

# Aula 10

## Pilhas, Filas e Listas Biligadas

*Programação II, 2018-2019*

*v1.2, 27-04-2018*

### Pilhas e filas

Definições e tipos de dados  
abstratos

Implementação em lista  
ligada

Implementação em vector

### Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas

### Pilhas e filas

Definições e tipos de dados  
abstratos

Implementação em lista  
ligada

Implementação em vector

### Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas

## 1 Pilhas e filas

Definições e tipos de dados abstratos

Implementação em lista ligada

Implementação em vector

## 2 Listas biligadas

## 3 Comparação entre diferentes tipos de listas ligadas

### Pilhas e filas

Definições e tipos de dados  
abstratos

Implementação em lista  
ligada

Implementação em vector

### Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas

## 1 Pilhas e filas

Definições e tipos de dados abstratos

Implementação em lista ligada

Implementação em vector

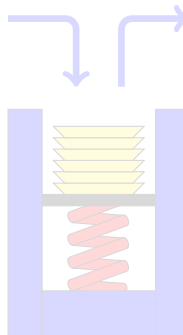
## 2 Listas biligadas

## 3 Comparação entre diferentes tipos de listas ligadas

# Pilha: definição

- É uma estrutura de dados que só pode ser modificada por uma das suas extremidades usualmente denominada como “topo”.

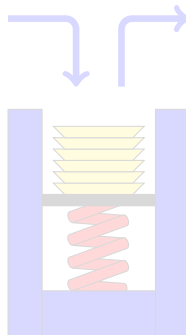
- Estrutura com gestão *LIFO*: *Last In First Out*;
  - O último elemento a entrar é o primeiro a sair.



# Pilha: definição

- É uma estrutura de dados que só pode ser modificada por uma das suas extremidades usualmente denominada como “topo”.

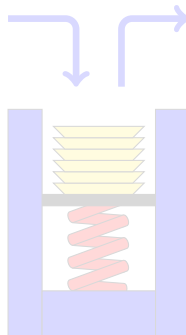
- Estrutura com gestão *LIFO*: *Last In First Out*;  
O último elemento a entrar é o primeiro a sair.



# Pilha: definição

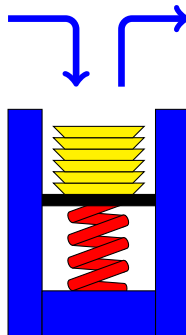
- É uma estrutura de dados que só pode ser modificada por uma das suas extremidades usualmente denominada como “topo”.

- Estrutura com gestão *LIFO*: *Last In First Out*;  
O último elemento a entrar é o primeiro a sair.



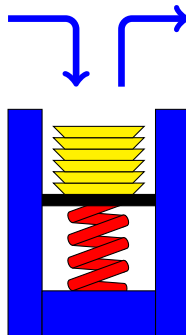
- É uma estrutura de dados que só pode ser modificada por uma das suas extremidades usualmente denominada como “topo”.

- Estrutura com gestão *LIFO*: *Last In First Out*;
  - O último elemento a entrar é o primeiro a sair.



- É uma estrutura de dados que só pode ser modificada por uma das suas extremidades usualmente denominada como “topo”.

- Estrutura com gestão *LIFO*: *Last In First Out*;
  - O último elemento a entrar é o primeiro a sair.





# Pilha: as operações push / pop

## Pilhas e filas

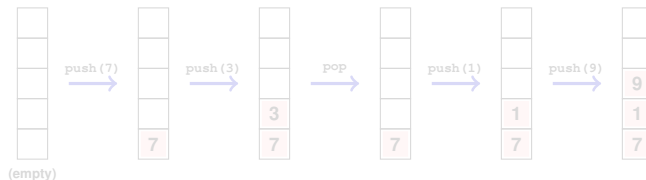
Definições e tipos de dados  
abstratos

Implementação em lista  
ligada

Implementação em vector

## Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas



# Pilha: as operações push / pop

## Pilhas e filas

Definições e tipos de dados  
abstratos

Implementação em lista  
ligada

Implementação em vector

## Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas



# Pilha: as operações push / pop

## Pilhas e filas

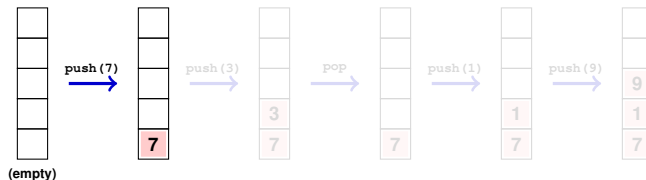
Definições e tipos de dados  
abstratos

Implementação em lista  
ligada

Implementação em vector

## Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas



# Pilha: as operações push / pop

## Pilhas e filas

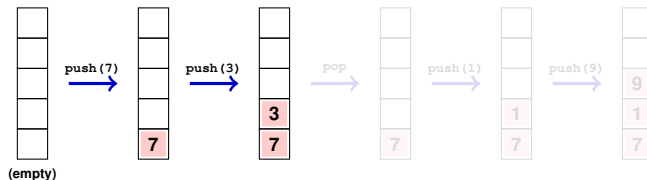
Definições e tipos de dados  
abstratos

Implementação em lista  
ligada

Implementação em vector

## Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas



# Pilha: as operações push / pop

## Pilhas e filas

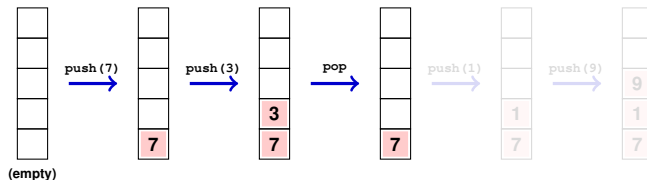
Definições e tipos de dados  
abstratos

Implementação em lista  
ligada

Implementação em vector

## Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas



# Pilha: as operações push / pop

## Pilhas e filas

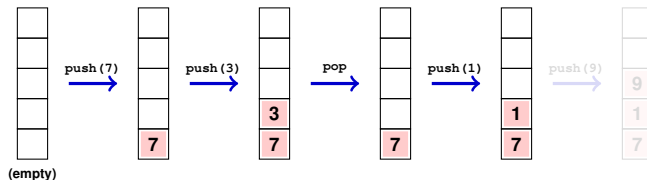
Definições e tipos de dados  
abstratos

Implementação em lista  
ligada

Implementação em vector

## Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas



# Pilha: as operações push / pop

## Pilhas e filas

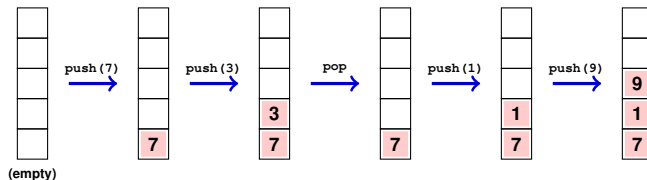
Definições e tipos de dados  
abstratos

Implementação em lista  
ligada

Implementação em vector

## Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas



# Pilha: exemplos de utilização

- Armazenamento de contextos de execução de subrotinas.
- Análise e avaliação de expressões matemáticas.
- Travessia *depth-first* de árvores e grafos.
- Algoritmos de tratamento de texto:
  - Inversão de cadeia de caracteres.
  - Detecção de palíndromo.
- ...



- Armazenamento de contextos de execução de subrotinas.
- Análise e avaliação de expressões matemáticas.
- Travessia *depth-first* de árvores e grafos.
- Algoritmos de tratamento de texto:
  - Inversão de cadeias de caracteres.
  - Detecção de palíndromo.
- ...

- Armazenamento de contextos de execução de subrotinas.
- Análise e avaliação de expressões matemáticas.
- Travessia *depth-first* de árvores e grafos.
- Algoritmos de tratamento de texto:
  - Inversão de cadeias de caracteres.
  - Detecção de palíndromo.
- ...

- Armazenamento de contextos de execução de subrotinas.
- Análise e avaliação de expressões matemáticas.
- Travessia *depth-first* de árvores e grafos.
- Algoritmos de tratamento de texto:
  - Inversão de cadeias de caracteres.
  - Detecção de palíndromo.
- ...

- Armazenamento de contextos de execução de subrotinas.
- Análise e avaliação de expressões matemáticas.
- Travessia *depth-first* de árvores e grafos.
- Algoritmos de tratamento de texto:
  - Inversão de cadeias de caracteres.
  - Detecção de palíndromo.
- ...

- Armazenamento de contextos de execução de subrotinas.
- Análise e avaliação de expressões matemáticas.
- Travessia *depth-first* de árvores e grafos.
- Algoritmos de tratamento de texto:
  - Inversão de cadeias de caracteres.
  - Detecção de palíndromo.
- . . .

## Pilhas e filas

### Definições e tipos de dados abstratos

Implementação em lista  
ligada

Implementação em vector

## Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas

- Armazenamento de contextos de execução de subrotinas.
- Análise e avaliação de expressões matemáticas.
- Travessia *depth-first* de árvores e grafos.
- Algoritmos de tratamento de texto:
  - Inversão de cadeias de caracteres.
  - Detecção de palíndromo.

• . . .

## Pilhas e filas

### Definições e tipos de dados abstratos

Implementação em lista ligada

Implementação em vector

## Listas biligadas

Comparação entre diferentes tipos de listas ligadas

- Armazenamento de contextos de execução de subrotinas.
- Análise e avaliação de expressões matemáticas.
- Travessia *depth-first* de árvores e grafos.
- Algoritmos de tratamento de texto:
  - Inversão de cadeias de caracteres.
  - Detecção de palíndromo.
- ...

# Pilha: tipo de dados abstrato

## Pilhas e filas

### Definições e tipos de dados abstratos

Implementação em lista ligada

Implementação em vector

### Listas biligadas

Comparação entre diferentes tipos de listas ligadas

- Nome do módulo:

  - Stack

- Serviços:

  - `push(elemento)` empilha um elemento no topo da pilha
  - `pop()` remove (desempilha) o elemento no topo da pilha
  - `top()` devolve o elemento no topo da pilha
  - `isEmpty()` verifica se a pilha está vazia
  - `isFull()` verifica se a pilha está cheia
  - `size()` retorna o tamanho actual da pilha
  - `clear()` limpa a pilha (retira todos os elementos)



- Nome do módulo:
  - `Stack`
- Serviços:
  - `push`: insere (empilha) um elemento no topo da pilha
  - `pop`: remove (desempilha) o elemento no topo da pilha
  - `top`: devolve o elemento no topo da pilha
  - `isEmpty`: verifica se a pilha está vazia
  - `isFull`: verifica se a pilha está cheia
  - `size`: retorna a dimensão actual da pilha
  - `clear`: limpa a pilha (retira todos os elementos)

- Nome do módulo:
  - `Stack`
- Serviços:
  - `push`: insere (empilha) um elemento no topo da pilha
  - `pop`: remove (desempilha) o elemento no topo da pilha
  - `top`: devolve o elemento no topo da pilha
  - `isEmpty`: verifica se a pilha está vazia
  - `isFull`: verifica se a pilha está cheia
  - `size`: retorna a dimensão actual da pilha
  - `clear`: limpa a pilha (retira todos os elementos)

- Nome do módulo:
  - `Stack`
- Serviços:
  - `push`: insere (empilha) um elemento no topo da pilha
  - `pop`: remove (desempilha) o elemento no topo da pilha
  - `top`: devolve o elemento no topo da pilha
  - `isEmpty`: verifica se a pilha está vazia
  - `isFull`: verifica se a pilha está cheia
  - `size`: retorna a dimensão actual da pilha
  - `clear`: limpa a pilha (retira todos os elementos)

- Nome do módulo:
  - `Stack`
- Serviços:
  - `push`: insere (empilha) um elemento no topo da pilha
  - `pop`: remove (desempilha) o elemento no topo da pilha
  - `top`: devolve o elemento no topo da pilha
  - `isEmpty`: verifica se a pilha está vazia
  - `isFull`: verifica se a pilha está cheia
  - `size`: retorna a dimensão actual da pilha
  - `clear`: limpa a pilha (retira todos os elementos)

- Nome do módulo:
  - `Stack`
- Serviços:
  - `push`: insere (empilha) um elemento no topo da pilha
  - `pop`: remove (desempilha) o elemento no topo da pilha
  - `top`: devolve o elemento no topo da pilha
  - `isEmpty`: verifica se a pilha está vazia
  - `isFull`: verifica se a pilha está cheia
  - `size`: retorna a dimensão actual da pilha
  - `clear`: limpa a pilha (retira todos os elementos)

- Nome do módulo:
  - `Stack`
- Serviços:
  - `push`: insere (empilha) um elemento no topo da pilha
  - `pop`: remove (desempilha) o elemento no topo da pilha
  - `top`: devolve o elemento no topo da pilha
  - `isEmpty`: verifica se a pilha está vazia
  - `isFull`: verifica se a pilha está cheia
  - `size`: retorna a dimensão actual da pilha
  - `clear`: limpa a pilha (retira todos os elementos)

- Nome do módulo:
  - `Stack`
- Serviços:
  - `push`: insere (empilha) um elemento no topo da pilha
  - `pop`: remove (desempilha) o elemento no topo da pilha
  - `top`: devolve o elemento no topo da pilha
  - `isEmpty`: verifica se a pilha está vazia
  - `isFull`: verifica se a pilha está cheia
  - `size`: retorna a dimensão actual da pilha
  - `clear`: limpa a pilha (retira todos os elementos)

- Nome do módulo:
  - `Stack`
- Serviços:
  - `push`: insere (empilha) um elemento no topo da pilha
  - `pop`: remove (desempilha) o elemento no topo da pilha
  - `top`: devolve o elemento no topo da pilha
  - `isEmpty`: verifica se a pilha está vazia
  - `isFull`: verifica se a pilha está cheia
  - `size`: retorna a dimensão actual da pilha
  - `clear`: limpa a pilha (retira todos os elementos)



- Nome do módulo:
  - `Stack`
- Serviços:
  - `push`: insere (empilha) um elemento no topo da pilha
  - `pop`: remove (desempilha) o elemento no topo da pilha
  - `top`: devolve o elemento no topo da pilha
  - `isEmpty`: verifica se a pilha está vazia
  - `isFull`: verifica se a pilha está cheia
  - `size`: retorna a dimensão actual da pilha
  - `clear`: limpa a pilha (retira todos os elementos)

- Nome do módulo:
  - `Stack`
- Serviços:
  - `push`: insere (empilha) um elemento no topo da pilha
  - `pop`: remove (desempilha) o elemento no topo da pilha
  - `top`: devolve o elemento no topo da pilha
  - `isEmpty`: verifica se a pilha está vazia
  - `isFull`: verifica se a pilha está cheia
  - `size`: retorna a dimensão actual da pilha
  - `clear`: limpa a pilha (retira todos os elementos)

## Pilhas e filas

### Definições e tipos de dados abstratos

Implementação em lista ligada

Implementação em vector

## Listas biligadas

Comparação entre diferentes tipos de listas ligadas

- **push(e)**

- Pré-condição:  $!isFull()$

- Pós-condição:  $!isEmpty() \wedge e == top() == e$

- **pop()**

- Pré-condição:  $!isEmpty()$

- Pós-condição:  $!isFull()$

- **top()**

- Pré-condição:  $!isEmpty()$

- **push(e)**
  - Pré-condição: `!isFull()`
  - Pós-condição: `!isEmpty() && (top() == e)`
- **pop()**
  - Pré-condição: `!isEmpty()`
  - Pós-condição: `!isFull()`
- **top()**
  - Pré-condição: `!isEmpty()`

- **push(e)**
  - Pré-condição: `!isFull()`
  - Pós-condição: `!isEmpty() && (top() == e)`
- **pop()**
  - Pré-condição: `!isEmpty()`
  - Pós-condição: `!isFull()`
- **top()**
  - Pré-condição: `!isEmpty()`

- **push(e)**
  - Pré-condição: `!isFull()`
  - Pós-condição: `!isEmpty() && (top() == e)`
- **pop()**
  - Pré-condição: `!isEmpty()`
  - Pós-condição: `!isFull()`
- **top()**
  - Pré-condição: `!isEmpty()`

- **push(e)**
  - Pré-condição: `!isFull()`
  - Pós-condição: `!isEmpty() && (top() == e)`
- **pop()**
  - Pré-condição: `!isEmpty()`
  - Pós-condição: `!isFull()`
- **top()**
  - Pré-condição: `!isEmpty()`

- **push(e)**
  - Pré-condição: `!isFull()`
  - Pós-condição: `!isEmpty() && (top() == e)`
- **pop()**
  - Pré-condição: `!isEmpty()`
  - Pós-condição: `!isFull()`
- **top()**
  - Pré-condição: `!isEmpty()`



- **push(e)**
  - Pré-condição: `!isFull()`
  - Pós-condição: `!isEmpty() && (top() == e)`
- **pop()**
  - Pré-condição: `!isEmpty()`
  - Pós-condição: `!isFull()`
- **top()**
  - Pré-condição: `!isEmpty()`

- **push(e)**
  - Pré-condição: `!isFull()`
  - Pós-condição: `!isEmpty() && (top() == e)`
- **pop()**
  - Pré-condição: `!isEmpty()`
  - Pós-condição: `!isFull()`
- **top()**
  - Pré-condição: `!isEmpty()`

- **push(e)**
  - Pré-condição: `!isFull()`
  - Pós-condição: `!isEmpty() && (top() == e)`
- **pop()**
  - Pré-condição: `!isEmpty()`
  - Pós-condição: `!isFull()`
- **top()**
  - Pré-condição: `!isEmpty()`

# Fila: definição

## Pilhas e filas

Definições e tipos de dados  
abstratos

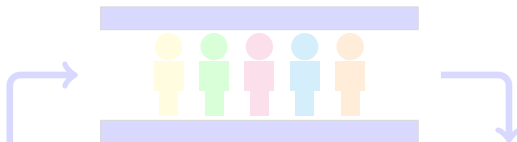
Implementação em lista  
ligada

Implementação em vector

## Listas biligadas

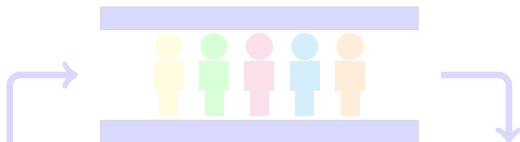
Comparação entre  
diferentes tipos de  
listas ligadas

- É uma estrutura de dados cujo acesso é feito por ambas as extremidades:
  - uma apenas para retirar elementos, e a outra apenas para os adicionar.



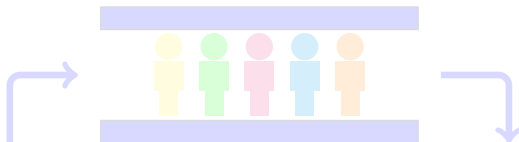
- Gerida segundo uma política *FIFO* (*First In First Out*)
  - retira-se sempre o valor mais antigo primeiro.

- É uma estrutura de dados cujo acesso é feito por ambas as extremidades:
  - uma apenas para colocar elementos, e a outra apenas para os retirar.



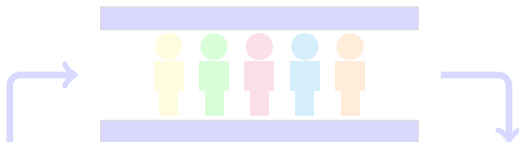
- Gerida segundo uma política *FIFO* (*First In First Out*)
  - Retirado sempre o valor mais antigo primeiro

- É uma estrutura de dados cujo acesso é feito por ambas as extremidades:
  - uma apenas para colocar elementos, e a outra apenas para os retirar.



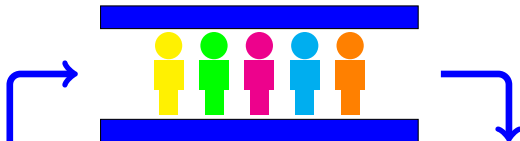
- Gerida segundo uma política *FIFO* (*First In First Out*)  
• Retirado sempre o valor mais antigo primeiro

- É uma estrutura de dados cujo acesso é feito por ambas as extremidades:
  - uma apenas para colocar elementos, e a outra apenas para os retirar.



- Gerida segundo uma política *FIFO* (*First In First Out*)  
• Retira-se sempre o valor mais antigo primeiro

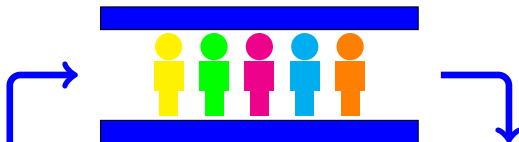
- É uma estrutura de dados cujo acesso é feito por ambas as extremidades:
  - uma apenas para colocar elementos, e a outra apenas para os retirar.



- Gerida segundo uma política *FIFO* (*First In First Out*)  
ou seja, o primeiro a entrar é o primeiro a sair.

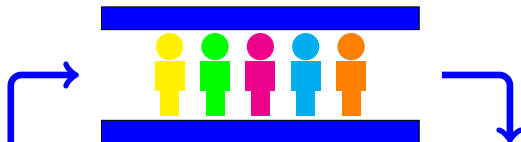


- É uma estrutura de dados cujo acesso é feito por ambas as extremidades:
  - uma apenas para colocar elementos, e a outra apenas para os retirar.



- Gerida segundo uma política *FIFO* (*First In First Out*)
  - extrai-se sempre o valor mais antigo primeiro.

- É uma estrutura de dados cujo acesso é feito por ambas as extremidades:
  - uma apenas para colocar elementos, e a outra apenas para os retirar.



- Gerida segundo uma política *FIFO* (*First In First Out*)
  - extrai-se sempre o valor mais antigo primeiro.

# Fila: tipo de dados abstrato

- Nome do módulo:

  - Queue

- Serviços:

  - `add(x)`: insere um elemento no fim da fila
  - `pop()`: retira o elemento do início da fila
  - `peek()`: retorna o elemento do início da fila
  - `isEmpty()`: verifica se a fila está vazia
  - `isFull()`: verifica se a fila está cheia
  - `size()`: retorna a dimensão atual da fila
  - `clear()`: limpa a fila (retirando todos os elementos)

## Pilhas e filas

### Definições e tipos de dados abstratos

Implementação em lista ligada

Implementação em vector

## Listas biligadas

Comparação entre diferentes tipos de listas ligadas

- Nome do módulo:
  - Queue
- Serviços:
  - in: insere um elemento no fim da fila
  - out: retira elemento do início da fila
  - peek: retorna o elemento do início da fila
  - isEmpty: verifica se a fila está vazia
  - isFull: verifica se a fila está cheia
  - size: retorna a dimensão actual da fila
  - clear: limpa a fila (retira todos os elementos)

- Nome do módulo:
  - Queue
- Serviços:
  - in: insere um elemento no fim da fila
  - out: retira elemento do início da fila
  - peek: retorna o elemento do início da fila
  - isEmpty: verifica se a fila está vazia
  - isFull: verifica se a fila está cheia
  - size: retorna a dimensão actual da fila
  - clear: limpa a fila (retira todos os elementos)

- Nome do módulo:
  - Queue
- Serviços:
  - `in`: insere um elemento no fim da fila
  - `out`: retira elemento do início da fila
  - `peek`: retorna o elemento do inicio da fila
  - `isEmpty`: verifica se a fila está vazia
  - `isFull`: verifica se a fila está cheia
  - `size`: retorna a dimensão actual da fila
  - `clear`: limpa a fila (retira todos os elementos)

- Nome do módulo:
  - Queue
- Serviços:
  - `in`: insere um elemento no fim da fila
  - `out`: retira elemento do início da fila
  - `peek`: retorna o elemento do início da fila
  - `isEmpty`: verifica se a fila está vazia
  - `isFull`: verifica se a fila está cheia
  - `size`: retorna a dimensão actual da fila
  - `clear`: limpa a fila (retira todos os elementos)

- Nome do módulo:
  - Queue
- Serviços:
  - in: insere um elemento no fim da fila
  - out: retira elemento do início da fila
  - peek: retorna o elemento do início da fila
  - isEmpty: verifica se a fila está vazia
  - isFull: verifica se a fila está cheia
  - size: retorna a dimensão actual da fila
  - clear: limpa a fila (retira todos os elementos)



- Nome do módulo:
  - Queue
- Serviços:
  - `in`: insere um elemento no fim da fila
  - `out`: retira elemento do início da fila
  - `peek`: retorna o elemento do inicio da fila
  - `isEmpty`: verifica se a fila está vazia
  - `isFull`: verifica se a fila está cheia
  - `size`: retorna a dimensão actual da fila
  - `clear`: limpa a fila (retira todos os elementos)

- Nome do módulo:
  - Queue
- Serviços:
  - `in`: insere um elemento no fim da fila
  - `out`: retira elemento do início da fila
  - `peek`: retorna o elemento do inicio da fila
  - `isEmpty`: verifica se a fila está vazia
  - `isFull`: verifica se a fila está cheia
  - `size`: retorna a dimensão actual da fila
  - `clear`: limpa a fila (retira todos os elementos)

- Nome do módulo:
  - Queue
- Serviços:
  - **in**: insere um elemento no fim da fila
  - **out**: retira elemento do início da fila
  - **peek**: retorna o elemento do inicio da fila
  - **isEmpty**: verifica se a fila está vazia
  - **isFull**: verifica se a fila está cheia
  - **size**: retorna a dimensão actual da fila
  - **clear**: limpa a fila (retira todos os elementos)

- Nome do módulo:
  - Queue
- Serviços:
  - in: insere um elemento no fim da fila
  - out: retira elemento do início da fila
  - peek: retorna o elemento do inicio da fila
  - isEmpty: verifica se a fila está vazia
  - isFull: verifica se a fila está cheia
  - size: retorna a dimensão actual da fila
  - clear: limpa a fila (retira todos os elementos)

- Nome do módulo:
  - Queue
- Serviços:
  - `in`: insere um elemento no fim da fila
  - `out`: retira elemento do início da fila
  - `peek`: retorna o elemento do inicio da fila
  - `isEmpty`: verifica se a fila está vazia
  - `isFull`: verifica se a fila está cheia
  - `size`: retorna a dimensão actual da fila
  - `clear`: limpa a fila (retira todos os elementos)

## Pilhas e filas

### Definições e tipos de dados abstratos

Implementação em lista ligada

Implementação em vector

## Listas biligadas

Comparação entre diferentes tipos de listas ligadas

- `in(v)`
  - Pré-condição: `!isFull()`
  - Pós-condição: `!isEmpty()`
- `out()`
  - Pré-condição: `!isEmpty()`
  - Pós-condição: `!isFull()`
- `peek()`
  - Pré-condição: `!isEmpty()`

## Pilhas e filas

### Definições e tipos de dados abstratos

Implementação em lista ligada

Implementação em vector

## Listas biligadas

Comparação entre diferentes tipos de listas ligadas

- **in(v)**
  - Pré-condição: `!isFull()`
  - Pós-condição: `!isEmpty()`
- **out()**
  - Pré-condição: `!isEmpty()`
  - Pós-condição: `!isFull()`
- **peek()**
  - Pré-condição: `!isEmpty()`

- **in(v)**
  - Pré-condição: `!isFull()`
  - Pós-condição: `!isEmpty()`
- **out()**
  - Pré-condição: `!isEmpty()`
  - Pós-condição: `!isFull()`
- **peek()**
  - Pré-condição: `!isEmpty()`



- **in(v)**
  - Pré-condição: `!isFull()`
  - Pós-condição: `!isEmpty()`
- **out()**
  - Pré-condição: `!isEmpty()`
  - Pós-condição: `!isFull()`
- **peek()**
  - Pré-condição: `!isEmpty()`

- **in(v)**
  - Pré-condição: `!isFull()`
  - Pós-condição: `!isEmpty()`
- **out()**
  - Pré-condição: `!isEmpty()`
  - Pós-condição: `!isFull()`
- **peek()**
  - Pré-condição: `!isEmpty()`

- **in(v)**
  - Pré-condição: `!isFull()`
  - Pós-condição: `!isEmpty()`
- **out()**
  - Pré-condição: `!isEmpty()`
  - Pós-condição: `!isFull()`
- **peek()**
  - Pré-condição: `!isEmpty()`

- **in(v)**
  - Pré-condição: `!isFull()`
  - Pós-condição: `!isEmpty()`
- **out()**
  - Pré-condição: `!isEmpty()`
  - Pós-condição: `!isFull()`
- **peek()**
  - Pré-condição: `!isEmpty()`

- **in(v)**
  - Pré-condição: `!isFull()`
  - Pós-condição: `!isEmpty()`
- **out()**
  - Pré-condição: `!isEmpty()`
  - Pós-condição: `!isFull()`
- **peek()**
  - Pré-condição: `!isEmpty()`

- **in(v)**
  - Pré-condição: `!isFull()`
  - Pós-condição: `!isEmpty()`
- **out()**
  - Pré-condição: `!isEmpty()`
  - Pós-condição: `!isFull()`
- **peek()**
  - Pré-condição: `!isEmpty()`

## Pilhas e filas

Definições e tipos de dados  
abstratos

## Implementação em lista ligada

Implementação em vector

## Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas

- Numa aula anterior, estudámos as listas ligadas.
- Comparando com os vectores, vimos que:
  - A grande vantagem das listas ligadas é serem estruturas de dados dinâmicas, portanto sem limitação na sua capacidade.
  - A grande desvantagem das listas ligadas é não facilitarem o acesso direto a cada elemento.
- No caso particular das pilhas e das filas:
  - Pode ser difícil prever o número de elementos.
  - Não há necessidade de aceder a elementos abaixo do topo da pilha.
  - Não há necessidade de aceder a elementos no meio da fila.
- Assim, em geral, a implementação de pilhas e filas em lista ligada é vantajosa, quando comparada com a implementação em vector.

- Numa aula anterior, estudámos as listas ligadas.
- Comparando com os vectores, vimos que:
  - A grande vantagem das listas ligadas é serem estruturas de dados dinâmicas, portanto sem limitação na sua capacidade.
  - A grande desvantagem das listas ligadas é não facilitarem o acesso direto a cada elemento.
- No caso particular das pilhas e das filas:
  - Pode ser difícil prever o número de elementos.
  - Não há necessidade de aceder a elementos abaixo do topo da pilha.
  - Não há necessidade de aceder a elementos no meio da fila.
- Assim, em geral, a implementação de pilhas e filas em lista ligada é vantajosa, quando comparada com a implementação em vector.



- Numa aula anterior, estudámos as listas ligadas.
- Comparando com os vectores, vimos que:
  - A grande vantagem das listas ligadas é serem estruturas de dados dinâmicas, portanto sem limitação na sua capacidade.
  - A grande desvantagem das listas ligadas é não facilitarem o acesso direto a cada elemento.
- No caso particular das pilhas e das filas:
  - Pode ser difícil prever o número de elementos.
  - Não há necessidade de aceder a elementos abaixo do topo da pilha.
  - Não há necessidade de aceder a elementos no meio da fila.
- Assim, em geral, a implementação de pilhas e filas em lista ligada é vantajosa, quando comparada com a implementação em vector.

- Numa aula anterior, estudámos as listas ligadas.
- Comparando com os vectores, vimos que:
  - A grande vantagem das listas ligadas é serem estruturas de dados dinâmicas, portanto sem limitação na sua capacidade.
  - A grande desvantagem das listas ligadas é não facilitarem o acesso direto a cada elemento.
- No caso particular das pilhas e das filas:
  - Pode ser difícil prever o número de elementos.
  - Não há necessidade de aceder a elementos abaixo do topo da pilha.
  - Não há necessidade de aceder a elementos no meio da fila.
- Assim, em geral, a implementação de pilhas e filas em lista ligada é vantajosa, quando comparada com a implementação em vector.

- Numa aula anterior, estudámos as listas ligadas.
- Comparando com os vectores, vimos que:
  - A grande vantagem das listas ligadas é serem estruturas de dados dinâmicas, portanto sem limitação na sua capacidade.
  - A grande desvantagem das listas ligadas é não facilitarem o acesso direto a cada elemento.
- No caso particular das pilhas e das filas:
  - Pode ser difícil prever o número de elementos.
  - Não há necessidade de aceder a elementos abaixo do topo da pilha.
  - Não há necessidade de aceder a elementos no meio da fila.
- Assim, em geral, a implementação de pilhas e filas em lista ligada é vantajosa, quando comparada com a implementação em vector.

# Relembrando: lista ligada simples

## Pilhas e filas

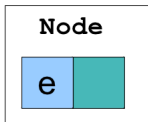
Definições e tipos de dados  
abstratos

## Implementação em lista ligada

Implementação em vector

## Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas



```
class Node
{
    int e;
    Node next;
}
```



## Relembrando: lista ligada com dupla entrada

### Pilhas e filas

Definições e tipos de dados  
abstratos

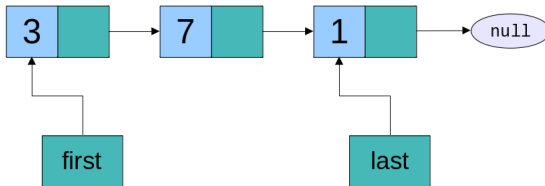
### Implementação em lista ligada

Implementação em vector

### Listas biligadas

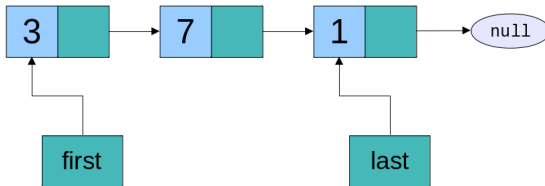
Comparação entre  
diferentes tipos de  
listas ligadas

- A lista possui acesso direto ao primeiro e último elementos.
- É simples acrescentar elementos no início e no fim da lista.
- É simples remover elementos do início da lista.



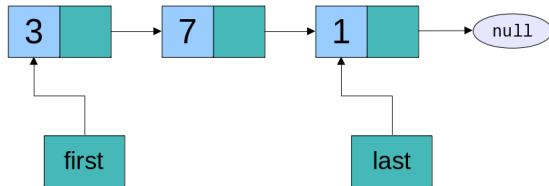
## Relembrando: lista ligada com dupla entrada

- A lista possui acesso direto ao primeiro e último elementos.
- É simples acrescentar elementos no início e no fim da lista.
- É simples remover elementos do início da lista.



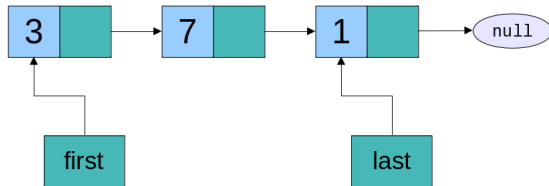
## Relembrando: lista ligada com dupla entrada

- A lista possui acesso direto ao primeiro e último elementos.
- É simples acrescentar elementos no início e no fim da lista.
- É simples remover elementos do início da lista.



## Relembrando: lista ligada com dupla entrada

- A lista possui acesso direto ao primeiro e último elementos.
- É simples acrescentar elementos no início e no fim da lista.
- É simples remover elementos do início da lista.





# Relembrando: lista ligada - tipo de dados abstrato

## Pilhas e filas

Definições e tipos de dados abstratos

## Implementação em lista ligada

Implementação em vector

## Listas biligadas

Comparação entre diferentes tipos de listas ligadas

- Nome do módulo:

  - `LinkedList`

- Serviços:

  - `addFront()`: insere um elemento no início da lista

  - `addLast()`: insere um elemento no fim da lista

  - `FrFirst()`: retorna o primeiro elemento da lista

  - `FrLast()`: retorna o último elemento da lista

  - `removeFrFirst()`: retira o elemento no início da lista

  - `FrSize()`: retorna o tamanho actual da lista

  - `isEmpty()`: verifica se a lista está vazia

  - `clear()`: limpa a lista (retirando todos os elementos)

# Relembrando: lista ligada - tipo de dados abstrato

- Nome do módulo:

- `LinkedList`

- Serviços:

- `addFirst`: insere um elemento no início da lista
  - `addLast`: insere um elemento no fim da lista
  - `first`: retorna o primeiro elemento da lista
  - `last`: retorna o último elemento lista
  - `removeFirst`: retira o elemento no início da lista
  - `size`: retorna a dimensão actual da lista
  - `isEmpty`: verifica se a lista está vazia
  - `clear`: limpa a lista (remove todos os elementos)

- Nome do módulo:
  - LinkedList
- Serviços:
  - `addFirst`: insere um elemento no início da lista
  - `addLast`: insere um elemento no fim da lista
  - `first`: retorna o primeiro elemento da lista
  - `last`: retorna o último elemento lista
  - `removeFirst`: retira o elemento no início da lista
  - `size`: retorna a dimensão actual da lista
  - `isEmpty`: verifica se a lista está vazia
  - `clear`: limpa a lista (remove todos os elementos)

- Nome do módulo:
  - `LinkedList`
- Serviços:
  - `addFirst`: insere um elemento no início da lista
  - `addLast`: insere um elemento no fim da lista
  - `first`: retorna o primeiro elemento da lista
  - `last`: retorna o último elemento lista
  - `removeFirst`: retira o elemento no início da lista
  - `size`: retorna a dimensão actual da lista
  - `isEmpty`: verifica se a lista está vazia
  - `clear`: limpa a lista (remove todos os elementos)

- Nome do módulo:
  - `LinkedList`
- Serviços:
  - `addFirst`: insere um elemento no início da lista
  - `addLast`: insere um elemento no fim da lista
  - `first`: retorna o primeiro elemento da lista
  - `last`: retorna o último elemento lista
  - `removeFirst`: retira o elemento no início da lista
  - `size`: retorna a dimensão actual da lista
  - `isEmpty`: verifica se a lista está vazia
  - `clear`: limpa a lista (remove todos os elementos)

- Nome do módulo:
  - `LinkedList`
- Serviços:
  - `addFirst`: insere um elemento no início da lista
  - `addLast`: insere um elemento no fim da lista
  - `first`: retorna o primeiro elemento da lista
  - `last`: retorna o último elemento lista
  - `removeFirst`: retira o elemento no início da lista
  - `size`: retorna a dimensão actual da lista
  - `isEmpty`: verifica se a lista está vazia
  - `clear`: limpa a lista (remove todos os elementos)

- Nome do módulo:
  - `LinkedList`
- Serviços:
  - `addFirst`: insere um elemento no início da lista
  - `addLast`: insere um elemento no fim da lista
  - `first`: retorna o primeiro elemento da lista
  - `last`: retorna o último elemento lista
  - `removeFirst`: retira o elemento no início da lista
  - `size`: retorna a dimensão actual da lista
  - `isEmpty`: verifica se a lista está vazia
  - `clear`: limpa a lista (remove todos os elementos)

- Nome do módulo:
  - `LinkedList`
- Serviços:
  - `addFirst`: insere um elemento no início da lista
  - `addLast`: insere um elemento no fim da lista
  - `first`: retorna o primeiro elemento da lista
  - `last`: retorna o último elemento lista
  - `removeFirst`: retira o elemento no início da lista
  - `size`: retorna a dimensão actual da lista
  - `isEmpty`: verifica se a lista está vazia
  - `clear`: limpa a lista (remove todos os elementos)



- Nome do módulo:
  - `LinkedList`
- Serviços:
  - `addFirst`: insere um elemento no início da lista
  - `addLast`: insere um elemento no fim da lista
  - `first`: retorna o primeiro elemento da lista
  - `last`: retorna o último elemento lista
  - `removeFirst`: retira o elemento no início da lista
  - `size`: retorna a dimensão actual da lista
  - `isEmpty`: verifica se a lista está vazia
  - `clear`: limpa a lista (remove todos os elementos)

- Nome do módulo:
  - `LinkedList`
- Serviços:
  - `addFirst`: insere um elemento no início da lista
  - `addLast`: insere um elemento no fim da lista
  - `first`: retorna o primeiro elemento da lista
  - `last`: retorna o último elemento lista
  - `removeFirst`: retira o elemento no início da lista
  - `size`: retorna a dimensão actual da lista
  - `isEmpty`: verifica se a lista está vazia
  - `clear`: limpa a lista (remove todos os elementos)

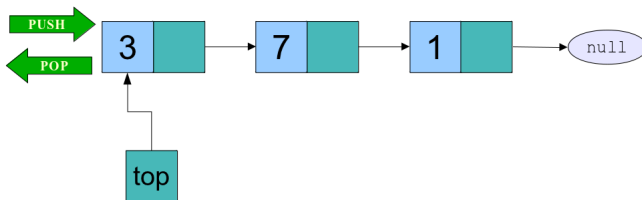
# Relembrando: lista ligada - tipo de dados abstrato

- Nome do módulo:
  - `LinkedList`
- Serviços:
  - `addFirst`: insere um elemento no início da lista
  - `addLast`: insere um elemento no fim da lista
  - `first`: retorna o primeiro elemento da lista
  - `last`: retorna o último elemento lista
  - `removeFirst`: retira o elemento no início da lista
  - `size`: retorna a dimensão actual da lista
  - `isEmpty`: verifica se a lista está vazia
  - `clear`: limpa a lista (remove todos os elementos)

- Nome do módulo:
  - `LinkedList`
- Serviços:
  - `addFirst`: insere um elemento no início da lista
  - `addLast`: insere um elemento no fim da lista
  - `first`: retorna o primeiro elemento da lista
  - `last`: retorna o último elemento lista
  - `removeFirst`: retira o elemento no início da lista
  - `size`: retorna a dimensão actual da lista
  - `isEmpty`: verifica se a lista está vazia
  - `clear`: limpa a lista (remove todos os elementos)

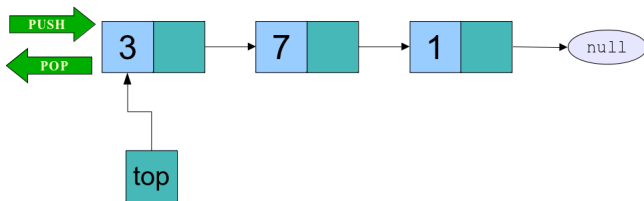
# Pilha: implementação em lista ligada

- Usa uma gestão *LIFO* (*Last In First Out*)
- O último elemento empilhado ( $top$ ) é o primeiro a desempilhar.
- O elemento  $top$  corresponde ao primeiro nó da lista ligada.
- O elemento no topo da pilha fica armazenado no primeiro nó da lista.
- O elemento na base da pilha fica armazenado no último nó da lista.



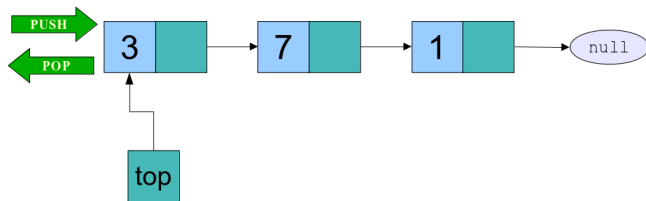
## Pilha: implementação em lista ligada

- Usa uma gestão *LIFO* (*Last In First Out*)
- O último elemento empilhado ( $t_{op}$ ) é o primeiro a desempilhar.
  - Método *push* corresponde a *addFirst* da lista ligada.
  - Método *pop* corresponde a *removeFirst* da lista ligada.
- O elemento no topo da pilha fica armazenado no primeiro nó da lista.
- O elemento na base da pilha fica armazenado no último nó da lista.



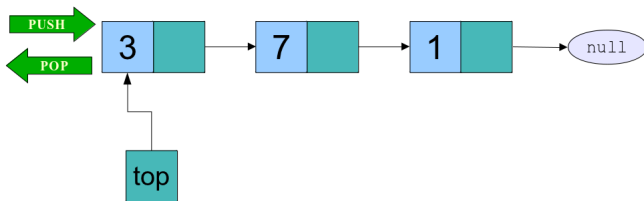
## Pilha: implementação em lista ligada

- Usa uma gestão *LIFO* (*Last In First Out*)
- O último elemento empilhado ( $t_{op}$ ) é o primeiro a desempilhar.
  - Método `push` corresponde a `addFirst` da lista ligada.
  - Método `pop` corresponde a `removeFirst` da lista ligada.
- O elemento no topo da pilha fica armazenado no primeiro nó da lista.
- O elemento na base da pilha fica armazenado no último nó da lista.



## Pilha: implementação em lista ligada

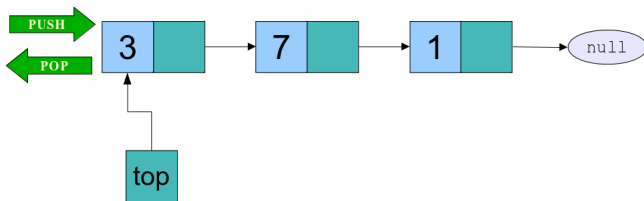
- Usa uma gestão *LIFO* (*Last In First Out*)
- O último elemento empilhado ( $t_{op}$ ) é o primeiro a desempilhar.
  - Método `push` corresponde a `addFirst` da lista ligada.
  - Método `pop` corresponde a `removeFirst` da lista ligada.
- O elemento no topo da pilha fica armazenado no primeiro nó da lista.
- O elemento na base da pilha fica armazenado no último nó da lista.





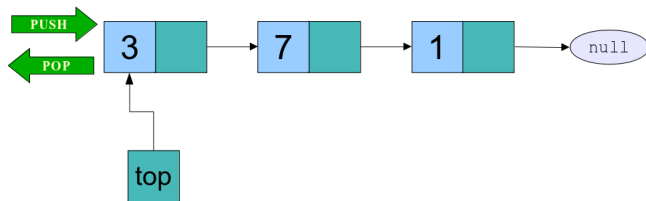
## Pilha: implementação em lista ligada

- Usa uma gestão *LIFO* (*Last In First Out*)
- O último elemento empilhado ( $t_{op}$ ) é o primeiro a desempilhar.
  - Método `push` corresponde a `addFirst` da lista ligada.
  - Método `pop` corresponde a `removeFirst` da lista ligada.
- O elemento no topo da pilha fica armazenado no primeiro nó da lista.
- O elemento na base da pilha fica armazenado no último nó da lista.



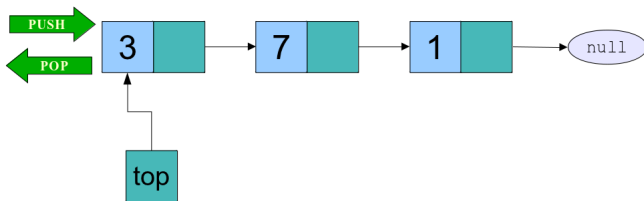
## Pilha: implementação em lista ligada

- Usa uma gestão *LIFO* (*Last In First Out*)
- O último elemento empilhado ( $t_{op}$ ) é o primeiro a desempilhar.
  - Método `push` corresponde a `addFirst` da lista ligada.
  - Método `pop` corresponde a `removeFirst` da lista ligada.
- O elemento no topo da pilha fica armazenado no primeiro nó da lista.
- O elemento na base da pilha fica armazenado no último nó da lista.



## Pilha: implementação em lista ligada

- Usa uma gestão *LIFO* (*Last In First Out*)
- O último elemento empilhado ( $t_{op}$ ) é o primeiro a desempilhar.
  - Método `push` corresponde a `addFirst` da lista ligada.
  - Método `pop` corresponde a `removeFirst` da lista ligada.
- O elemento no topo da pilha fica armazenado no primeiro nó da lista.
- O elemento na base da pilha fica armazenado no último nó da lista.



# Pilha genérica: implementação em lista ligada

## Pilhas e filas

Definições e tipos de dados  
abstratos

## Implementação em lista ligada

Implementação em vector

## Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas

```
public class Stack<E> {  
  
    private LinkedList<E> list = new LinkedList<E>();  
  
    public void push(E element) {  
        list.addFirst(element);  
    }  
  
    public E top() {  
        return list.first();  
    }  
  
    public void pop() {  
        list.removeFirst();  
    }  
  
    public int size() {  
        return list.size();  
    }  
  
    public boolean isEmpty() {  
        return list.isEmpty();  
    }  
  
}
```

# Pilha genérica: implementação em lista ligada

## Pilhas e filas

Definições e tipos de dados  
abstratos

## Implementação em lista ligada

Implementação em vector

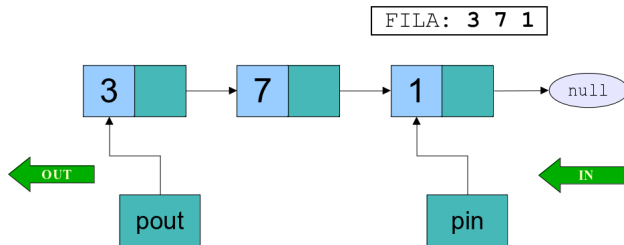
## Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas

```
public class Stack<E> {  
  
    private LinkedList<E> list = new LinkedList<E>();  
  
    public void push(E element) {  
        list.addFirst(element);  
    }  
  
    public E top() {  
        return list.first();  
    }  
  
    public void pop() {  
        list.removeFirst();  
    }  
  
    public int size() {  
        return list.size();  
    }  
  
    public boolean isEmpty() {  
        return list.isEmpty();  
    }  
  
}
```

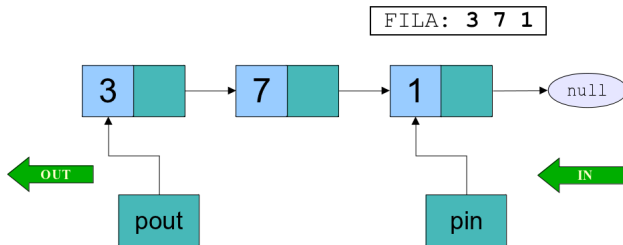
# Fila: implementação em lista ligada

- Usa uma gestão *FIFO* (*First In First Out*).
- O primeiro elemento introduzido é o primeiro a remover, por isso tem que ficar no primeiro nó da lista.
  - elemento que corresponde a remover? 3 da lista ligada.
- O último elemento introduzido fica armazenado no último nó da lista e será o último a ser removido.
  - elemento que corresponde a adicionar? 1 da lista ligada.



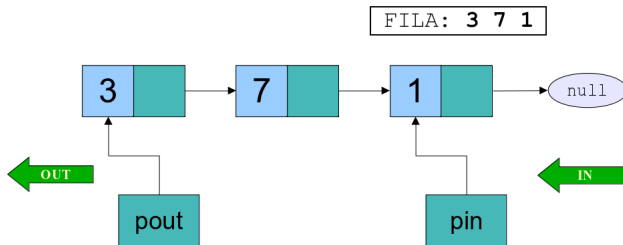
## Fila: implementação em lista ligada

- Usa uma gestão *FIFO* (*First In First Out*).
- O primeiro elemento introduzido é o primeiro a remover, por isso tem que ficar no primeiro nó da lista.
  - Método *out* corresponde a *removeFirst* da lista ligada.
- O último elemento introduzido fica armazenado no último nó da lista e será o último a ser removido.
  - Método *in* corresponde a *addLast* da lista ligada.



## Fila: implementação em lista ligada

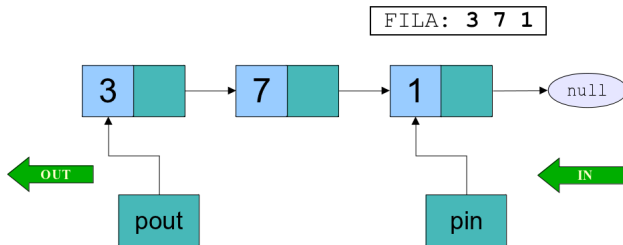
- Usa uma gestão *FIFO* (*First In First Out*).
- O primeiro elemento introduzido é o primeiro a remover, por isso tem que ficar no primeiro nó da lista.
  - Método `out` corresponde a `removeFirst` da lista ligada.
- O último elemento introduzido fica armazenado no último nó da lista e será o último a ser removido.
  - Método `in` corresponde a `addLast` da lista ligada.





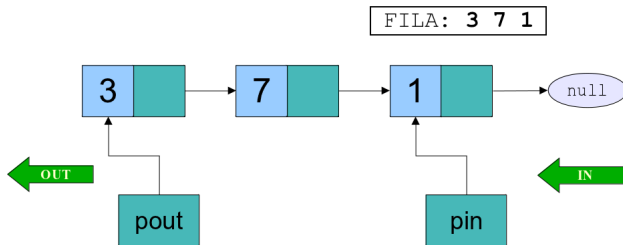
## Fila: implementação em lista ligada

- Usa uma gestão *FIFO* (*First In First Out*).
- O primeiro elemento introduzido é o primeiro a remover, por isso tem que ficar no primeiro nó da lista.
  - Método `out` corresponde a `removeFirst` da lista ligada.
- O último elemento introduzido fica armazenado no último nó da lista e será o último a ser removido.
  - Método `in` corresponde a `addLast` da lista ligada.



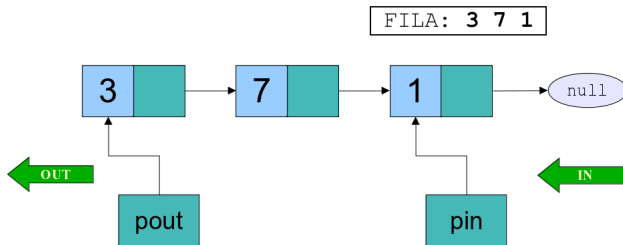
## Fila: implementação em lista ligada

- Usa uma gestão *FIFO* (*First In First Out*).
- O primeiro elemento introduzido é o primeiro a remover, por isso tem que ficar no primeiro nó da lista.
  - Método `out` corresponde a `removeFirst` da lista ligada.
- O último elemento introduzido fica armazenado no último nó da lista e será o último a ser removido.
  - Método `in` corresponde a `addLast` da lista ligada.



## Fila: implementação em lista ligada

- Usa uma gestão *FIFO* (*First In First Out*).
- O primeiro elemento introduzido é o primeiro a remover, por isso tem que ficar no primeiro nó da lista.
  - Método `out` corresponde a `removeFirst` da lista ligada.
- O último elemento introduzido fica armazenado no último nó da lista e será o último a ser removido.
  - Método `in` corresponde a `addLast` da lista ligada.



# Fila genérica: implementação em lista ligada

## Pilhas e filas

Definições e tipos de dados  
abstratos

## Implementação em lista ligada

Implementação em vector

## Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas

```
public class Queue<E> {  
  
    private LinkedList<E> list = new LinkedList<E>();  
  
    public void in(E element) {  
        list.addLast(element);  
    }  
  
    public E peek() {  
        return list.first();  
    }  
  
    public void out() {  
        list.removeFirst();  
    }  
  
    public int size() {  
        return list.size();  
    }  
  
    public boolean isEmpty() {  
        return list.isEmpty();  
    }  
  
}
```

# Fila genérica: implementação em lista ligada

## Pilhas e filas

Definições e tipos de dados  
abstratos

Implementação em lista  
ligada

Implementação em vector

## Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas

```
public class Queue<E> {  
  
    private LinkedList<E> list = new LinkedList<E>();  
  
    public void in(E element) {  
        list.addLast(element);  
    }  
  
    public E peek() {  
        return list.first();  
    }  
  
    public void out() {  
        list.removeFirst();  
    }  
  
    public int size() {  
        return list.size();  
    }  
  
    public boolean isEmpty() {  
        return list.isEmpty();  
    }  
  
}
```

# Pilha: implementação em vector

## Pilhas e filas

Definições e tipos de dados  
abstratos

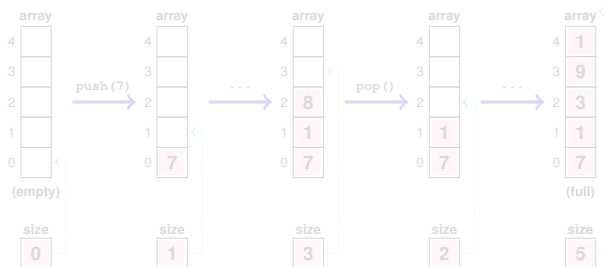
Implementação em lista  
ligada

Implementação em vector

## Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas

- Precisamos de dois atributos:
  - O vector que armazena os elementos
  - O número de elementos, que funciona também como índice da primeira posição livre



## Pilhas e filas

Definições e tipos de dados  
abstratos

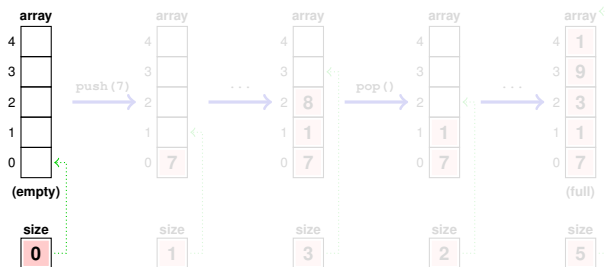
Implementação em lista  
ligada

Implementação em vector

## Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas

- Precisamos de dois atributos:
  - O vector que armazena os elementos
  - O número de elementos, que funciona também como índice da primeira posição livre



## Pilhas e filas

Definições e tipos de dados  
abstratos

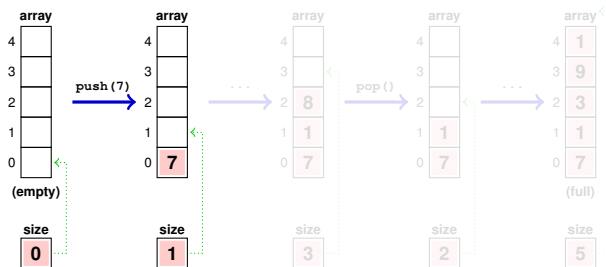
Implementação em lista  
ligada

Implementação em vector

## Listas biligadas

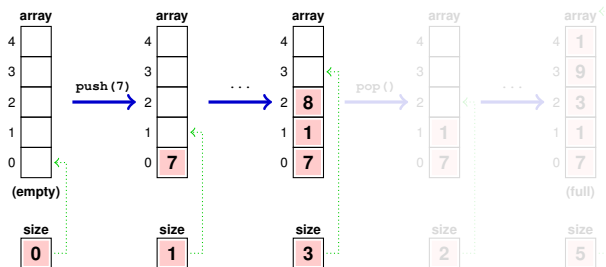
Comparação entre  
diferentes tipos de  
listas ligadas

- Precisamos de dois atributos:
  - O vector que armazena os elementos
  - O número de elementos, que funciona também como índice da primeira posição livre

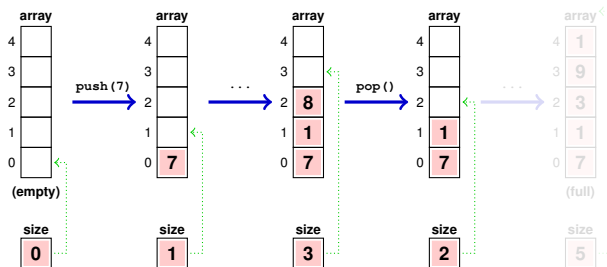




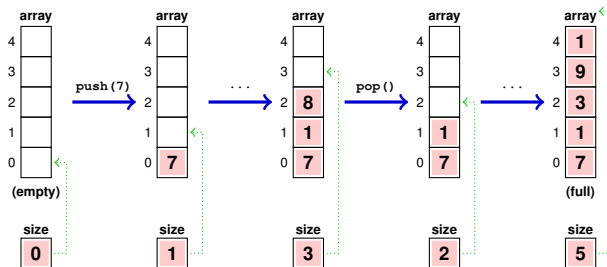
- Precisamos de dois atributos:
  - O vector que armazena os elementos
  - O número de elementos, que funciona também como índice da primeira posição livre



- Precisamos de dois atributos:
  - O vector que armazena os elementos
  - O número de elementos, que funciona também como índice da primeira posição livre



- Precisamos de dois atributos:
  - O vector que armazena os elementos
  - O número de elementos, que funciona também como índice da primeira posição livre



# Pilha genérica: implementação em vector

## Pilhas e filas

Definições e tipos de dados  
abstratos

Implementação em lista  
ligada

Implementação em vector

## Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas

```
public class Stack<E> {  
  
    private E[] array;  
    private int size;  
  
    public Stack(int maxSize) {  
        assert maxSize >= 0;  
        array = (E[]) new Object[maxSize];  
        size = 0;  
    }  
  
    public void push(E e) {  
        assert !isFull();  
        array[size] = e;  
        size++;  
        assert !isEmpty() && top() == e;  
    }  
  
    public void pop() {  
        assert !isEmpty();  
        size--;  
        assert !isFull();  
    }  
}
```

```
    public E top() {  
        assert !isEmpty();  
        return array[size-1];  
    }  
  
    public boolean isEmpty() {  
        return size == 0;  
    }  
  
    public boolean isFull() {  
        return size == array.length;  
    }  
  
    public int size() {  
        return size;  
    }  
  
    public void clear() {  
        size = 0;  
        assert isEmpty();  
    }  
}
```

# Pilha genérica: implementação em vector

## Pilhas e filas

Definições e tipos de dados  
abstratos

Implementação em lista  
ligada

Implementação em vector

## Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas

```
public class Stack<E> {  
  
    private E[] array;  
    private int size;  
  
    public Stack(int maxSize) {  
        assert maxSize >= 0;  
        array = (E[]) new Object[maxSize];  
        size = 0;  
    }  
  
    public void push(E e) {  
        assert !isFull();  
        array[size] = e;  
        size++;  
        assert !isEmpty() && top() == e;  
    }  
  
    public void pop() {  
        assert !isEmpty();  
        size--;  
        assert !isFull();  
    }  
}
```

```
    public E top() {  
        assert !isEmpty();  
        return array[size-1];  
    }  
  
    public boolean isEmpty() {  
        return size == 0;  
    }  
  
    public boolean isFull() {  
        return size == array.length;  
    }  
  
    public int size() {  
        return size;  
    }  
  
    public void clear() {  
        size = 0;  
        assert isEmpty();  
    }  
}
```

## Pilhas e filas

Definições e tipos de dados  
abstratos

Implementação em lista  
ligada

Implementação em vector

## Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas

- A forma mais eficiente de implementar é uma estrutura conhecida como *buffer circular*.
- Requer 4 atributos:
  - O vector que armazena os elementos.
  - O número de elementos.
  - O índice do próximo elemento a ser retirado (*cabeça da fila*).
  - O índice do próximo elemento a ser ocupado (*cauda da fila*).
- Sempre que se insere ou retira um elemento, incrementa-se o índice respetivo em *aritmética modular*.
  - Ou seja, quando o índice atinge o limite, é reposto a zero.

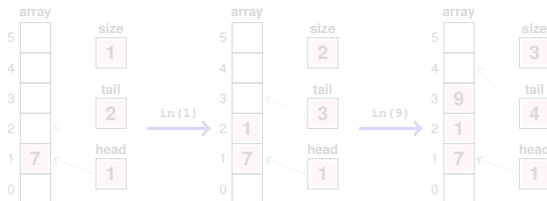
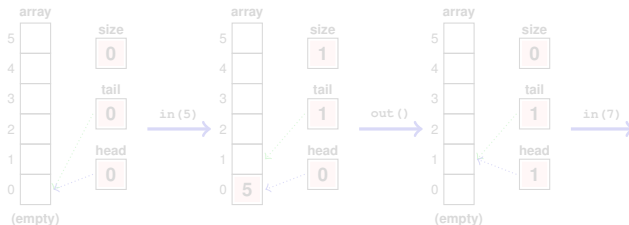
- A forma mais eficiente de implementar é uma estrutura conhecida como *buffer circular*.
- Requer 4 atributos:
  - O vector que armazena os elementos.
  - O número de elementos.
  - O índice do próximo elemento a ser retirado (*cabeça* da fila).
  - O índice do próximo elemento a ser ocupado (*cauda* da fila).
- Sempre que se insere ou retira um elemento, incrementa-se o índice respetivo em *aritmética modular*.
  - Ou seja, quando o índice atinge o limite, é reposto a zero.

- A forma mais eficiente de implementar é uma estrutura conhecida como *buffer circular*.
- Requer 4 atributos:
  - O vector que armazena os elementos.
  - O número de elementos.
  - O índice do próximo elemento a ser retirado (*cabeça* da fila).
  - O índice do próximo elemento a ser ocupado (*cauda* da fila).
- Sempre que se insere ou retira um elemento, incrementa-se o índice respetivo em *aritmética modular*.
  - Ou seja, quando o índice atinge o limite, é reposto a zero.



- A forma mais eficiente de implementar é uma estrutura conhecida como *buffer circular*.
- Requer 4 atributos:
  - O vector que armazena os elementos.
  - O número de elementos.
  - O índice do próximo elemento a ser retirado (*cabeça* da fila).
  - O índice do próximo elemento a ser ocupado (*cauda* da fila).
- Sempre que se insere ou retira um elemento, incrementa-se o índice respetivo em *aritmética modular*.
  - Ou seja, quando o índice atinge o limite, é reposto a zero.

# Fila: exemplo



## Pilhas e filas

Definições e tipos de dados abstratos

Implementação em lista ligada

Implementação em vector

## Listas biligadas

Comparação entre diferentes tipos de listas ligadas

# Fila: exemplo

## Pilhas e filas

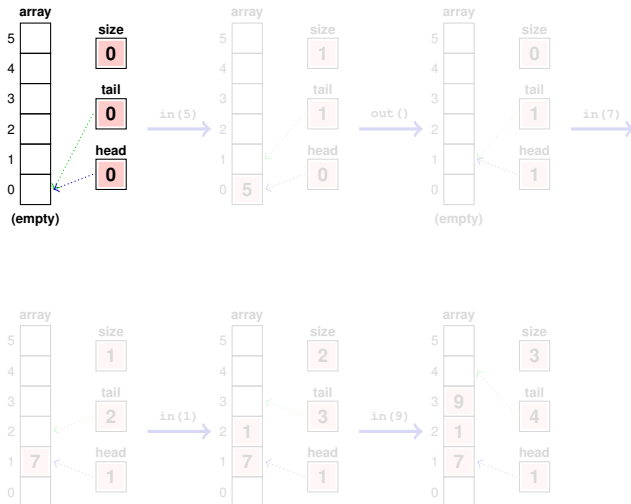
Definições e tipos de dados abstratos

Implementação em lista ligada

Implementação em vector

## Listas biligadas

Comparação entre diferentes tipos de listas ligadas



# Fila: exemplo

## Pilhas e filas

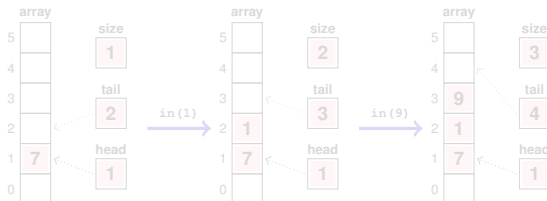
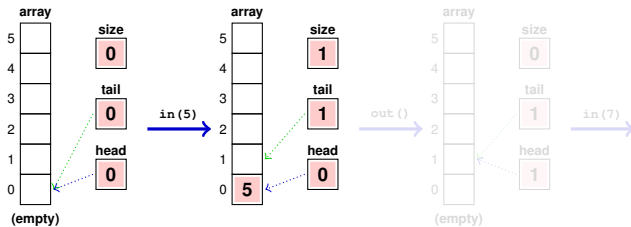
Definições e tipos de dados  
abstratos

Implementação em lista  
ligada

Implementação em vector

## Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas



# Fila: exemplo

## Pilhas e filas

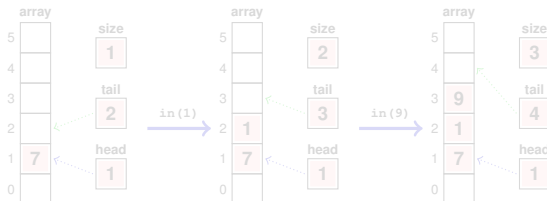
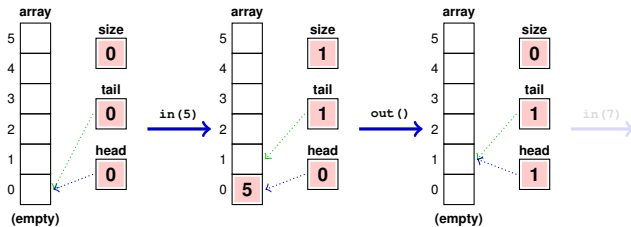
Definições e tipos de dados abstratos

Implementação em lista ligada

Implementação em vector

## Listas biligadas

Comparação entre diferentes tipos de listas ligadas



# Fila: exemplo

## Pilhas e filas

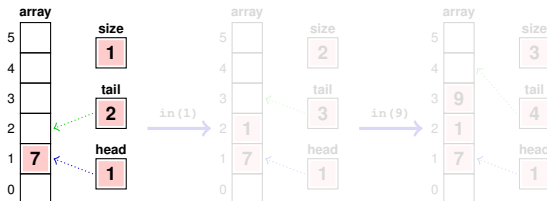
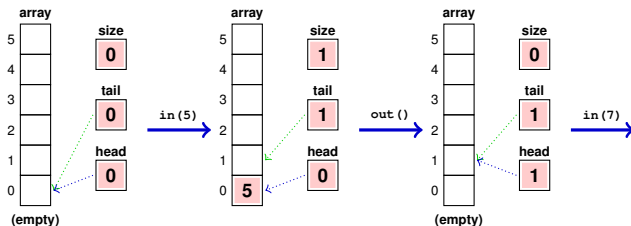
Definições e tipos de dados abstratos

Implementação em lista ligada

Implementação em vector

## Listas biligadas

Comparação entre diferentes tipos de listas ligadas



# Fila: exemplo

## Pilhas e filas

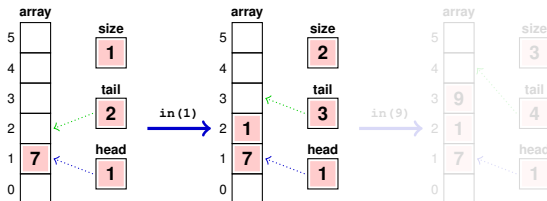
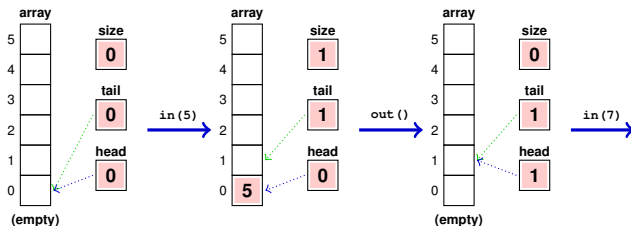
Definições e tipos de dados  
abstratos

Implementação em lista  
ligada

Implementação em vector

## Listas bilgadas

Comparação entre  
diferentes tipos de  
listas ligadas



# Fila: exemplo

## Pilhas e filas

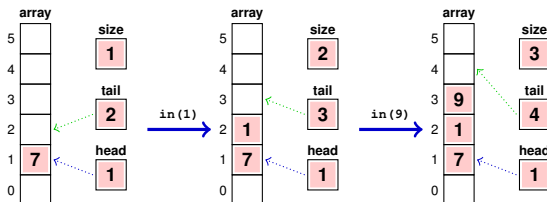
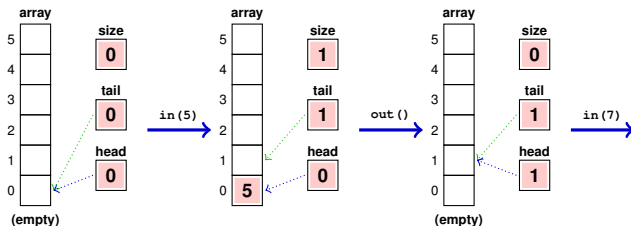
Definições e tipos de dados  
abstratos

Implementação em lista  
ligada

Implementação em vector

## Listas bilgadas

Comparação entre  
diferentes tipos de  
listas ligadas





# Fila: exemplo - gestão circular

## Pilhas e filas

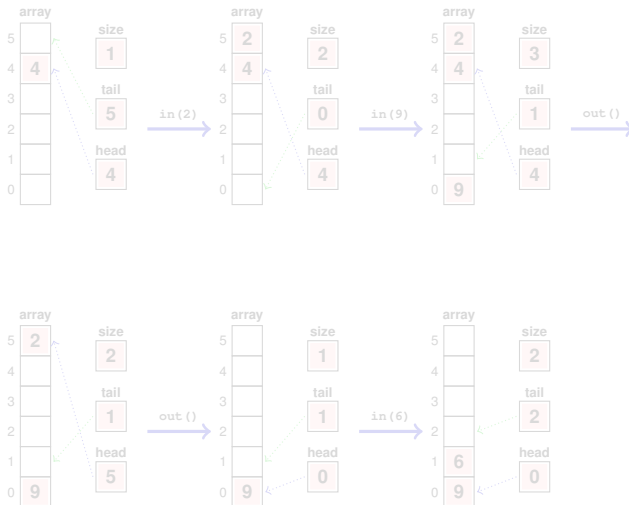
Definições e tipos de dados  
abstratos

Implementação em lista  
ligada

Implementação em vector

## Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas



# Fila: exemplo - gestão circular

## Pilhas e filas

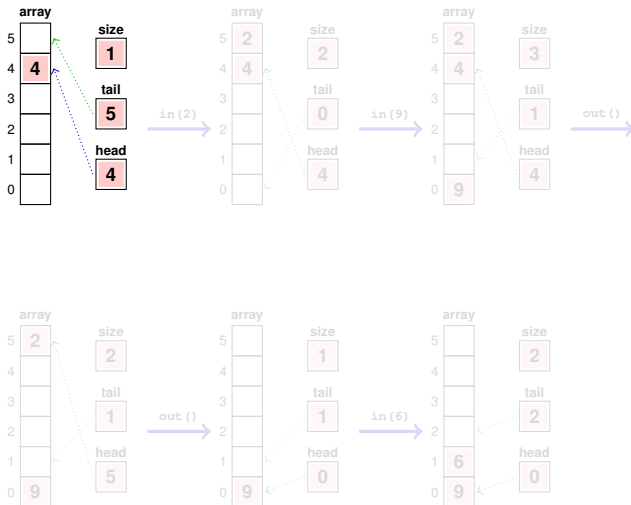
Definições e tipos de dados  
abstratos

Implementação em lista  
ligada

Implementação em vector

## Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas



# Fila: exemplo - gestão circular

## Pilhas e filas

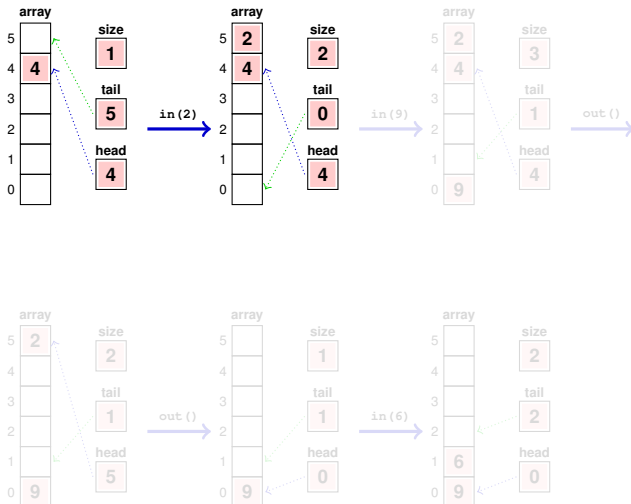
Definições e tipos de dados  
abstratos

Implementação em lista  
ligada

Implementação em vector

## Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas



# Fila: exemplo - gestão circular

## Pilhas e filas

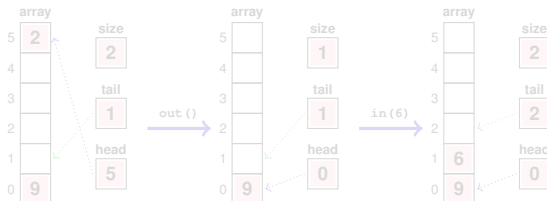
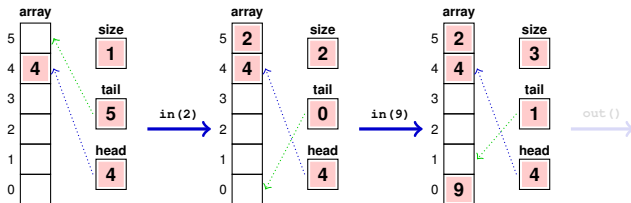
Definições e tipos de dados abstratos

Implementação em lista ligada

Implementação em vector

## Listas biligadas

Comparação entre diferentes tipos de listas ligadas



# Fila: exemplo - gestão circular

## Pilhas e filas

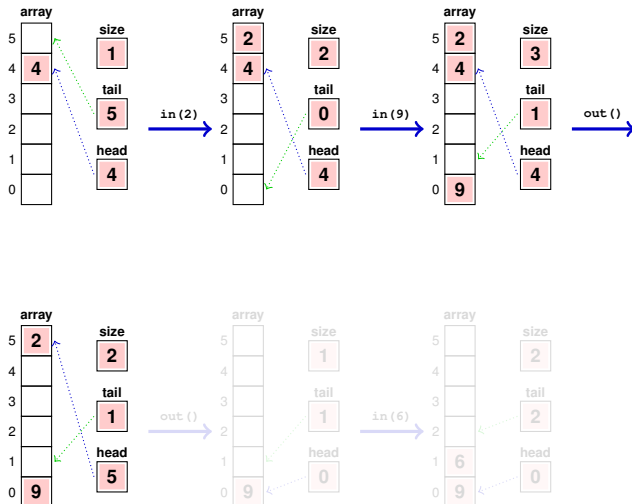
Definições e tipos de dados  
abstratos

Implementação em lista  
ligada

Implementação em vector

## Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas



# Fila: exemplo - gestão circular

## Pilhas e filas

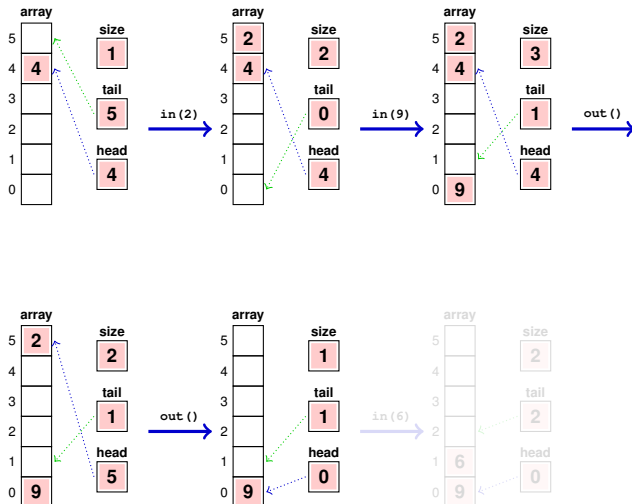
Definições e tipos de dados abstratos

Implementação em lista ligada

Implementação em vector

## Listas biligadas

Comparação entre diferentes tipos de listas ligadas



# Fila: exemplo - gestão circular

## Pilhas e filas

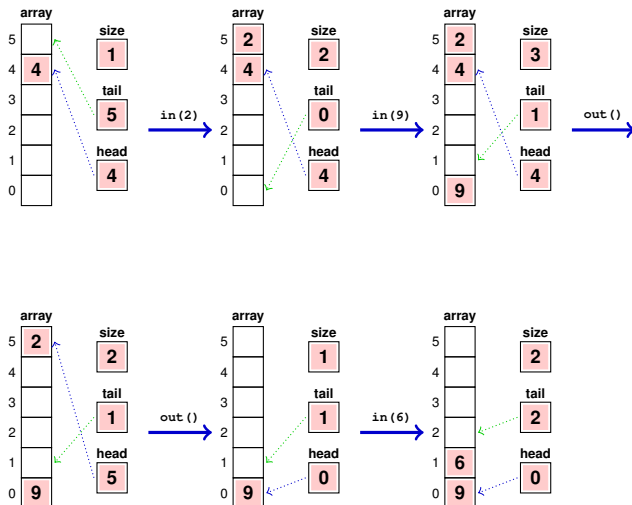
Definições e tipos de dados abstratos

Implementação em lista ligada

Implementação em vector

## Listas biligadas

Comparação entre diferentes tipos de listas ligadas



# Fila: exemplo - empty/full

## Pilhas e filas

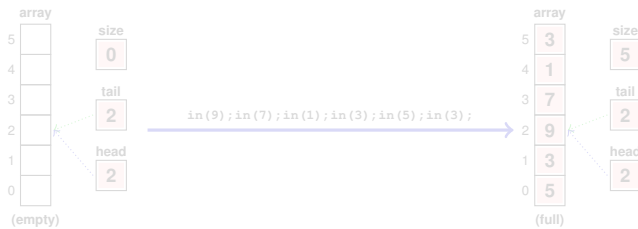
Definições e tipos de dados abstratos

Implementação em lista ligada

Implementação em vector

## Listas biligadas

Comparação entre diferentes tipos de listas ligadas





# Fila: exemplo - empty/full

## Pilhas e filas

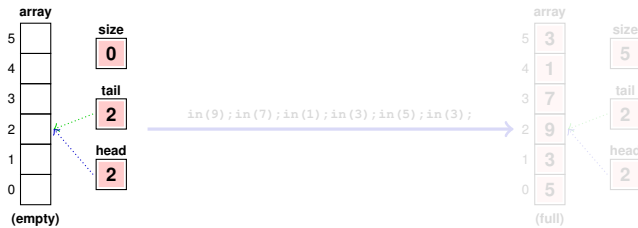
Definições e tipos de dados abstratos

Implementação em lista ligada

Implementação em vector

## Listas biligadas

Comparação entre diferentes tipos de listas ligadas



## Fila: exemplo - empty/full

### Pilhas e filas

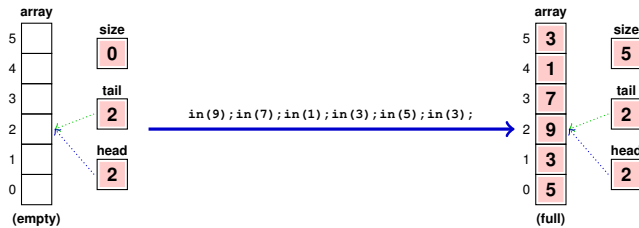
Definições e tipos de dados abstratos

Implementação em lista ligada

Implementação em vector

### Listas biligadas

Comparação entre diferentes tipos de listas ligadas



# Fila genérica: implementação em vector

## Pilhas e filas

Definições e tipos de dados  
abstratos

Implementação em lista  
ligada

Implementação em vector

## Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas

```
public class Queue<E> {  
  
    private E[] array;  
    private int size;  
    private int head;  
    private int tail;  
  
    public Queue(int maxSize) {  
        assert maxSize >= 0;  
  
        array = (T[]) new Object[maxSize];  
        size = head = tail = 0;  
    }  
  
    public void in(E e) {  
        assert !isFull();  
        array[tail] = e;  
        tail = nextPosition(tail);  
        size++;  
    }  
  
    public void out() {  
        assert !isEmpty();  
        head = nextPosition(head);  
        size--;  
    }  
}
```

```
    public E peek() {  
        assert !isEmpty();  
        return array[head];  
    }  
  
    public int size() {  
        return size;  
    }  
  
    public boolean isEmpty() {  
        return size == 0;  
    }  
  
    public boolean isFull() {  
        return size == array.length;  
    }  
  
    public void clear() {  
        head = tail = size = 0;  
    }  
  
    private int nextPosition(int p) {  
        return (p + 1) % array.length;  
    }  
}
```

# Fila genérica: implementação em vector

## Pilhas e filas

Definições e tipos de dados  
abstratos

Implementação em lista  
ligada

Implementação em vector

## Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas

```
public class Queue<E> {  
  
    private E[] array;  
    private int size;  
    private int head;  
    private int tail;  
  
    public Queue(int maxSize) {  
        assert maxSize >= 0;  
  
        array = (T[]) new Object[maxSize];  
        size = head = tail = 0;  
    }  
  
    public void in(E e) {  
        assert !isFull();  
        array[tail] = e;  
        tail = nextPosition(tail);  
        size++;  
    }  
  
    public void out() {  
        assert !isEmpty();  
        head = nextPosition(head);  
        size--;  
    }  
}
```

```
    public E peek() {  
        assert !isEmpty();  
        return array[head];  
    }  
  
    public int size() {  
        return size;  
    }  
  
    public boolean isEmpty() {  
        return size == 0;  
    }  
  
    public boolean isFull() {  
        return size == array.length;  
    }  
  
    public void clear() {  
        head = tail = size = 0;  
    }  
  
    private int nextPosition(int p) {  
        return (p + 1) % array.length;  
    }  
}
```

# Correspondência entre listas, pilhas e filas

## Pilhas e filas

Definições e tipos de dados abstratos

Implementação em lista ligada

Implementação em vector

## Listas biligadas

Comparação entre diferentes tipos de listas ligadas

Lista	descrição	Pilha	Fila
addLast	acrescenta um elemento no fim da lista	-	in
addFirst	acrescenta um elemento no início da lista	push	-
first	devolve o primeiro elemento da lista	top	peek
removeFirst	remove o primeiro elemento da lista	pop	out

- Os tipos de dados abstratos das pilhas e filas correspondem a subconjuntos do tipo de dados abstrato da lista ligada.
- Podemos dizer que os tipos de dados abstratos das pilhas e filas são *açúcar sintático* para certos perfis de utilização das listas.

## Pilhas e filas

Definições e tipos de dados abstratos

Implementação em lista ligada

Implementação em vector

## Listas biligadas

Comparação entre diferentes tipos de listas ligadas

Lista	descrição	Pilha	Fila
addLast	acrescenta um elemento no fim da lista	-	in
addFirst	acrescenta um elemento no início da lista	push	-
first	devolve o primeiro elemento da lista	top	peek
removeFirst	remove o primeiro elemento da lista	pop	out

- Os tipos de dados abstratos das pilhas e filas correspondem a subconjuntos do tipo de dados abstrato da lista ligada.
- Podemos dizer que os tipos de dados abstratos das pilhas e filas são *açúcar sintático* para certos perfis de utilização das listas.

## Pilhas e filas

Definições e tipos de dados abstratos

Implementação em lista ligada

Implementação em vector

## Listas biligadas

Comparação entre diferentes tipos de listas ligadas

- Implementação em lista ligada:
  - Todos os métodos do tipo de dados abstrato têm complexidade constante ( $O(1)$ ).
- Implementação em vector com dimensão fixa:
  - Todos os métodos do tipo de dados abstrato têm complexidade constante ( $O(1)$ ).
- Implementação em vector com re-dimensionamento:
  - Sempre que a pilha ou fila cresce, temos que criar um novo vector e transferir a informação para esse vector.
  - Mesmo assim, a operação push/pop tem a sua complexidade linear ( $O(n)$ ).
  - Os restantes métodos do tipo de dados abstrato têm complexidade constante ( $O(1)$ ).

## Pilhas e filas

Definições e tipos de dados abstratos

Implementação em lista ligada

Implementação em vector

## Listas biligadas

Comparação entre diferentes tipos de listas ligadas

- Implementação em lista ligada:
  - Todos os métodos do tipo de dados abstrato têm complexidade constante ( $O(1)$ ).
- Implementação em vector com dimensão fixa:
  - Todos os métodos do tipo de dados abstrato têm complexidade constante ( $O(1)$ ).
- Implementação em vector com re-dimensionamento:
  - Sempre que a pilha ou fila enche, temos que criar um novo vector e transferir a informação para esse vector.
  - Nesses casos, a operação `push` passa a ter complexidade linear ( $O(n)$ ).
  - Os restantes métodos do tipo de dados abstrato têm complexidade constante ( $O(1)$ ).



## Pilhas e filas

Definições e tipos de dados abstratos

Implementação em lista ligada

Implementação em vector

## Listas biligadas

Comparação entre diferentes tipos de listas ligadas

- Implementação em lista ligada:
  - Todos os métodos do tipo de dados abstrato têm complexidade constante ( $O(1)$ ).
- Implementação em vector com dimensão fixa:
  - Todos os métodos do tipo de dados abstrato têm complexidade constante ( $O(1)$ ).
- Implementação em vector com re-dimensionamento:
  - Sempre que a pilha ou fila enche, temos que criar um novo vector e transferir a informação para esse vector.
  - Nesses casos, a operação `push` passa a ter complexidade linear ( $O(n)$ ).
  - Os restantes métodos do tipo de dados abstrato têm complexidade constante ( $O(1)$ ).

## Pilhas e filas

Definições e tipos de dados abstratos

Implementação em lista ligada

Implementação em vector

## Listas biligadas

Comparação entre diferentes tipos de listas ligadas

- Implementação em lista ligada:
  - Todos os métodos do tipo de dados abstrato têm complexidade constante ( $O(1)$ ).
- Implementação em vector com dimensão fixa:
  - Todos os métodos do tipo de dados abstrato têm complexidade constante ( $O(1)$ ).
- Implementação em vector com re-dimensionamento:
  - Sempre que a pilha ou fila enche, temos que criar um novo vector e transferir a informação para esse vector.
  - Nesses casos, a operação push passa a ter complexidade linear ( $O(n)$ ).
  - Os restantes métodos do tipo de dados abstrato têm complexidade constante ( $O(1)$ ).

## Pilhas e filas

Definições e tipos de dados abstratos

Implementação em lista ligada

Implementação em vector

## Listas biligadas

Comparação entre diferentes tipos de listas ligadas

- Implementação em lista ligada:
  - Todos os métodos do tipo de dados abstrato têm complexidade constante ( $O(1)$ ).
- Implementação em vector com dimensão fixa:
  - Todos os métodos do tipo de dados abstrato têm complexidade constante ( $O(1)$ ).
- Implementação em vector com re-dimensionamento:
  - Sempre que a pilha ou fila enche, temos que criar um novo vector e transferir a informação para esse vector.
  - Nesses casos, a operação push passa a ter complexidade linear ( $O(n)$ ).
  - Os restantes métodos do tipo de dados abstrato têm complexidade constante ( $O(1)$ ).

## Pilhas e filas

Definições e tipos de dados abstratos

Implementação em lista ligada

Implementação em vector

## Listas biligadas

Comparação entre diferentes tipos de listas ligadas

- Implementação em lista ligada:
  - Todos os métodos do tipo de dados abstrato têm complexidade constante ( $O(1)$ ).
- Implementação em vector com dimensão fixa:
  - Todos os métodos do tipo de dados abstrato têm complexidade constante ( $O(1)$ ).
- Implementação em vector com re-dimensionamento:
  - Sempre que a pilha ou fila enche, temos que criar um novo vector e transferir a informação para esse vector.
  - Nesses casos, a operação `push` passa a ter complexidade linear ( $O(n)$ ).
  - Os restantes métodos do tipo de dados abstrato têm complexidade constante ( $O(1)$ ).

## Pilhas e filas

Definições e tipos de dados abstratos

Implementação em lista ligada

Implementação em vector

## Listas biligadas

Comparação entre diferentes tipos de listas ligadas

- Implementação em lista ligada:
  - Todos os métodos do tipo de dados abstrato têm complexidade constante ( $O(1)$ ).
- Implementação em vector com dimensão fixa:
  - Todos os métodos do tipo de dados abstrato têm complexidade constante ( $O(1)$ ).
- Implementação em vector com re-dimensionamento:
  - Sempre que a pilha ou fila enche, temos que criar um novo vector e transferir a informação para esse vector.
  - Nesses casos, a operação `push` passa a ter complexidade linear ( $O(n)$ ).
  - Os restantes métodos do tipo de dados abstrato têm complexidade constante ( $O(1)$ ).

- Implementação em lista ligada:
  - Todos os métodos do tipo de dados abstrato têm complexidade constante ( $O(1)$ ).
- Implementação em vector com dimensão fixa:
  - Todos os métodos do tipo de dados abstrato têm complexidade constante ( $O(1)$ ).
- Implementação em vector com re-dimensionamento:
  - Sempre que a pilha ou fila enche, temos que criar um novo vector e transferir a informação para esse vector.
  - Nesses casos, a operação `push` passa a ter complexidade linear ( $O(n)$ ).
  - Os restantes métodos do tipo de dados abstrato têm complexidade constante ( $O(1)$ ).

- Implementação em lista ligada:
  - Todos os métodos do tipo de dados abstrato têm complexidade constante ( $O(1)$ ).
- Implementação em vector com dimensão fixa:
  - Todos os métodos do tipo de dados abstrato têm complexidade constante ( $O(1)$ ).
- Implementação em vector com re-dimensionamento:
  - Sempre que a pilha ou fila enche, temos que criar um novo vector e transferir a informação para esse vector.
  - Nesses casos, a operação `push` passa a ter complexidade linear ( $O(n)$ ).
  - Os restantes métodos do tipo de dados abstrato têm complexidade constante ( $O(1)$ ).

## Pilhas e filas

Definições e tipos de dados abstratos

Implementação em lista ligada

Implementação em vector

## Listas biligadas

Comparação entre diferentes tipos de listas ligadas

- Estrutura de dados sequencial em que cada elemento da lista contém uma referência para o próximo elemento e outra para o anterior.
  - Cada lista possui referências para o primeiro e o último elemento a que se refere não existe.
- Ao contrário da lista ligada, permite um acesso sequencial do fim para o início.
- Facilita a remoção do último elemento (`removeLast`).



## Pilhas e filas

Definições e tipos de dados abstratos

Implementação em lista ligada

Implementação em vector

## Listas biligadas

Comparação entre diferentes tipos de listas ligadas

- Estrutura de dados sequencial em que cada elemento da lista contém uma referência para o próximo elemento e outra para o anterior.
  - Cada uma dessas referências terá o valor `null` caso o elemento a que se refere não exista.
- Ao contrário da lista ligada, permite um acesso sequencial do fim para o início.
- Facilita a remoção do último elemento (`removeLast`).

## Pilhas e filas

Definições e tipos de dados  
abstratos

Implementação em lista  
ligada

Implementação em vector

## Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas

- Estrutura de dados sequencial em que cada elemento da lista contém uma referência para o próximo elemento e outra para o anterior.
  - Cada uma dessas referências terá o valor `null` caso o elemento a que se refere não exista.
- Ao contrário da lista ligada, permite um acesso sequencial do fim para o início.
- Facilita a remoção do último elemento (`removeLast`).

## Pilhas e filas

Definições e tipos de dados  
abstratos

Implementação em lista  
ligada

Implementação em vector

## Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas

- Estrutura de dados sequencial em que cada elemento da lista contém uma referência para o próximo elemento e outra para o anterior.
  - Cada uma dessas referências terá o valor `null` caso o elemento a que se refere não exista.
- Ao contrário da lista ligada, permite um acesso sequencial do fim para o início.
- Facilita a remoção do último elemento (`removeLast`).

## Pilhas e filas

Definições e tipos de dados  
abstratos

Implementação em lista  
ligada

Implementação em vector

## Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas

- Estrutura de dados sequencial em que cada elemento da lista contém uma referência para o próximo elemento e outra para o anterior.
  - Cada uma dessas referências terá o valor `null` caso o elemento a que se refere não exista.
- Ao contrário da lista ligada, permite um acesso sequencial do fim para o início.
- Facilita a remoção do último elemento (`removeLast`).

# Lista biligada: nós e ligações

## Pilhas e filas

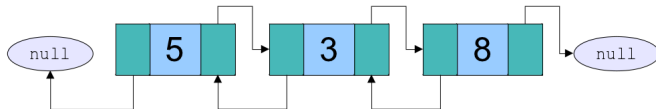
Definições e tipos de dados  
abstratos

Implementação em lista  
ligada

Implementação em vector

## Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas



# Lista biligada: nós e ligações

## Pilhas e filas

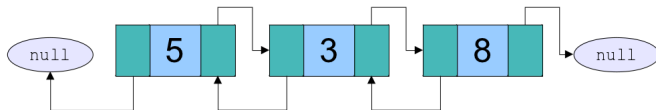
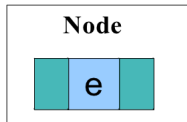
Definições e tipos de dados  
abstratos

Implementação em lista  
ligada

Implementação em vector

## Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas



# Lista biligada: nós e ligações

## Pilhas e filas

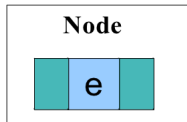
Definições e tipos de dados  
abstratos

Implementação em lista  
ligada

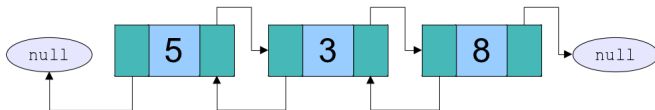
Implementação em vector

## Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas



```
class Node
{
    Node prev;
    int e;
    Node next;
}
```



# Lista biligada: primeiro e último elementos

## Pilhas e filas

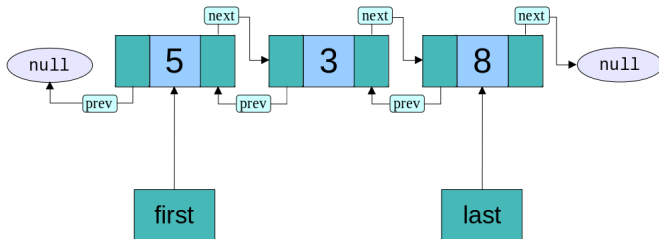
Definições e tipos de dados  
abstratos

Implementação em lista  
ligada

Implementação em vector

## Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas





# Lista biligada: tipo de dados abstrato

- Nome do módulo:

`LinkedData`

- Serviços:

- `addFront`: insere um elemento no início da lista.

- `addTail`: insere um elemento no fim da lista.

- `getFirstElement`: o primeiro elemento da lista.

- `getLastElement`: o último elemento da lista.

- `removeFront`: retira o elemento no início da lista.

- `removeLast`: retira o elemento no fim da lista.

- `clear`: devolve a lista vazia (sem elementos).

- `isEmpty`: verifica se a lista está vazia.

- `clear`: limpa a lista (retorna sem elementos).

## Pilhas e filas

Definições e tipos de dados  
abstratos

Implementação em lista  
ligada

Implementação em vector

## Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas

## Pilhas e filas

Definições e tipos de dados  
abstratos

Implementação em lista  
ligada

Implementação em vector

## Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas

- Nome do módulo:
  - `LinkedList`
- Serviços:
  - `addFirst`: insere um elemento no início da lista.
  - `addLast`: insere um elemento no fim da lista.
  - `first`: devolve o primeiro elemento da lista.
  - `last`: devolve o último elemento lista.
  - `removeFirst`: retira o elemento no início da lista.
  - `removeLast`: retira o elemento no início da lista.
  - `size`: devolve a dimensão actual da lista.
  - `isEmpty`: verifica se a lista está vazia.
  - `clear`: limpa a lista (remove todos os elementos).

## Pilhas e filas

Definições e tipos de dados  
abstratos

Implementação em lista  
ligada

Implementação em vector

## Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas

- Nome do módulo:
  - `LinkedList`
- Serviços:
  - `addFirst`: insere um elemento no início da lista.
  - `addLast`: insere um elemento no fim da lista.
  - `first`: devolve o primeiro elemento da lista.
  - `last`: devolve o último elemento lista.
  - `removeFirst`: retira o elemento no início da lista.
  - `removeLast`: retira o elemento no início da lista.
  - `size`: devolve a dimensão actual da lista.
  - `isEmpty`: verifica se a lista está vazia.
  - `clear`: limpa a lista (remove todos os elementos).

## Pilhas e filas

Definições e tipos de dados  
abstratos

Implementação em lista  
ligada

Implementação em vector

## Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas

- Nome do módulo:
  - `LinkedList`
- Serviços:
  - `addFirst`: insere um elemento no início da lista.
  - `addLast`: insere um elemento no fim da lista.
  - `first`: devolve o primeiro elemento da lista.
  - `last`: devolve o último elemento lista.
  - `removeFirst`: retira o elemento no início da lista.
  - `removeLast`: retira o elemento no início da lista.
  - `size`: devolve a dimensão actual da lista.
  - `isEmpty`: verifica se a lista está vazia.
  - `clear`: limpa a lista (remove todos os elementos).

- Nome do módulo:
  - `LinkedList`
- Serviços:
  - `addFirst`: insere um elemento no início da lista.
  - `addLast`: insere um elemento no fim da lista.
  - `first`: devolve o primeiro elemento da lista.
  - `last`: devolve o último elemento lista.
  - `removeFirst`: retira o elemento no início da lista.
  - `removeLast`: retira o elemento no início da lista.
  - `size`: devolve a dimensão actual da lista.
  - `isEmpty`: verifica se a lista está vazia.
  - `clear`: limpa a lista (remove todos os elementos).

- Nome do módulo:
  - `LinkedList`
- Serviços:
  - `addFirst`: insere um elemento no início da lista.
  - `addLast`: insere um elemento no fim da lista.
  - `first`: devolve o primeiro elemento da lista.
  - `last`: devolve o último elemento lista.
  - `removeFirst`: retira o elemento no início da lista.
  - `removeLast`: retira o elemento no início da lista.
  - `size`: devolve a dimensão actual da lista.
  - `isEmpty`: verifica se a lista está vazia.
  - `clear`: limpa a lista (remove todos os elementos).

## Pilhas e filas

Definições e tipos de dados  
abstratos

Implementação em lista  
ligada

Implementação em vector

## Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas

- Nome do módulo:
  - `LinkedList`
- Serviços:
  - `addFirst`: insere um elemento no início da lista.
  - `addLast`: insere um elemento no fim da lista.
  - `first`: devolve o primeiro elemento da lista.
  - `last`: devolve o último elemento lista.
  - `removeFirst`: retira o elemento no início da lista.
  - `removeLast`: retira o elemento no início da lista.
  - `size`: devolve a dimensão actual da lista.
  - `isEmpty`: verifica se a lista está vazia.
  - `clear`: limpa a lista (remove todos os elementos).

- Nome do módulo:
  - `LinkedList`
- Serviços:
  - `addFirst`: insere um elemento no início da lista.
  - `addLast`: insere um elemento no fim da lista.
  - `first`: devolve o primeiro elemento da lista.
  - `last`: devolve o último elemento lista.
  - `removeFirst`: retira o elemento no início da lista.
  - `removeLast`: retira o elemento no fim da lista.
  - `size`: devolve a dimensão actual da lista.
  - `isEmpty`: verifica se a lista está vazia.
  - `clear`: limpa a lista (remove todos os elementos).



- Nome do módulo:
  - `LinkedList`
- Serviços:
  - `addFirst`: insere um elemento no início da lista.
  - `addLast`: insere um elemento no fim da lista.
  - `first`: devolve o primeiro elemento da lista.
  - `last`: devolve o último elemento lista.
  - `removeFirst`: retira o elemento no início da lista.
  - `removeLast`: retira o elemento no fim da lista.
  - `size`: devolve a dimensão actual da lista.
  - `isEmpty`: verifica se a lista está vazia.
  - `clear`: limpa a lista (remove todos os elementos).

- Nome do módulo:
  - `LinkedList`
- Serviços:
  - `addFirst`: insere um elemento no início da lista.
  - `addLast`: insere um elemento no fim da lista.
  - `first`: devolve o primeiro elemento da lista.
  - `last`: devolve o último elemento lista.
  - `removeFirst`: retira o elemento no início da lista.
  - `removeLast`: retira o elemento no início da lista.
  - `size`: devolve a dimensão actual da lista.
  - `isEmpty`: verifica se a lista está vazia.
  - `clear`: limpa a lista (remove todos os elementos).

- Nome do módulo:
  - `LinkedList`
- Serviços:
  - `addFirst`: insere um elemento no início da lista.
  - `addLast`: insere um elemento no fim da lista.
  - `first`: devolve o primeiro elemento da lista.
  - `last`: devolve o último elemento lista.
  - `removeFirst`: retira o elemento no início da lista.
  - `removeLast`: retira o elemento no início da lista.
  - `size`: devolve a dimensão actual da lista.
  - `isEmpty`: verifica se a lista está vazia.
  - `clear`: limpa a lista (remove todos os elementos).

- Nome do módulo:
  - `LinkedList`
- Serviços:
  - `addFirst`: insere um elemento no início da lista.
  - `addLast`: insere um elemento no fim da lista.
  - `first`: devolve o primeiro elemento da lista.
  - `last`: devolve o último elemento lista.
  - `removeFirst`: retira o elemento no início da lista.
  - `removeLast`: retira o elemento no início da lista.
  - `size`: devolve a dimensão actual da lista.
  - `isEmpty`: verifica se a lista está vazia.
  - `clear`: limpa a lista (remove todos os elementos).

- Nome do módulo:
  - `LinkedList`
- Serviços:
  - `addFirst`: insere um elemento no início da lista.
  - `addLast`: insere um elemento no fim da lista.
  - `first`: devolve o primeiro elemento da lista.
  - `last`: devolve o último elemento lista.
  - `removeFirst`: retira o elemento no início da lista.
  - `removeLast`: retira o elemento no início da lista.
  - `size`: devolve a dimensão actual da lista.
  - `isEmpty`: verifica se a lista está vazia.
  - `clear`: limpa a lista (remove todos os elementos).

## Lista biligada: novo elemento em lista vazia

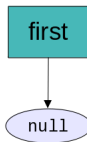
```
addLast(1)
```

```
size == 0
```

## Lista biligada: novo elemento em lista vazia

```
addLast(1)
```

```
size == 0
```



## Lista biligada: novo elemento em lista vazia

### Pilhas e filas

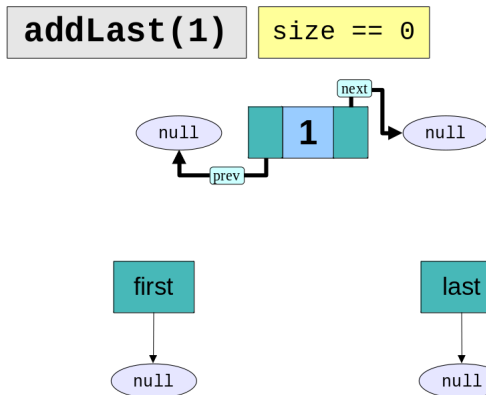
Definições e tipos de dados  
abstratos

Implementação em lista  
ligada

Implementação em vector

### Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas





## Lista biligada: novo elemento em lista vazia

### Pilhas e filas

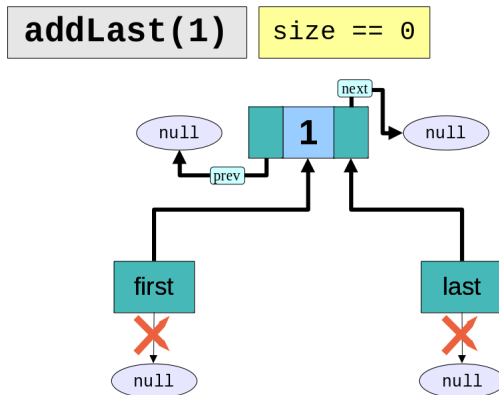
Definições e tipos de dados  
abstratos

Implementação em lista  
ligada

Implementação em vector

### Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas



## Lista biligada: novo elemento

```
addLast(8)
```

```
size > 0
```

### Pilhas e filas

Definições e tipos de dados  
abstratos

Implementação em lista  
ligada

Implementação em vector

### Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas

## Lista biligada: novo elemento

### Pilhas e filas

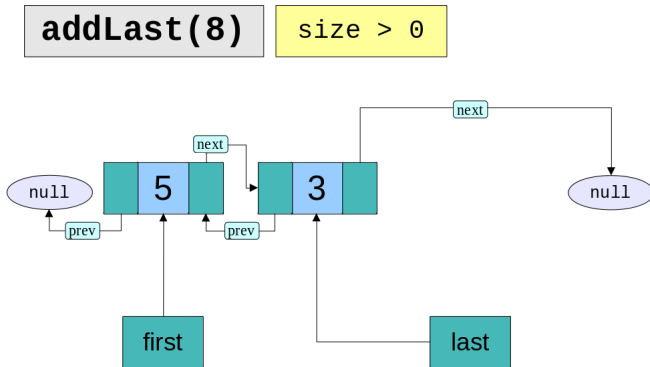
Definições e tipos de dados  
abstratos

Implementação em lista  
ligada

Implementação em vector

### Listas biligadas

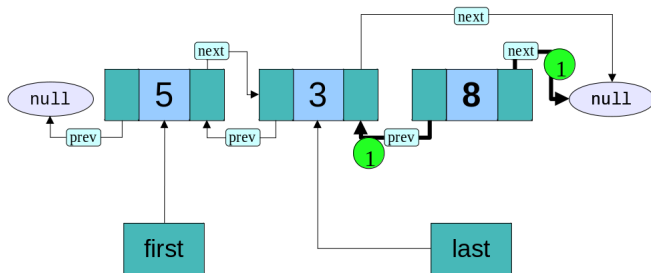
Comparação entre  
diferentes tipos de  
listas ligadas



## Lista biligada: novo elemento

**addLast(8)**

size > 0



### Pilhas e filas

Definições e tipos de dados  
abstratos

Implementação em lista  
ligada

Implementação em vector

### Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas

# Lista biligada: novo elemento

## Pilhas e filas

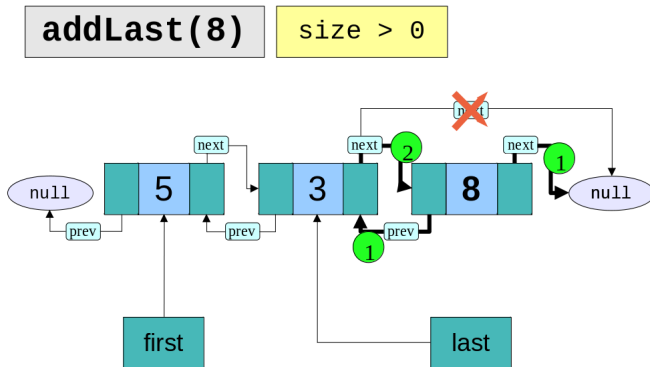
Definições e tipos de dados  
abstratos

Implementação em lista  
ligada

Implementação em vector

## Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas



## Lista biligada: novo elemento

### Pilhas e filas

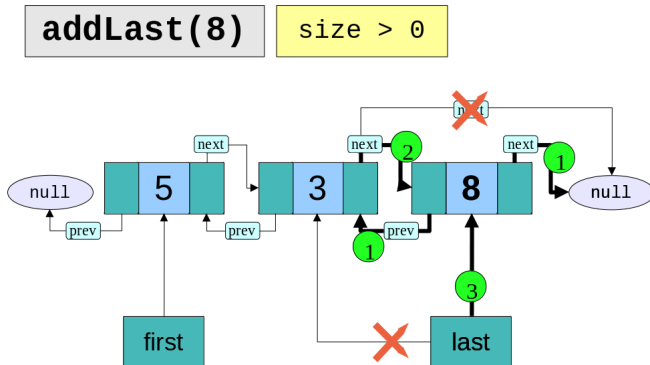
Definições e tipos de dados  
abstratos

Implementação em lista  
ligada

Implementação em vector

### Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas



## Lista biligada: remoção do último elemento

```
removeFirst()
```

```
size == 1
```

### Pilhas e filas

Definições e tipos de dados  
abstratos

Implementação em lista  
ligada

Implementação em vector

### Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas



## Lista biligada: remoção do último elemento

### Pilhas e filas

Definições e tipos de dados  
abstratos

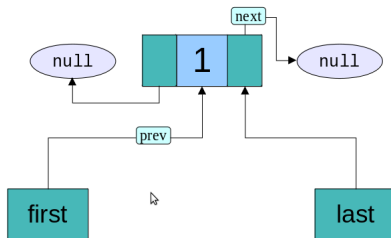
Implementação em lista  
ligada

Implementação em vector

### Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas

`removeFirst()`      `size == 1`





## Lista biligada: remoção do último elemento

### Pilhas e filas

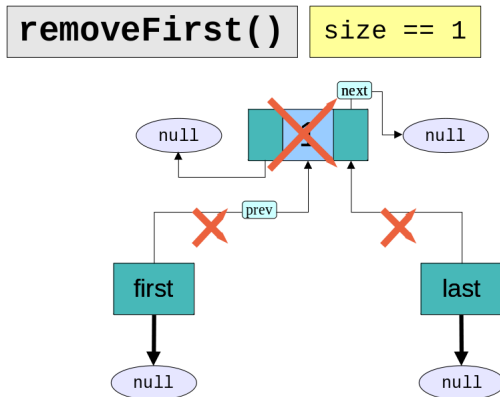
Definições e tipos de dados  
abstratos

Implementação em lista  
ligada

Implementação em vector

### Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas



## Lista biligada: remoção

```
removeFirst()
```

```
size > 1
```

### Pilhas e filas

Definições e tipos de dados  
abstratos

Implementação em lista  
ligada

Implementação em vector

### Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas

# Lista biligada: remoção

## Pilhas e filas

Definições e tipos de dados  
abstratos

Implementação em lista  
ligada

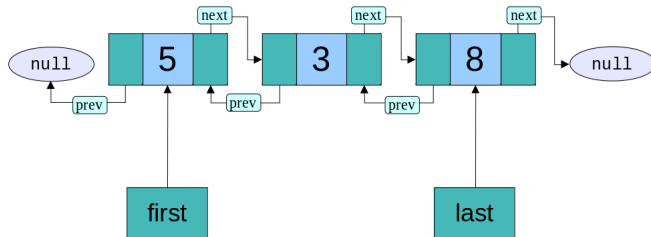
Implementação em vector

## Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas

`removeFirst()`

`size > 1`



# Lista biligada: remoção

## Pilhas e filas

Definições e tipos de dados  
abstratos

Implementação em lista  
ligada

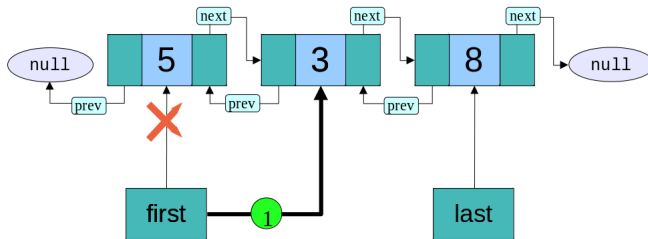
Implementação em vector

## Listas biligadas

Comparação entre  
diferentes tipos de  
listas ligadas

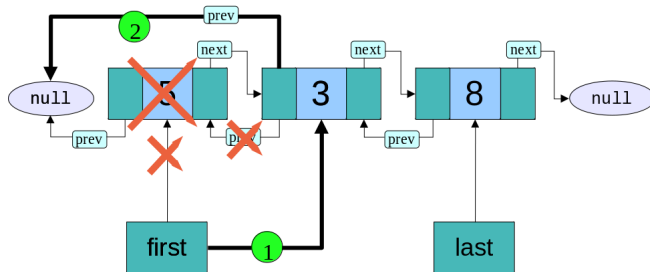
**removeFirst()**

size > 1



## Pilhas, Filas e Listas Biligadas

```
size > 1
```



### Definições e tipos de dados abstratos

### Implementação em vector

## Listas biligadas

## Comparação entre diferentes tipos de listas ligadas

# Tipos de Listas Ligadas

Tipo de Lista	Simple	Simple	Circular Simple	Biligada	Circular Biligada
Atributos	first	first	last	first	first
Operações		last		last	(last)
<i>insert first</i>	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$
<i>remove first</i>	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$
<i>insert last</i>	$O(n)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$
<i>remove last</i>	$O(n)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$
<i>scan forward</i>	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
<i>scan backward</i>	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n)$	$O(n)$
<i>insert middle</i>	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
<i>remove middle</i>	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$

## Pilhas e filas

Definições e tipos de dados abstratos

Implementação em lista ligada

Implementação em vector

## Listas biligadas

Comparação entre diferentes tipos de listas ligadas

# Tipos de Listas Ligadas

Tipo de Lista	Simple	Simple	Circular Simple	Biligada	Circular Biligada
Atributos	first	first	last	first	first
Operações		last		last	(last)
<i>insert first</i>	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$
<i>remove first</i>	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$
<i>insert last</i>	$O(n)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$
<i>remove last</i>	$O(n)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$
<i>scan forward</i>	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
<i>scan backward</i>	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n)$	$O(n)$
<i>insert middle</i>	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
<i>remove middle</i>	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$

## Pilhas e filas

Definições e tipos de dados abstratos

Implementação em lista ligada

Implementação em vector

## Listas biligadas

Comparação entre diferentes tipos de listas ligadas