD)
2. TAD Pilha - Conceito e operações
3. TAD Pilha - Implementação em *array*

Proibida cópia, apropriação ou uso sem autorização de qualquer parte deste material - Lei nº 9 610/98

Estrutura de Dados

- Dados organizados de maneira estruturada, com algum relacionamento lógico entre si
- Exemplo: *struct* na linguagem C:

```
struct Pessoa {
    char nome[50];
    int idade;
    float altura;
}
```

Proibida cópia, apropriação ou uso sem autorização de qualquer parte deste material - Lei nº 9 610/98

Tipo Abstrato de Dados (TAD)

- Estrutura de dados que inclui operações para manipulação dos dados, como:
  - Operação para criar a estrutura de dados
  - Operação para inserir/remover elementos
  - Operação para acessar os elementos
- Exemplos: Pilhas e Filas.

 Proibida cópia, apropriação ou uso sem autorização de qualquer parte deste material - Lei nº 9 610/98

TAD Pilha - Conceito

- Análogo a uma pilha de pratos em um restaurante
- Dados podem ser adicionados e removidos de uma única extremidade, denominada **topo** da pilha
- O último dado inserido na pilha será o primeiro a ser removido
- **L**ast **I**n **F**irst **O**ut (LIFO) - último a entrar, primeiro a sair

Proibida cópia, apropriação ou uso sem autorização de qualquer parte deste material - Lei nº 9 610/98

TAD Pilha - Exemplos de Aplicações

- Histórico de páginas armazenadas pelo navegador de internet
- Operação desfazer (ctrl-z) em softwares de edição
- Avaliação de operações aritméticas

Proibida cópia, apropriação ou uso sem autorização de qualquer parte deste material - Lei nº 9 610/98

TAD Pilha - Operações Fundamentais

- **Push**: adiciona um elemento no topo da pilha
- **Pop**: remove e retorna o elemento do topo da pilha
- **Peek**: retorna o elemento do topo sem removê-lo

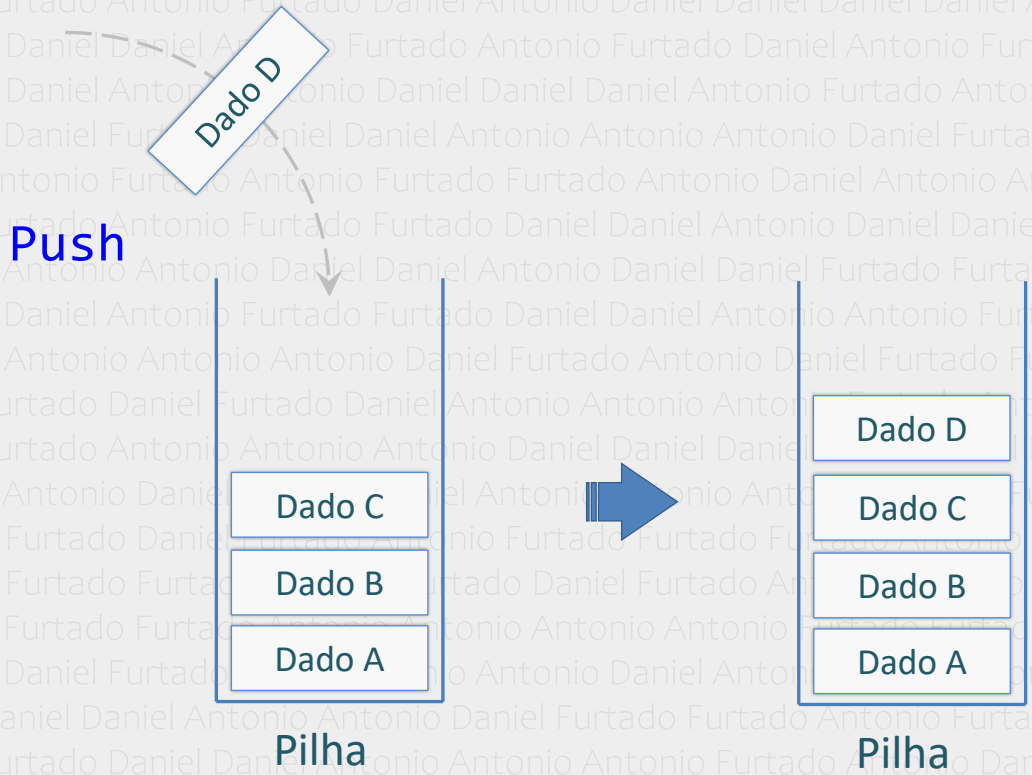
Proibida cópia, apropriação ou uso sem autorização de qualquer parte deste material - Lei nº 9 610/98

TAD Pilha - Operações Adicionais

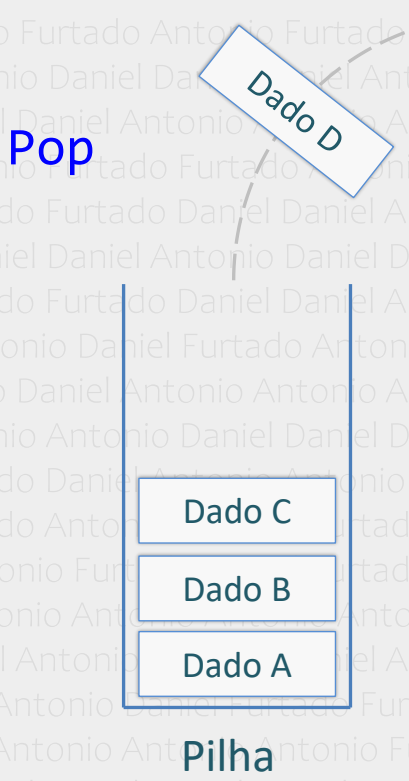
- **`Count`**: retorna o número de elementos na pilha
- **`IsEmpty`**: indica se a pilha está vazia ou não
- **`IsFull`**: indica se a pilha está cheia ou não

Proibida cópia, apropriação ou uso sem autorização de qualquer parte deste material - Lei nº 9 610/98

TAD Pilha - Operação **Push**



TAD Pilha - Operação **Pop**



Implementação em Array

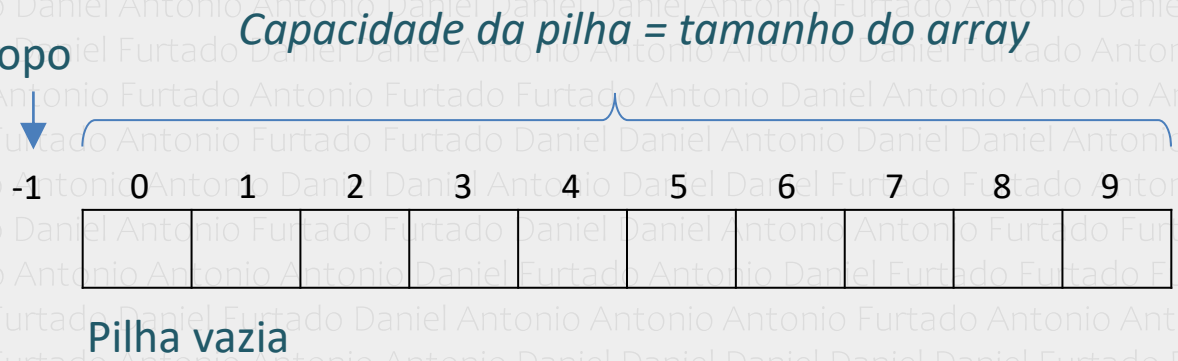
 Proibida cópia, apropriação ou uso sem autorização de qualquer parte deste material - Lei nº 9 610/98

Implementação em Array - Ideia Geral

- Os dados empilhados vão sendo colocados em sequência no array, começando na posição 0
- A posição ocupada pelo último dado inserido representa o topo da pilha (de onde ocorrerá as remoções na operação *pop*)
- O controle da última posição utilizada é feito com o auxilio de uma variável (geralmente denominada *topo,* ou, em inglês, *top*)
- A pilha vazia pode ser representada utilizando o valor -1 para a variável *topo*

Proibida cópia, apropriação ou uso sem autorização de qualquer parte deste material - Lei nº 9 610/98

Implementação em Array - Ideia Geral



Implementação em Array - Ideia Geral

Push(15)

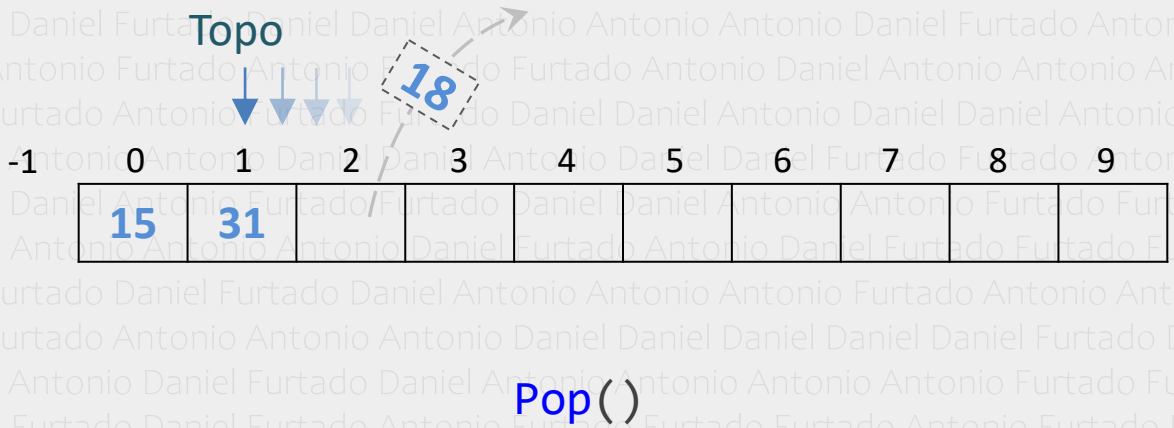
Implementação em Array - Ideia Geral

Push(31)

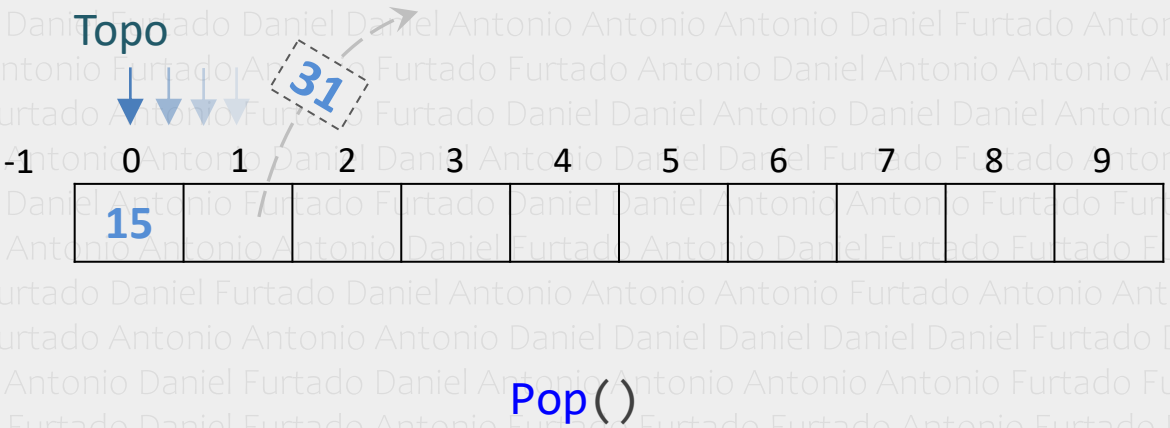
Implementação em Array - Ideia Geral

Push(18)

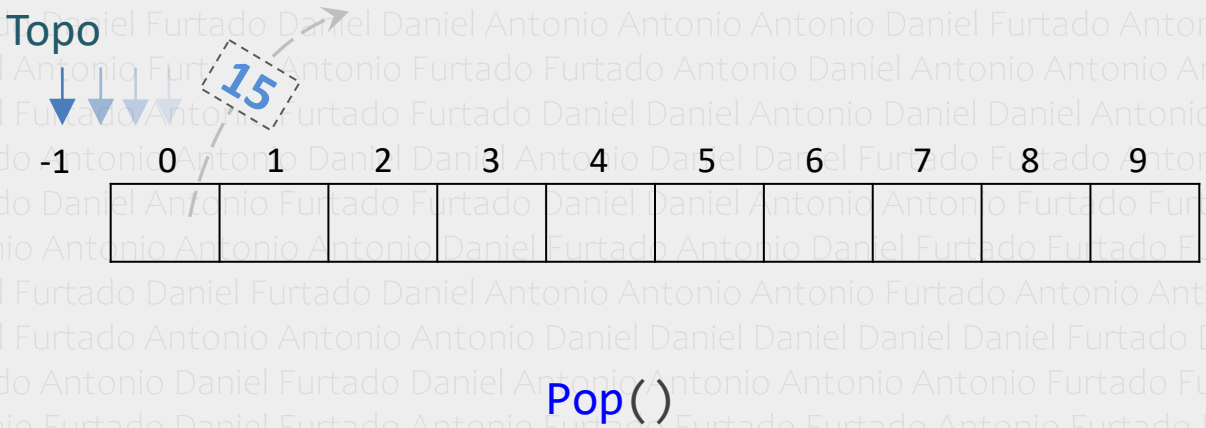
Implementação em Array - Ideia Geral



Implementação em Array - Ideia Geral



Implementação em Array - Ideia Geral



Neste ponto a pilha volta a ficar vazia e **Topo** volta a ter o valor -1

Implementação em Array - Python

```
import array
class Pilha:
    def __init__(self, capacity):
        self.items = array.array('i',[0 for i in range(capacity)])
        self.top = -1

    def Push(self, newItem):
        ...
    def Pop(self):
        ...
    def Peek(self):
        ...
    def IsEmpty(self):
        ...
    def IsFull(self):
        ...
    def Count(self):
        ...
```

Exemplo abreviado de classe em Python correspondente a um **TAD Pilha** em *array* para armazenar inteiros. *A **pilha vazia** é representada com o atributo **top** valendo **-1***

Exemplo de Uso da Pilha - Python

```
# Programa principal
# Cria pilha para armaz. até 10 itens
minhaPilha = Pilha(10)
minhaPilha.Push(2) # Empilha o valor 2
minhaPilha.Push(5) # Empilha o valor 5
minhaPilha.Push(7) # Empilha o valor 7

# Desempilha o topo e mostra na tela (7)
print(minhaPilha.Pop())

# Desempilha o topo e mostra na tela (5)
print(minhaPilha.Pop())

minhaPilha.Push(3)
print(minhaPilha.Pop()) # mostrará 3
print(minhaPilha.Pop()) # mostrará 2
```

Proibida cópia, apropriação ou uso sem autorização de qualquer parte deste material - Lei nº 9 610/98

Método **Push** - Ideia Geral

1. Verificar se a pilha não está cheia
2. Incrementar **top** para a próxima posição vazia do array
3. Armazenar o novo item na posição indicada por **top**

Proibida cópia, apropriação ou uso sem autorização de qualquer parte deste material - Lei nº 9 610/98

Método **Push** - Implementação

```
def Push(self, newItem):
    if self.IsFull():
        raise Exception("Pilha cheia")
    self.top += 1
    self.items[self.top] = newItem
```

OBS: O método IsFull será apresentado posteriormente

Método **Pop** - Ideia Geral

1. Verificar se a pilha não está vazia
2. Resgatar o item da posição indicada por **top**
3. Decrementar **top** para que passe a apontar para o novo topo (ou para -1, caso a pilha fique vazia)
4. Retornar o valor resgatado no passo 2

Proibida cópia, apropriação ou uso sem autorização de qualquer parte deste material - Lei nº 9 610/98

Método **Pop** - Implementação

```
def Pop(self):
    if self.IsEmpty():
        raise Exception("Pilha vazia")
    item = self.items[self.top]
    self.top -= 1
    return item
```

OBS: O método IsEmpty será apresentado posteriormente

Método **Peek**

```
def Peek(self):
    if self.IsEmpty():
        raise Exception("Pilha vazia")
    return self.items[self.top]
```

A operação **Peek** apenas retorna o elemento do topo da pilha, sem removê-lo (sem alterar **top**)

Operações **IsEmpty**, **IsFull** e **Count**

```
def IsEmpty(self):
    return self.top == -1

def IsFull(self):
    return self.top == len(self.items)-1

def Count(self):
    return self.top + 1
```

Implementação em Array

Vantagens

- ✔ Simplicidade
- ✔ Eficiência

Desvantagens

- ✘ Tamanho fixo
- ✘ Desperdício de memória

OBS: É possível ter implementações em *array* com abordagens dinâmicas. Por exemplo, uma possibilidade seria criar um novo *array* com o dobro do tamanho cada vez que o tamanho máximo for ocupado, copiando os elementos para o novo *array*.

Proibida cópia, apropriação ou uso sem autorização de qualquer parte deste material - Lei nº 9 610/98

Exercício

1. Implementar por completo o TAD Pilha em array utilizando as definições de classe e métodos apresentadas neste material
2. Acrescentar um método na classe para exibir, sem modificar a pilha, todos os seus elementos (do primeiro ao último inserido)
3. Criar um código principal para exemplificar o uso da pilha. Inserir elementos, remover, mostrar, etc.

Proibida cópia, apropriação ou uso sem autorização de qualquer parte deste material - Lei nº 9 610/98

Referências

- Narasimha Karumanchi. **Data Structures and Algorithms Made Easy**: Data Structures and Algorithmic Puzzles. CareerMonk Publications, 5th Edition, 2016.
- Michael T. Goodrich, Roberto Tamassia, Michael H. Goldwasser. **Data Structures and Algorithms in Java.** Wiley, 6th Edition, 2014.
- Marcin Jamro. **C# Data Structures and Algorithms:** Explore the possibilities of C# for developing a variety of efficient applications. Packt Publishing, 2018.
- GOODRICH, M.T.; TAMASSIA, R.; GOLDWASSER, M.H. **Data structures and algorithms in Python**. John Wiley & Sons Ltd, 2013.
- BAKA, B. **Python Data Structures and Algorithms**: Improve application performance with graphs, stacks, and queues. Packt Publishing, 2017.