

# Algoritmos e Estruturas de Dados I

## Pilhas

Prof. Flávio José M. Coelho

[fcoelho@uea.edu.br](mailto:fcoelho@uea.edu.br)

# Objetivos

1. Entender o que são pilhas
2. Conhecer a **utilidade** das pilhas.
3. Entender a **implementação** de um **TAD pilha**.

# Pilhas



Uma **pilha** (*stack*) contém itens empilhados...



... mas não dessa forma!



Uma pilha de pizzas.

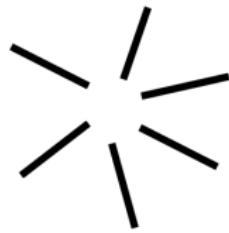


O **fundo** guarda o item mais **antigo**.



O **topo** guarda o item mais **recente**.

Uma pilha **vazia** não contém itens.



Itens podem ser  
**empilhados** (*push*).



Itens podem ser  
**empilhados** (*push*).



Itens podem ser  
**empilhados** (*push*).



Itens podem ser  
**empilhados** (*push*).



Itens podem ser  
**empilhados** (*push*).



E, desempilhados  
(pop)...



E, desempilhados  
(pop)...



E, desempilhados  
(pop)...



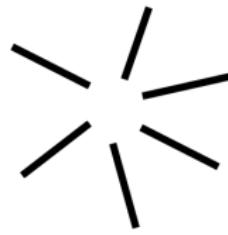
E, desempilhados  
(pop)...



E, desempilhados  
(pop)...



...até a pilha voltar a ficar vazia.



# Pilhas: propriedades

1. É uma lista **LIFO** (*last in, first out*): último item a entrar é o primeiro a sair.
2. **Topo**: item mais recente; **fundo**: item mais antigo.

# Pilhas: aplicações

1. Memorizar tarefas a serem executadas posteriormente;
2. Controle de chamadas a subrotinas em programas (usado para implementar (**recursividade**));
3. Análise sintática de programas;
4. Avaliação de expressões aritméticas (notação polonesa reversa).

# Pilhas: aplicações

Notação polonesa reversa (NPR, posfixada): operadores vêm depois dos operandos em expressões matemáticas.

**Infixada:** “1 + 2”    **NPR:** “1 2 +”

“(2 - 3) \* 7”  $\Rightarrow$  “2 3 - 7 \*”

“2 - (3 \* 7)”  $\Rightarrow$  “2 3 7 \* -”

“(2 - 3) \* (7 + 5)”  $\Rightarrow$  “2 3 - 7 5 + \*”

# Pilhas: aplicações

Avaliando expressão em NPR, por pilha:

# Pilhas: aplicações

Avaliando expressão em NPR, por pilha:

1. Se você leu um número da entrada,  
**empilhe-o na pilha**;

# Pilhas: aplicações

Avaliando expressão em NPR, por pilha:

1. Se você leu um número da entrada,  
**empilhe-o na pilha**;
2. Se você leu um operador binário,  
**desempilhe** dois itens da pilha, aplique o  
operador sobre os valores e **empilhe** o  
resultado;

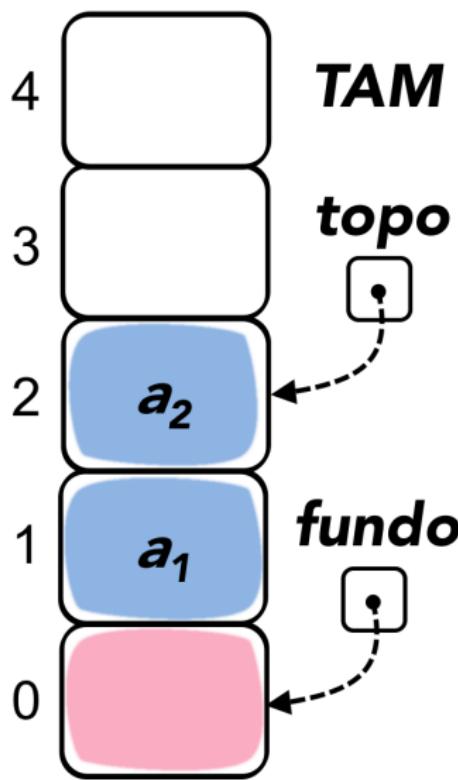
# Pilhas: aplicações

Avaliando expressão em NPR, por pilha:

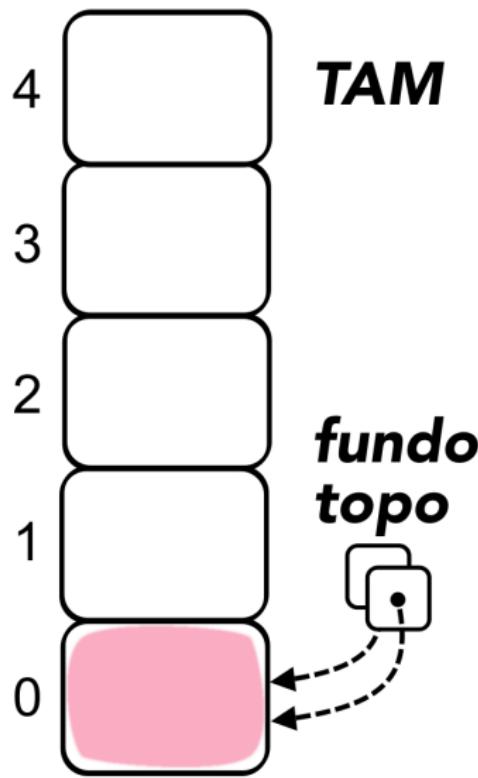
1. Se você leu um número da entrada, **empilhe-o na pilha**;
2. Se você leu um operador binário, **desempilhe** dois itens da pilha, aplique o operador sobre os valores e **empilhe** o resultado;
3. Quando a entrada terminar, o resultado final estará no **topo** da pilha.

# TAD Pilha

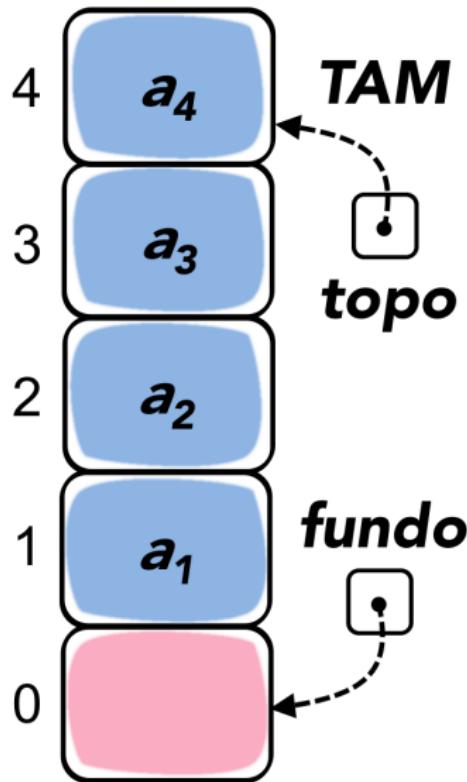
## (implementação estática)



A pilha é uma lista estática adaptada.



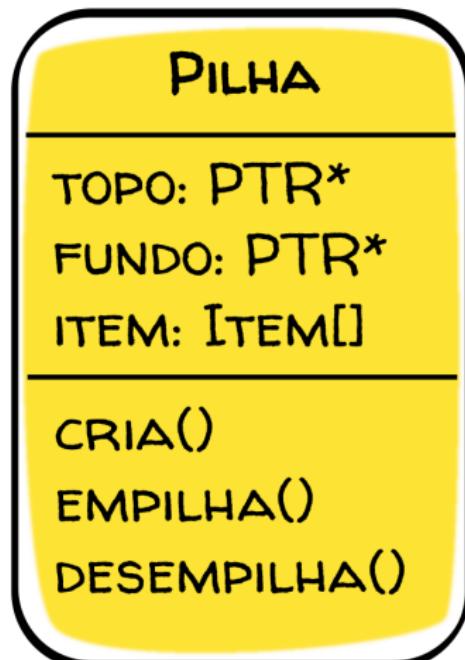
A pilha está vazia quando **topo** = **fundo**.



A pilha está cheia quando **topo** = **TAM**.

**TAM** é o tamanho do vetor.

# TAD Pilha



\* PONTEIRO LÓGICO = INT

# Operações

As operações de criação, empilhamento e desempilhamento do **TAD Pilha** são, respectivamente, adaptações das operações de criação, inserção e remoção do **TAD Lista** com implementação estática.

## CRIA(P)

1 P.fundo = 0

2 P.topo = P.fundo

## EMPILHA(P, item)

- 1    **se** P.topo == MAX
- 2        pilha cheia
- 3    **senão**
- 4        P.topo = P.topo + 1
- 5        P.item[P.topo] = item

## DESEMPILHA(P, item)

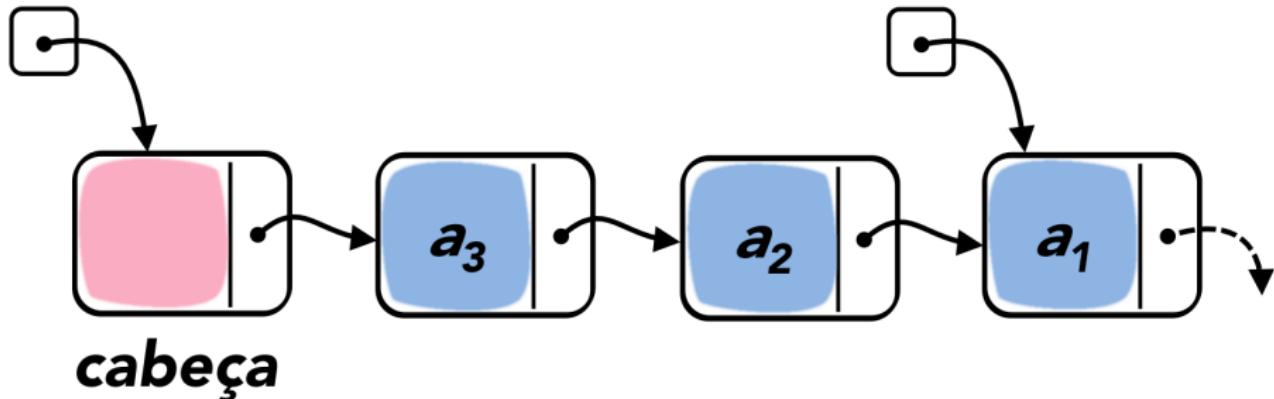
```
1  se P.topo == P.fundo  
2      pilha vazia  
3  senão  
4      item = P.item[P.topo]  
5      P.topo = P.topo - 1
```

# TAD Pilha

## (implementação dinâmica)

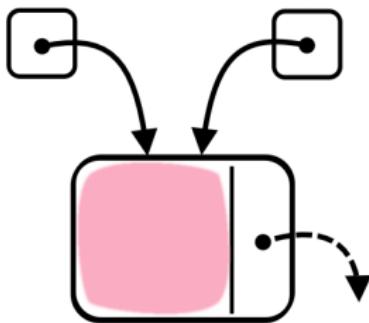
**topo**

**fundo**



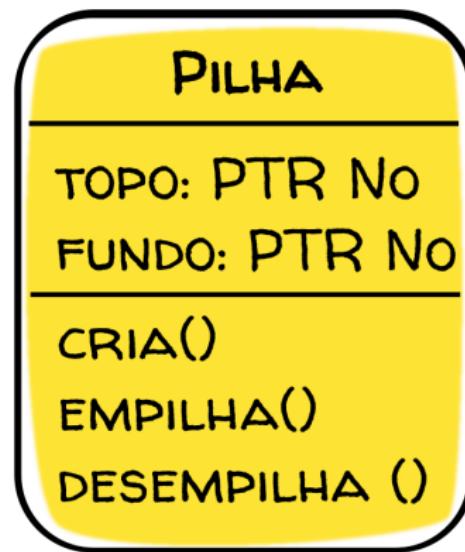
A pilha é uma lista dinâmica adaptada.

**fundo**      **topo**



A pilha está vazia quando **topo** = **fundo**.

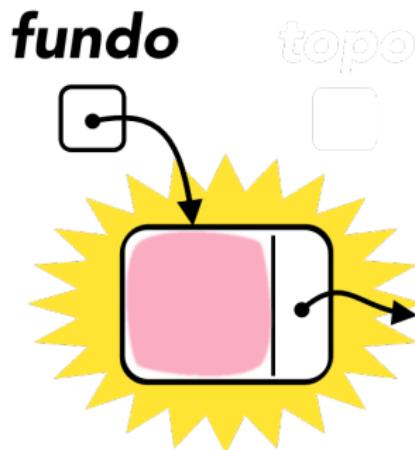
# TAD Pilha



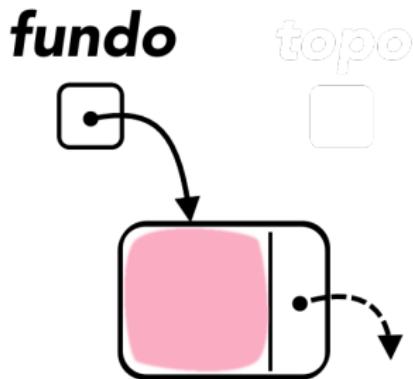
# Operações

As operações de criação, empilhamento e desempilhamento do **TAD Pilha** são, respectivamente, adaptações das operações de criação, inserção e remoção do **TAD Lista Ligada**.

# CRIA(P)

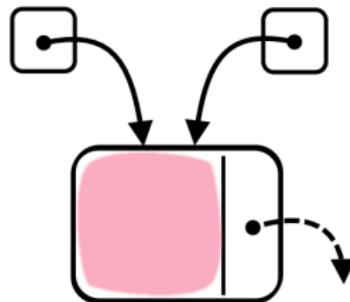


# CRIA(P)



# CRIA(P)

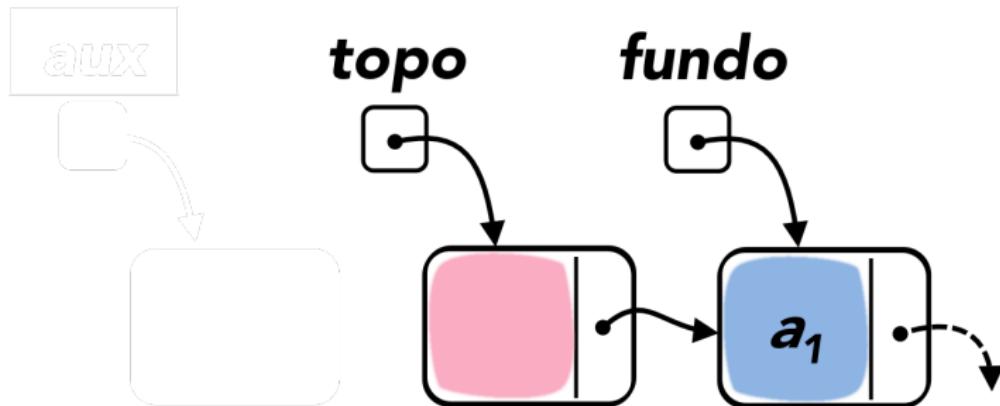
**fundo topo**



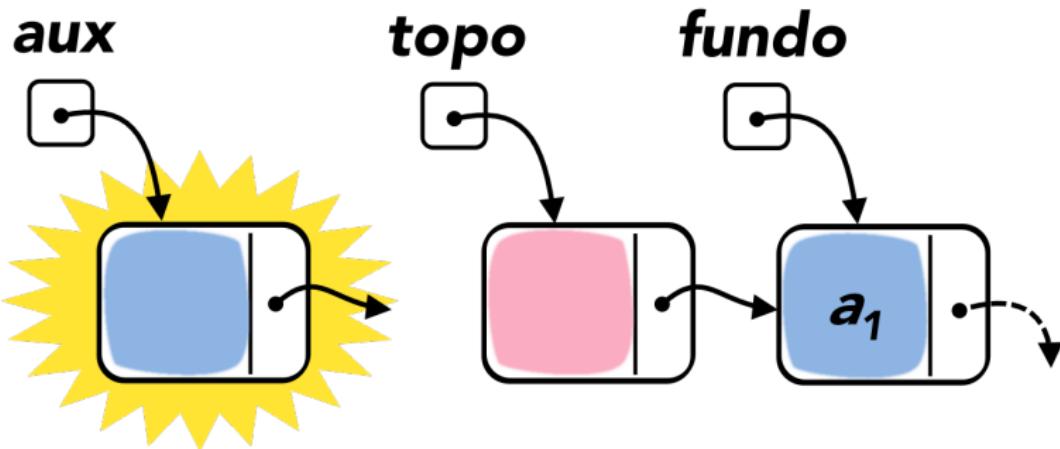
## CRIA(P)

- 1 P.fundo = aloque novo **No** // *cabeça*
- 2 P.fundo.prox = NIL
- 3 P.topo = P.fundo

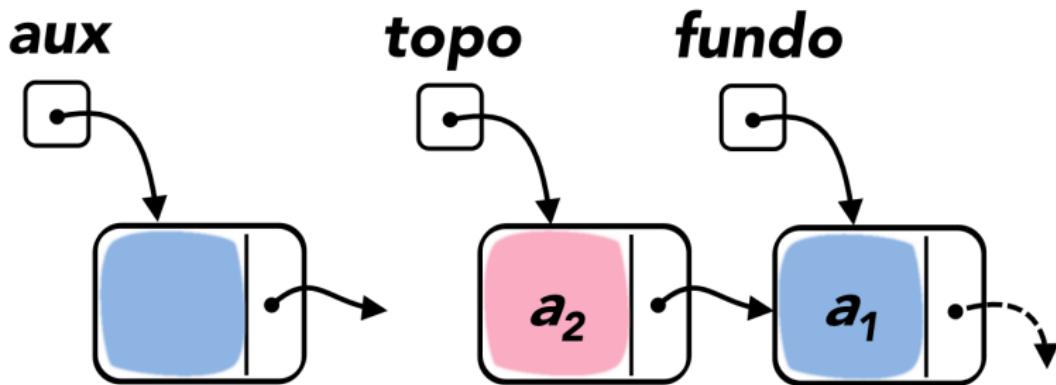
# EMPILHA(P, item)



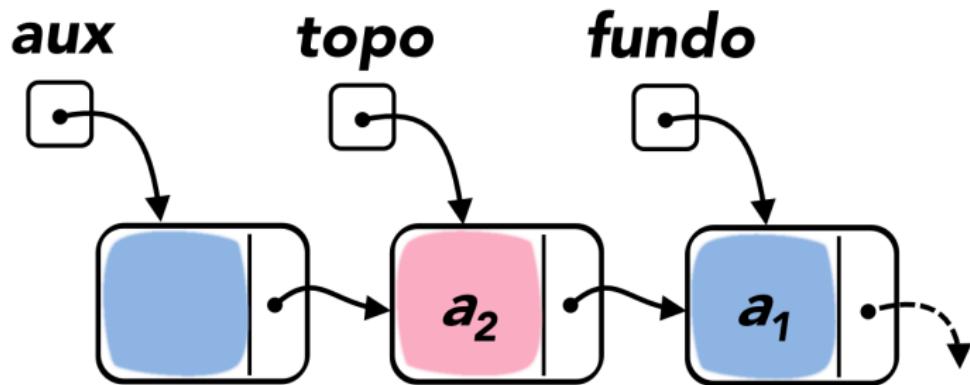
# EMPILHA(P, item)



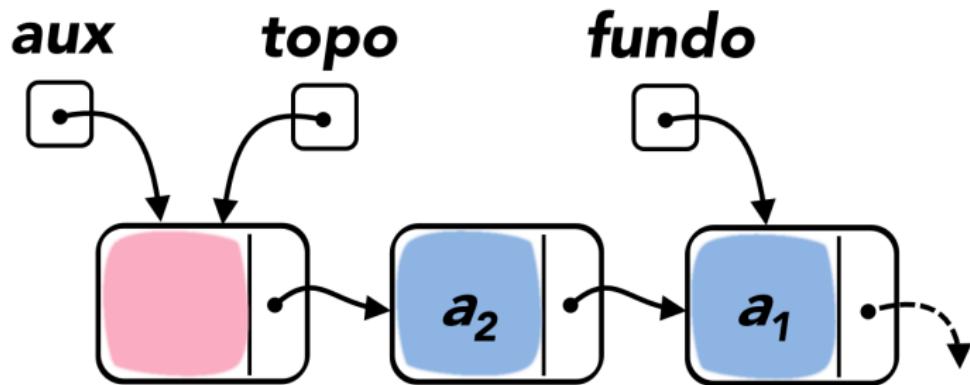
# EMPILHA(P, item)



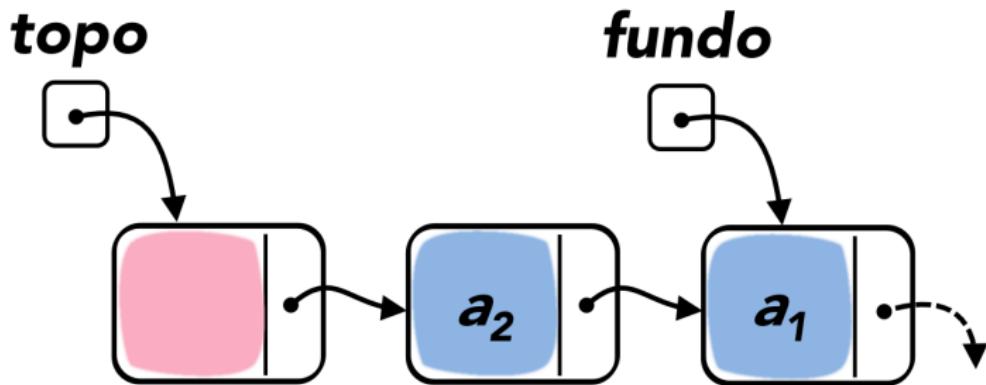
# EMPILHA(P, item)



# EMPILHA(P, item)



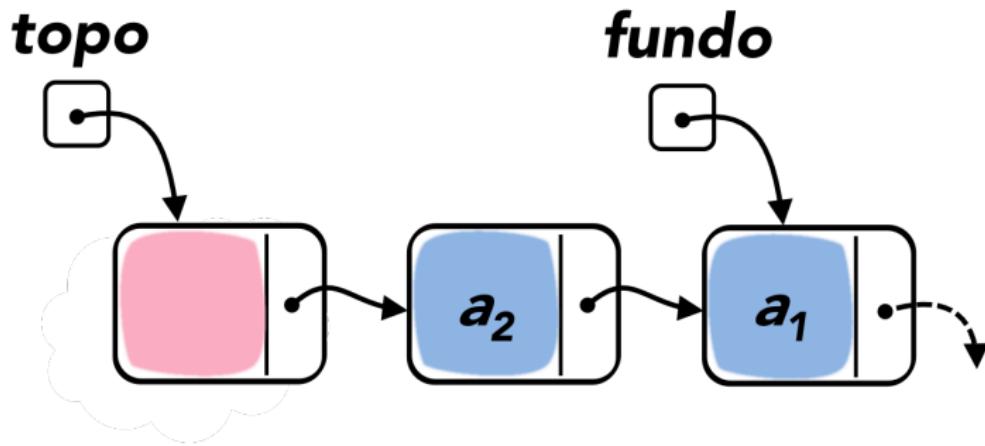
# EMPILHA(P, item)



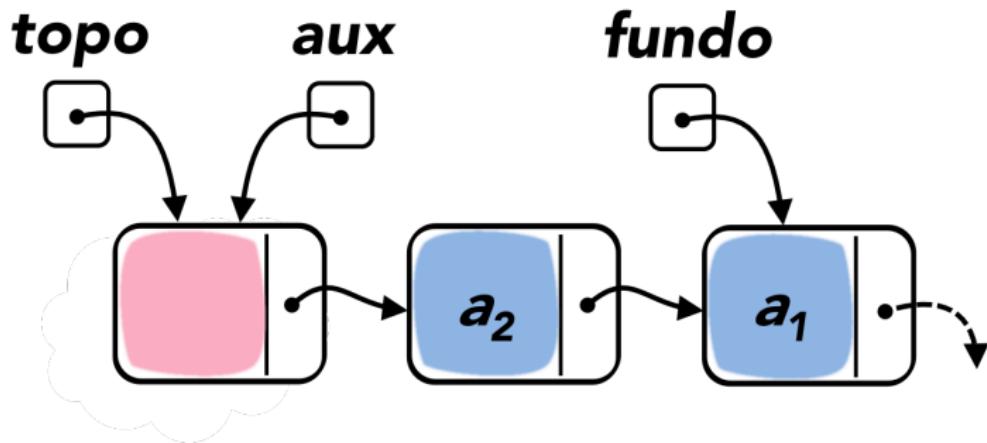
## EMPILHA(P, item)

- 1 aux = aloque novo **No** // nova cabeça
- 2 P.topo.item = item
- 3 aux.prox = P.topo
- 4 P.topo = aux

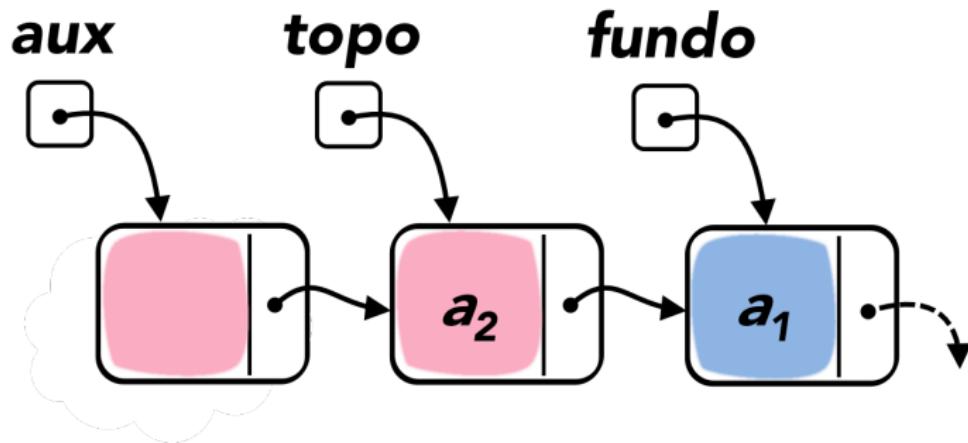
# DESEMPILHA(P, item)



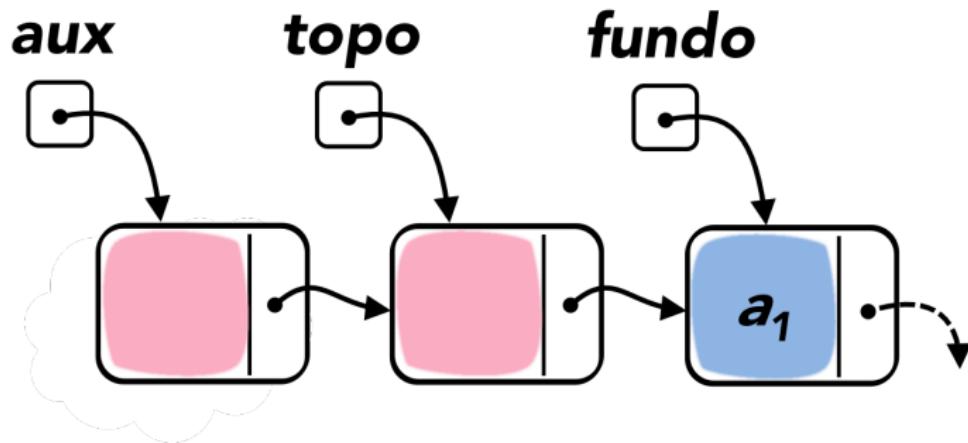
# DESEMPILHA(P, item)



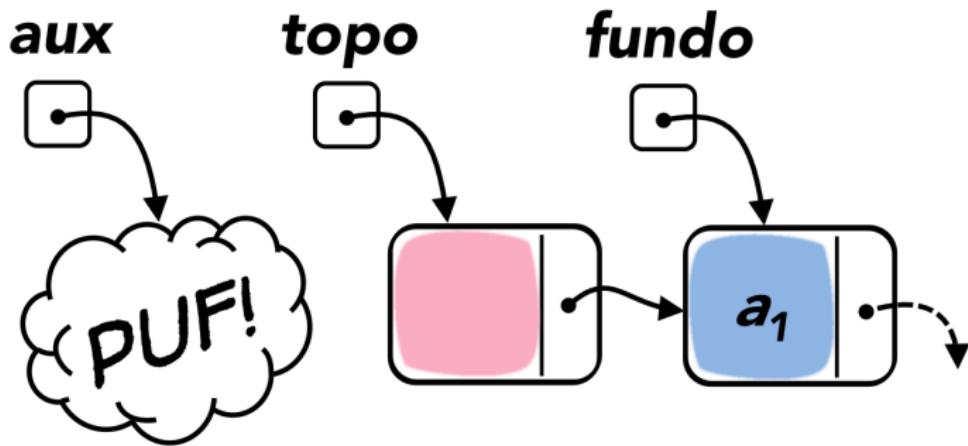
# DESEMPILHA(P, item)



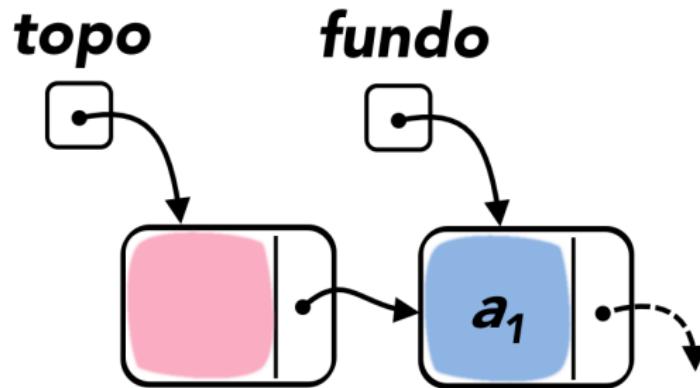
# DESEMPILHA(P, item)



# DESEMPILHA(P, item)



# DESEMPILHA(P, item)



## DESEMPILHA(P, item)

- 1 aux = P.topo
- 2 P.topo = P.topo.prox
- 3 item = P.topo.item
- 4 desaloque aux

# Referências

-  Glenn G. Chappell, CS311 - Data Structures and Algorithms Lecture Slides - Stacks. Department of Computer Science University of Alaska Fairbanks. November, 2009.
-  T. H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, 3rd edition, MIT Press, 2010
-  N. Ziviani. Projeto de Algoritmos com Implementação em Pascal C. Cengage Learning, 2012.

## Onde obter este material:

[est.uea.edu.br/fcoelho](http://est.uea.edu.br/fcoelho)