Aula 14: Estruturas de dados básicas, parte 2

David Déharbe
Programa de Pós-graduação em Sistemas e Computação
Universidade Federal do Rio Grande do Norte
Centro de Ciências Exatas e da Terra
Departamento de Informática e Matemática Aplicada

Download me from `http://DavidDeharbe.github.io`.

PPgSC

Plano da aula

Listas encadeadas
- Introdução
- Listas simplesmente encadeadas
- Listas simplesmente encadeadas circulares
- Listas duplamente encadeadas
- Listas duplamente encadeadas circulares
- Célula sentinela

Listas encadeadas e arranjos

PPgSC

Listas encadeadas

- simples
- coleções homogêneas de dados
- muitas variações
  1. ordenada ou não ordenadas
  2. simplesmente encadeada ou duplamente encadeada
  3. circular ou não circular
  4. sem sentinela ou com sentinela
- implementação:
  - registros e ponteiros
  - arranjos

PPgSC

Listas simplesmente encadeadas

Formalização 1/2

- coleção homogênea: $\langle v_1, \ldots, v_n \rangle$
- possivelmente vazia: $n \geq 0$

- a representação é um conjunto de células de lista
- cada valor é armazenado em uma célula

$$\forall i \mid 1 \leq i \leq n \cdot cell(v_i).val = v_i$$

- cada valor é armazenado em uma célula diferente

$$\forall i, j \mid 1 \leq i < j \leq n \cdot cell(v_i) \neq cell(v_j)$$

- as células são encadeadas
  - cada célula possui uma referência para a célula seguinte:

$$\forall i \mid 1 \leq i < n \cdot cell(v_i).next = cell(v_{i+1})$$

  - com exceção da última que referência um valor especial:

$$cell(v_n).next = \text{NIL}$$

PPgSC

Listas simplesmente encadeadas

Formalização 2/2

- coleção homogênea: $\langle v_1, \ldots, v_n \rangle$
- possivelmente vazia: $n \geq 0$

Seja *hd* a representação da coleção

- quando não vazia, o acesso à coleção é realizado pela primeira célula

$$hd = cell(v_1)$$

- a coleção vazia é representada por um valor especial

$$hd = \textsc{Nil}$$

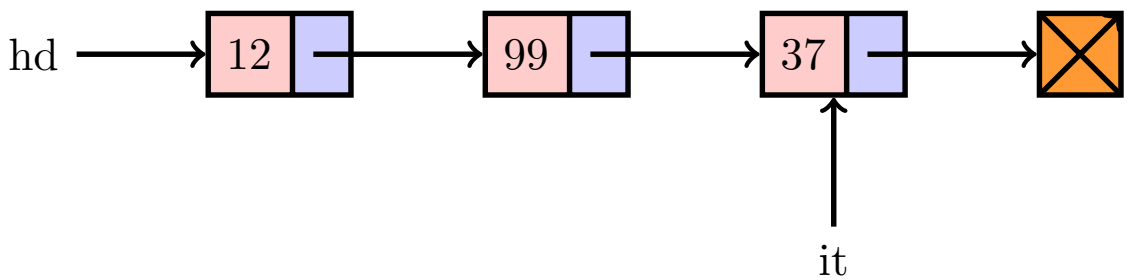
Uma posição na coleção (iterador) na coleção é a célula que guarda o valor nesta posição.

PPgSC

Listas simplesmente encadeadas

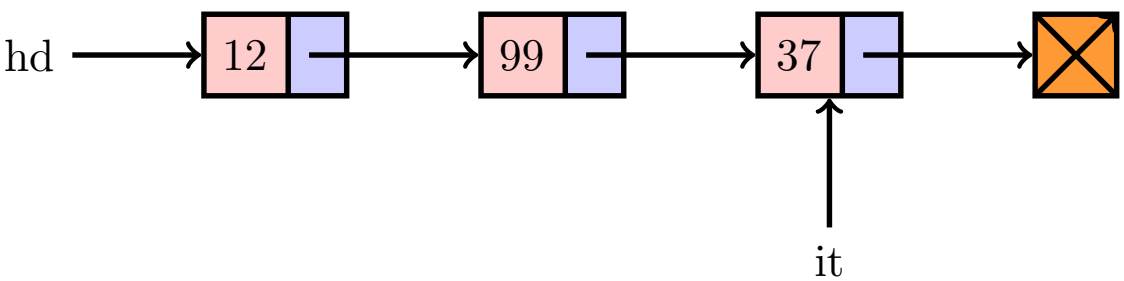
Ilustração

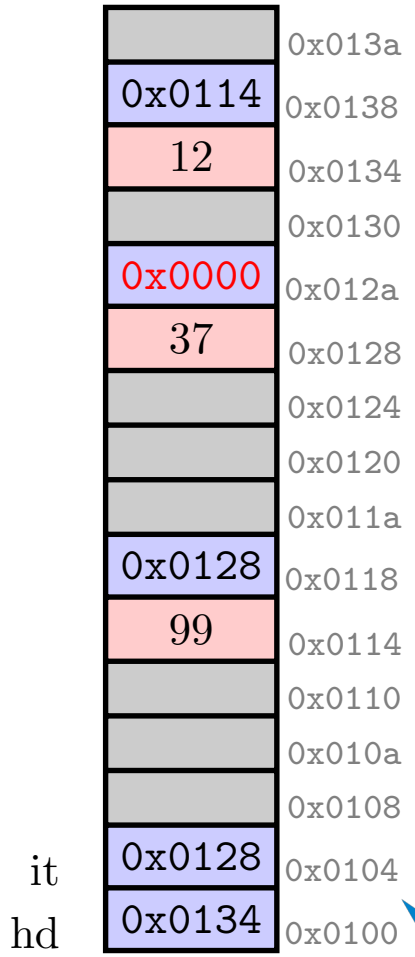
$\langle 12, 99, 37 \rangle$



Listas simplesmente encadeadas

Implementação usando registros e ponteiros

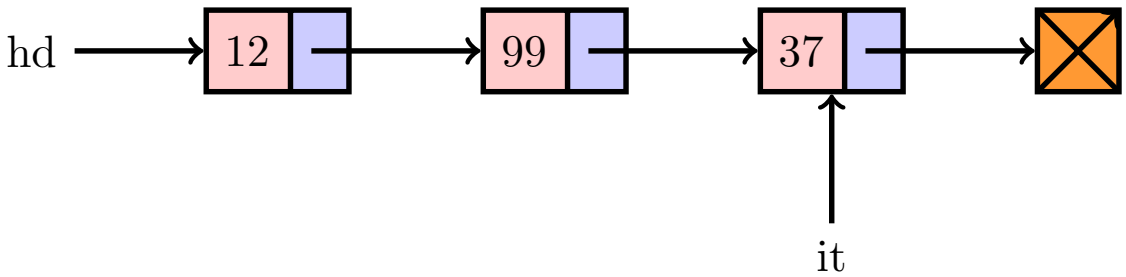




$it = hd$
while $it \neq \text{NIL}$
    // process $it.val$
    $it = it.next$

Inserção

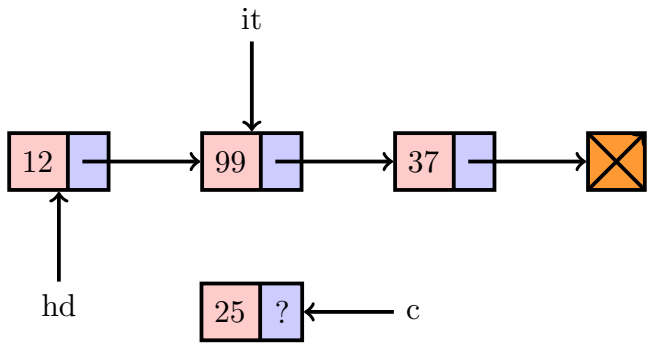
Listas simplesmente encadeadas



- caso geral: após uma dada posição
- caso especial: em primeira posição

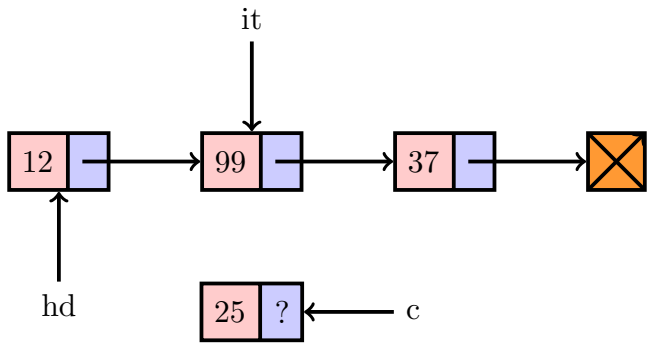
Inserção: caso geral

Listas simplesmente encadeadas



Inserção: caso geral

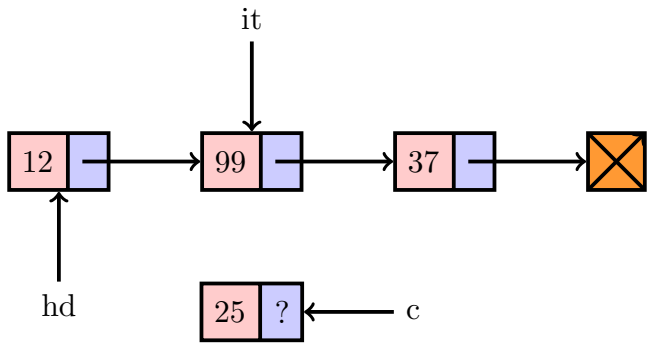
Listas simplesmente encadeadas



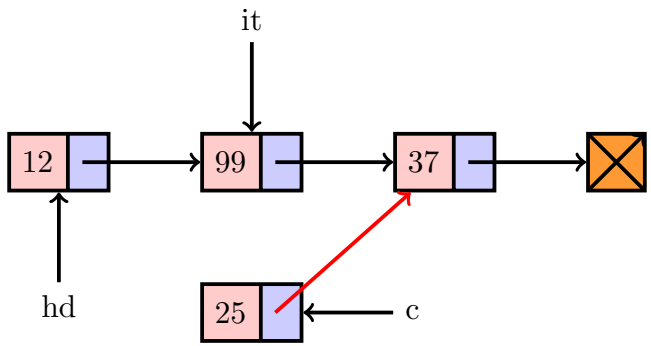
$$c.next = it.next$$

Inserção: caso geral

Listas simplesmente encadeadas

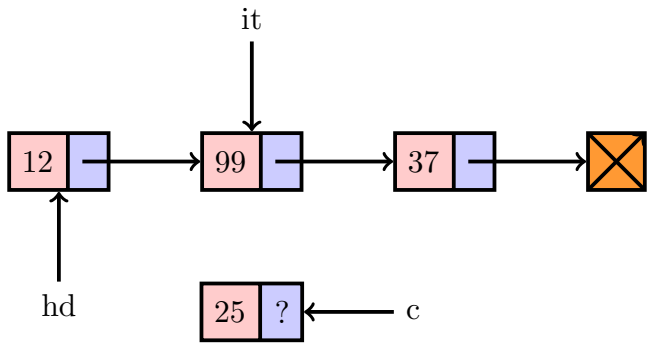


$$c.next = it.next$$

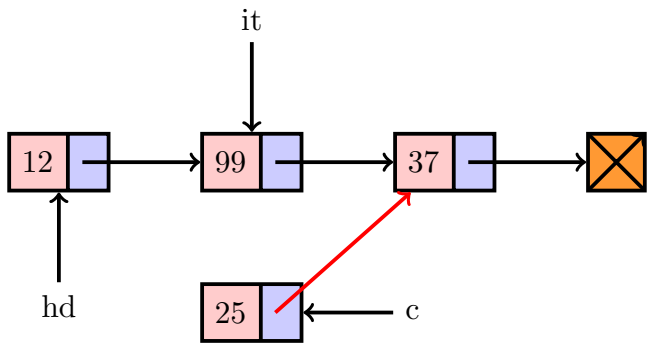


Inserção: caso geral

Listas simplesmente encadeadas



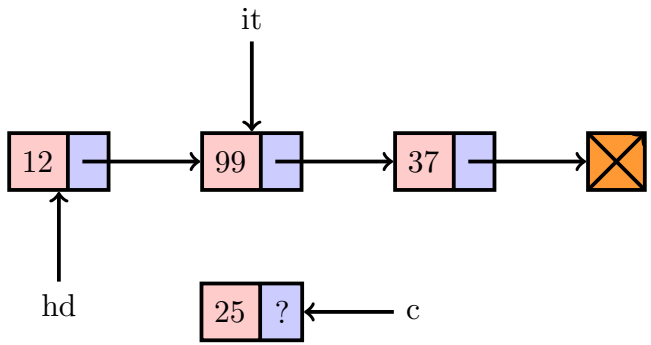
$c.next = it.next$



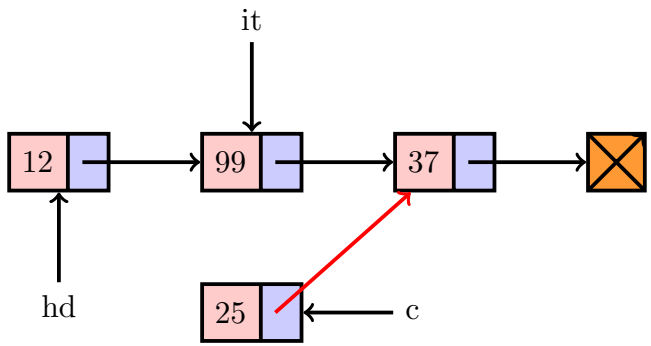
$it.next = c$

Inserção: caso geral

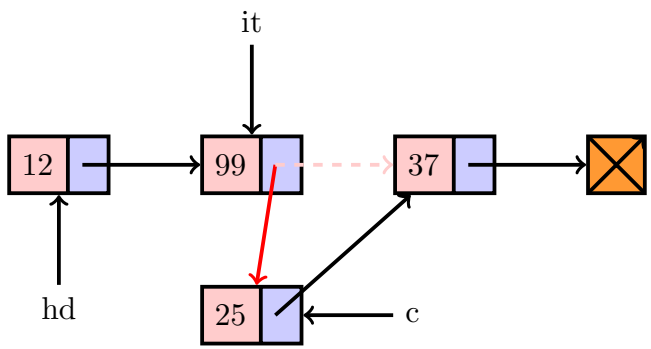
Listas simplesmente encadeadas



$c.next = it.next$

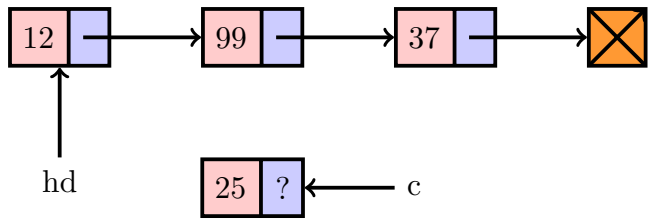


$it.next = c$



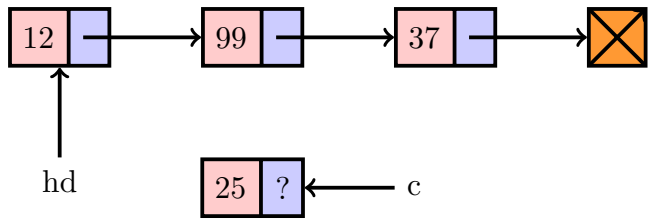
Inserção: caso especial

Listas simplesmente encadeadas



Inserção: caso especial

Listas simplesmente encadeadas

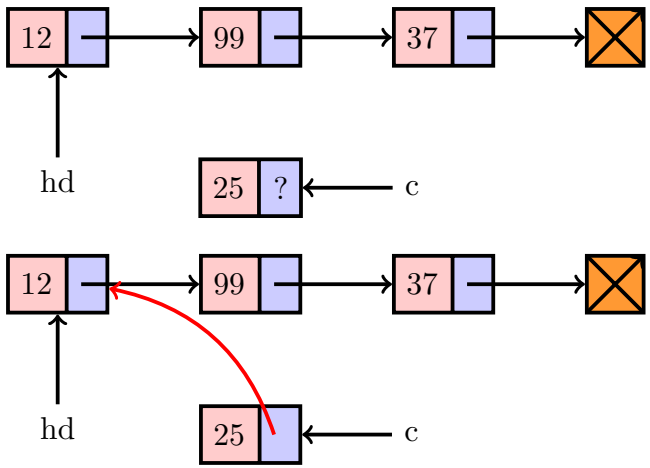


$$c.next = hd$$

Inserção: caso especial

Listas simplesmente encadeadas

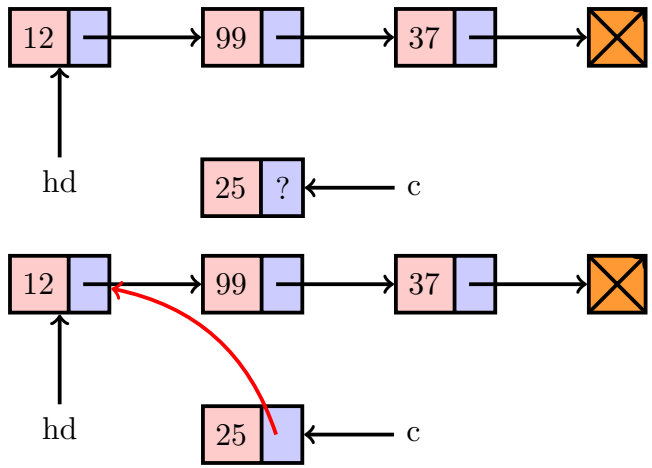
$c.next = hd$



Inserção: caso especial

Listas simplesmente encadeadas

$c.next = hd$



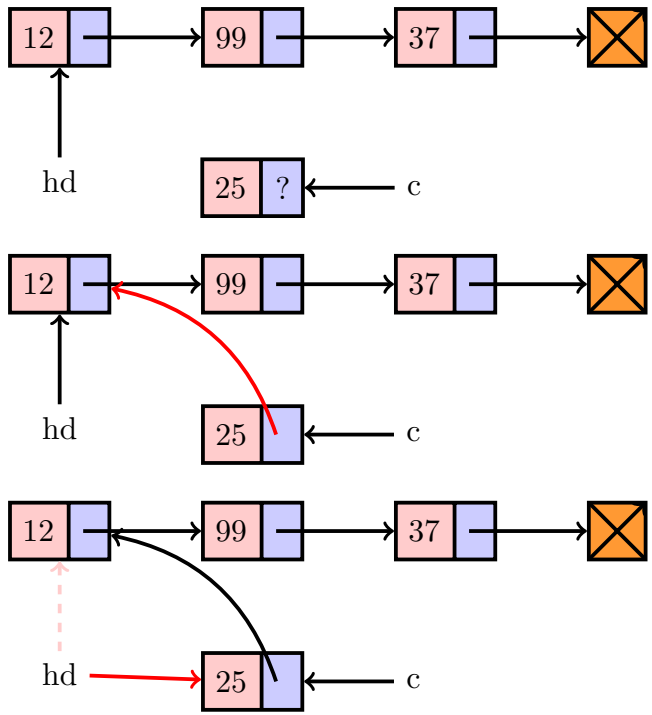
$hd = c$

Inserção: caso especial

Listas simplesmente encadeadas

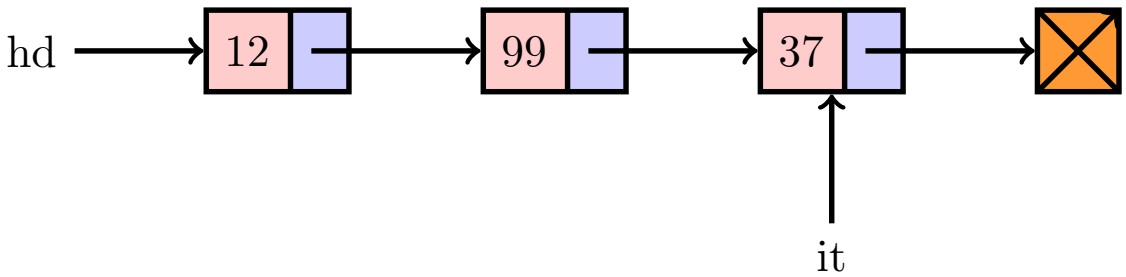
$c.next = hd$

$hd = c$



Remoção

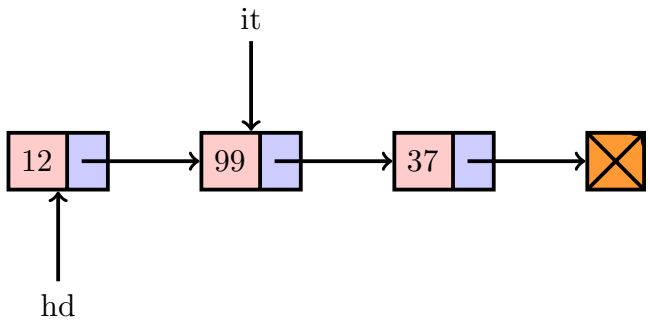
Listas simplesmente encadeadas



- caso geral: após uma dada posição
- caso especial: em primeira posição

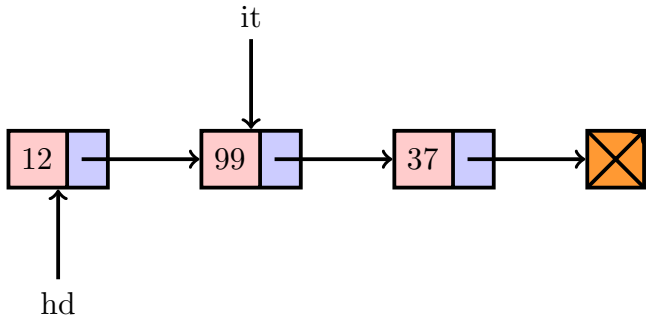
Remoção: caso geral

Listas simplesmente encadeadas



Remoção: caso geral

Listas simplesmente encadeadas

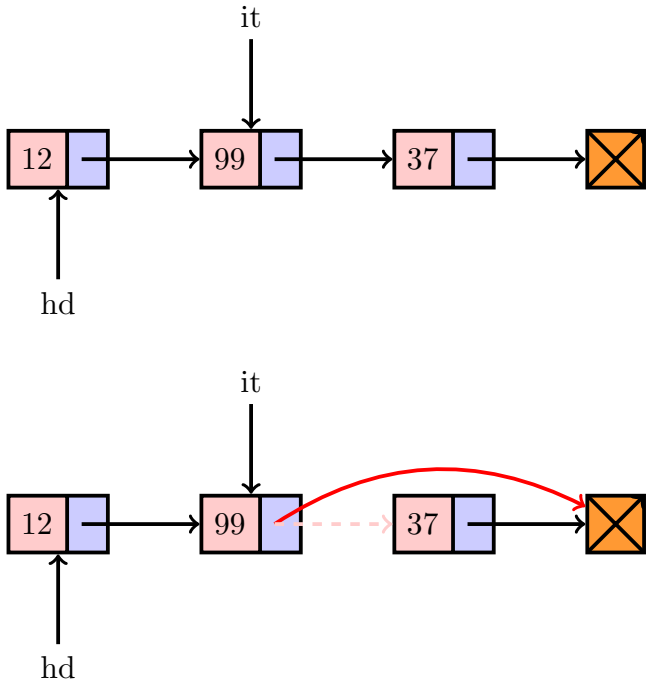


$$it.next = it.next.next$$

Remoção: caso geral

Listas simplesmente encadeadas

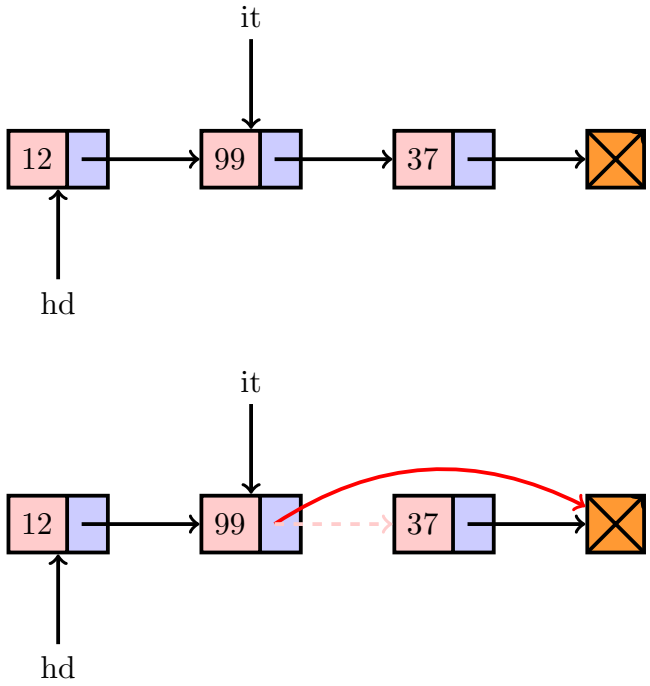
$$it.next = it.next.next$$



Remoção: caso geral

Listas simplesmente encadeadas

$it.next = it.next.next$

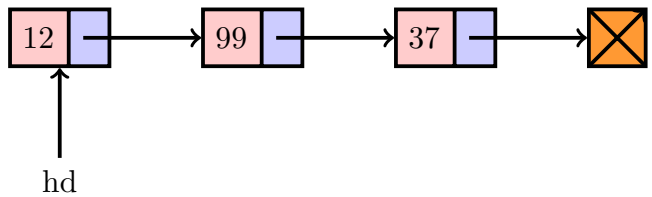


- se for necessário gerenciar os recursos de memória:

$$tmp = it.next$$
$$it.next = it.next.next$$
$$\text{FREE}(tmp)$$

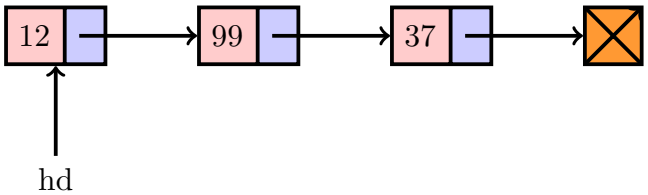
Remoção: caso especial

Listas simplesmente encadeadas



Remoção: caso especial

Listas simplesmente encadeadas

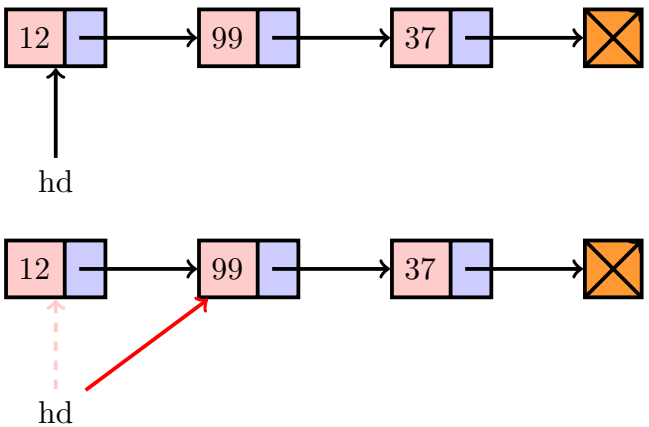


$$hd = hd.next$$

Remoção: caso especial

Listas simplesmente encadeadas

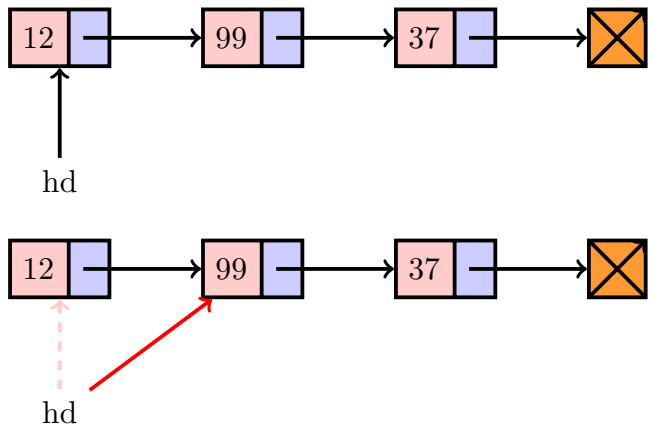
$$hd = hd.next$$



Remoção: caso especial

Listas simplesmente encadeadas

$hd = hd.next$



- se for necessário gerenciar os recursos de memória:

$tmp = hd$
$hd = hd.next$
$\textsc{Free}(tmp)$

Exercícios

Listas encadeadas

- Escrever um algoritmo de inserção
  - assumindo a lista não ordenada
  - assumindo a lista ordenada
- Escrever um algoritmo de remoção
  - assumindo a lista não ordenada
  - assumindo a lista ordenada
- Escrever um algoritmo de busca
  - assumindo a lista não ordenada
  - assumindo a lista ordenada

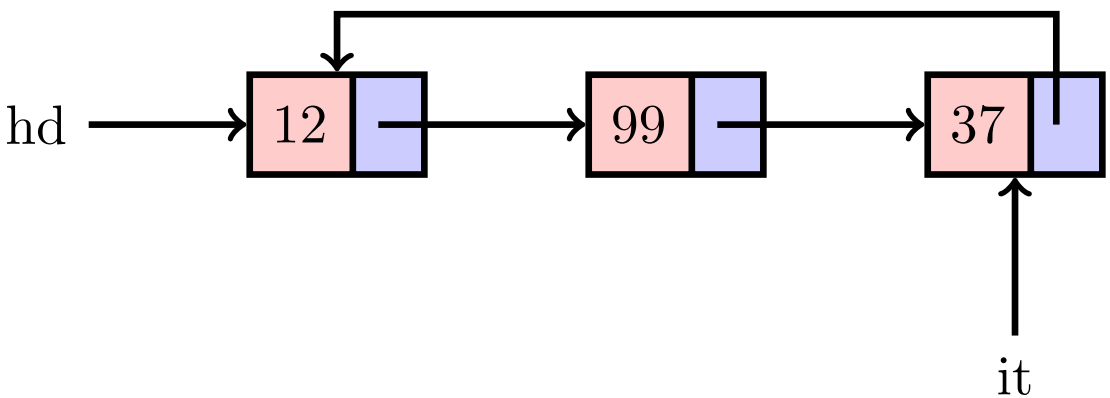
Listas simplesmente encadeadas circulares

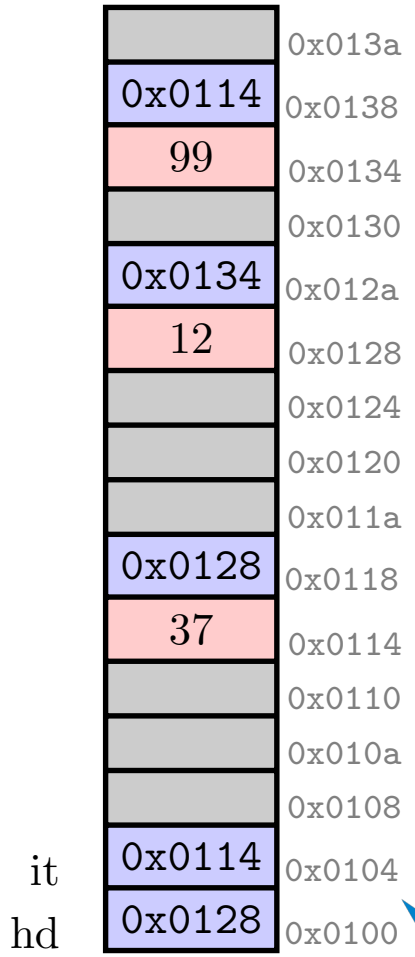
- a última célula referência a primeira

$$cell(v_n).next = cell(v_1)$$

Listas simplesmente encadeadas circulares

Implementação usando registros e ponteiros





```
if hd ≠ NIL
    it = hd
    repeat
        // process it.val
        it = it.next
    until it == hd
```

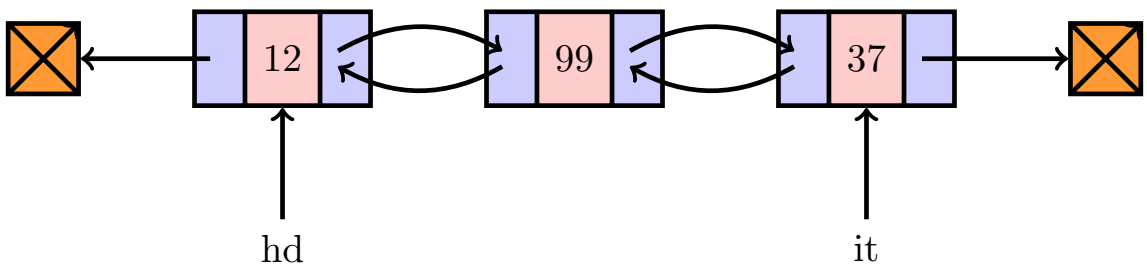
Exercícios

Listas simplesmente encadeadas circulares

Adapte os algoritmos de inserção, remoção e busca desenvolvidos para as listas simplesmente encadeadas circulares.

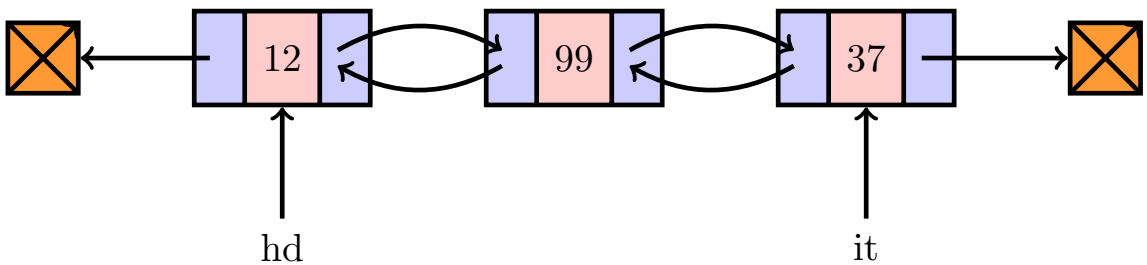
Listas duplamente encadeadas

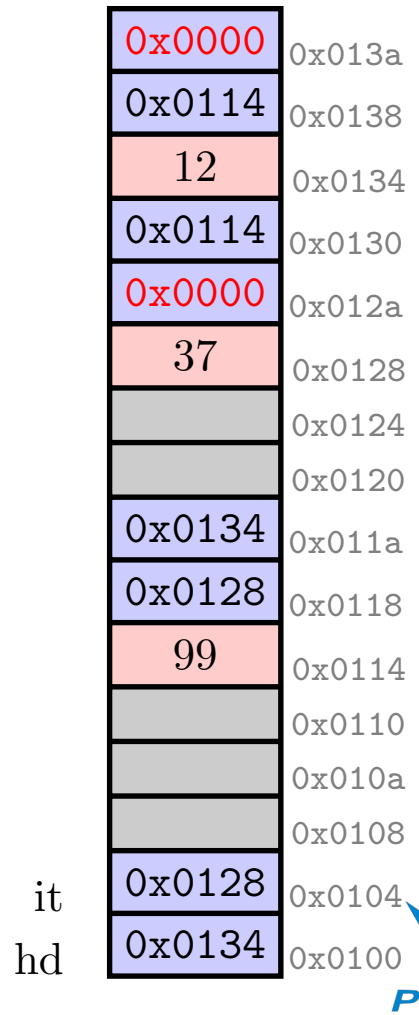
- acesso direto à posição anterior
- remoção do item na posição



Implementação usando registros e ponteiros

Listas duplamente encadeadas





- as células são duplamente encadeadas
  - cada célula possui uma referência para a célula anterior:

$$\forall i \mid 1 < i \leq n \cdot cell(v_i).prev = cell(v_{i-1})$$

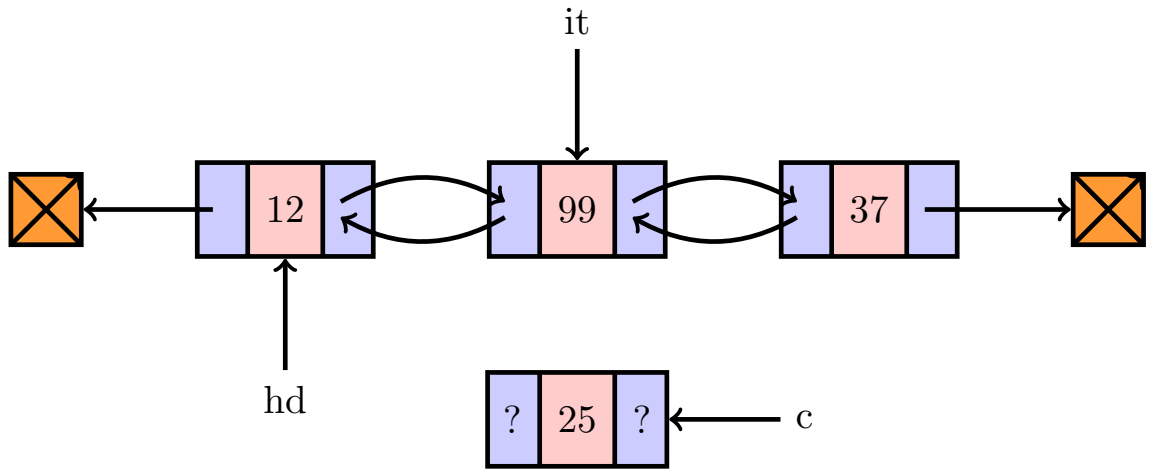
  - com exceção da primeira que referência um valor especial:

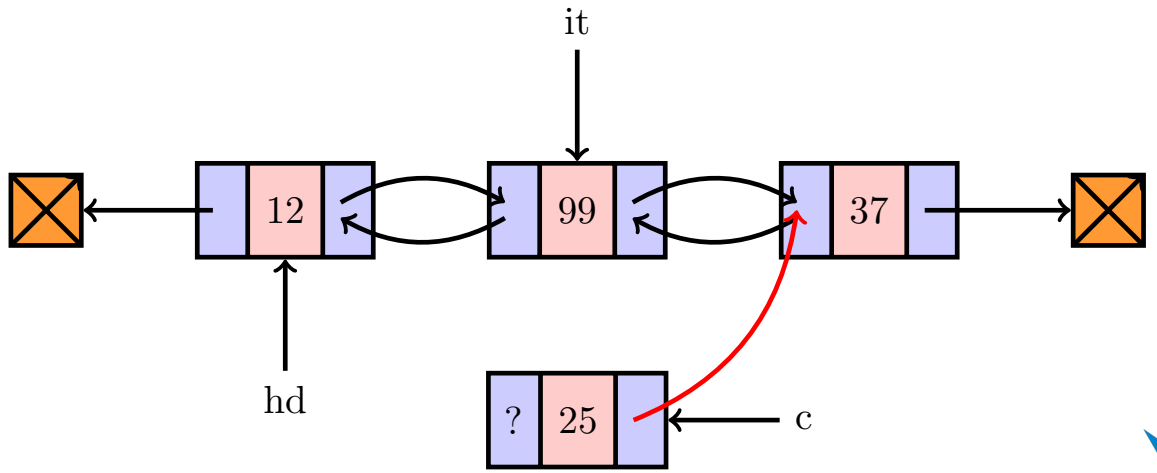
$$cell(v_1).prev = \text{NIL}$$

Inserção após a posição atual

Listas duplamente encadeadas

$c.next = it.next$

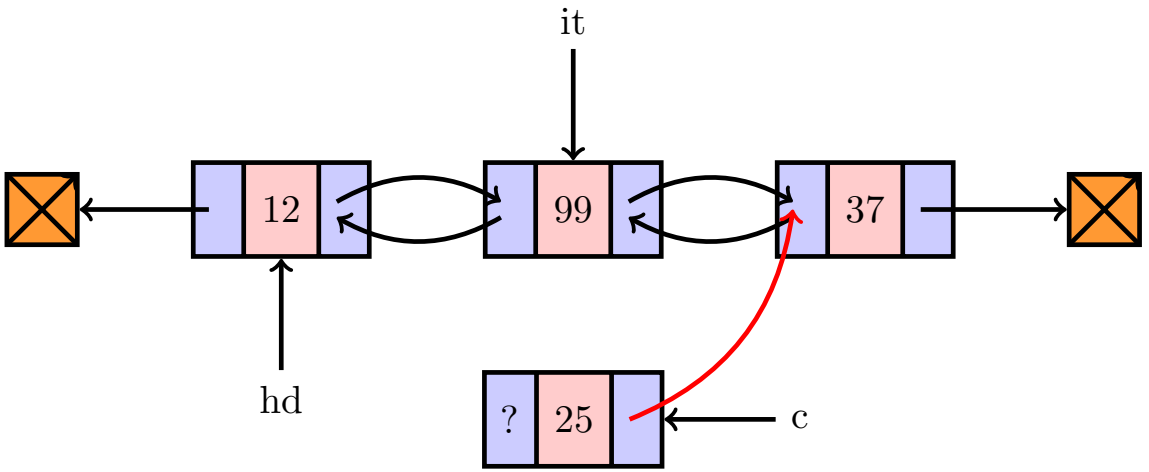




Inserção após a posição atual

Listas duplamente encadeadas

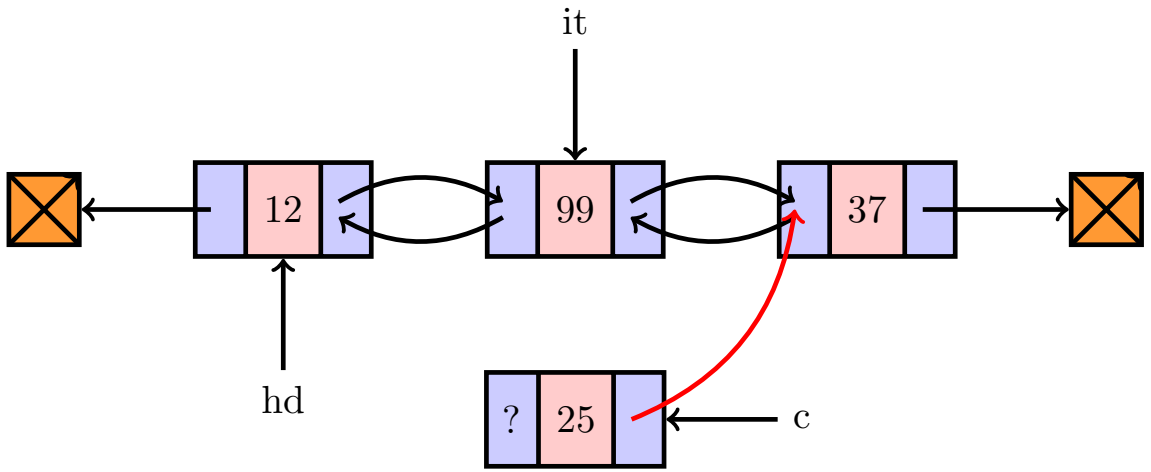
$c.next = it.next$



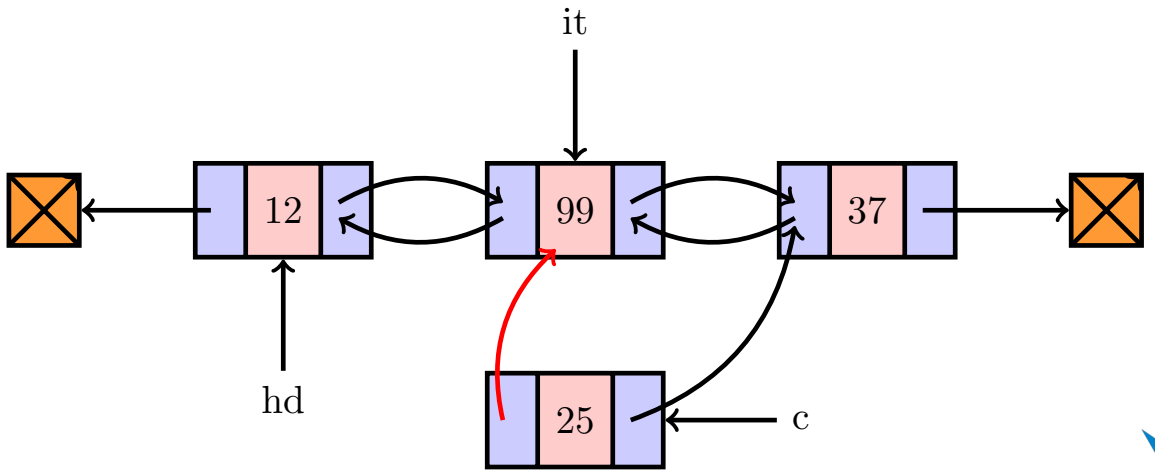
Inserção após a posição atual

Listas duplamente encadeadas

$c.next = it.next$



$c.prev = it$

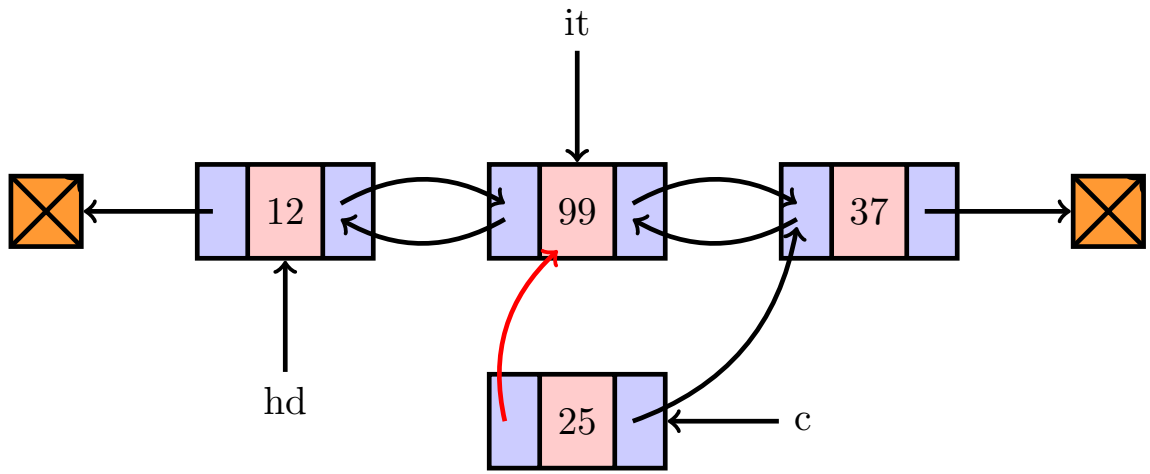


Inserção após a posição atual

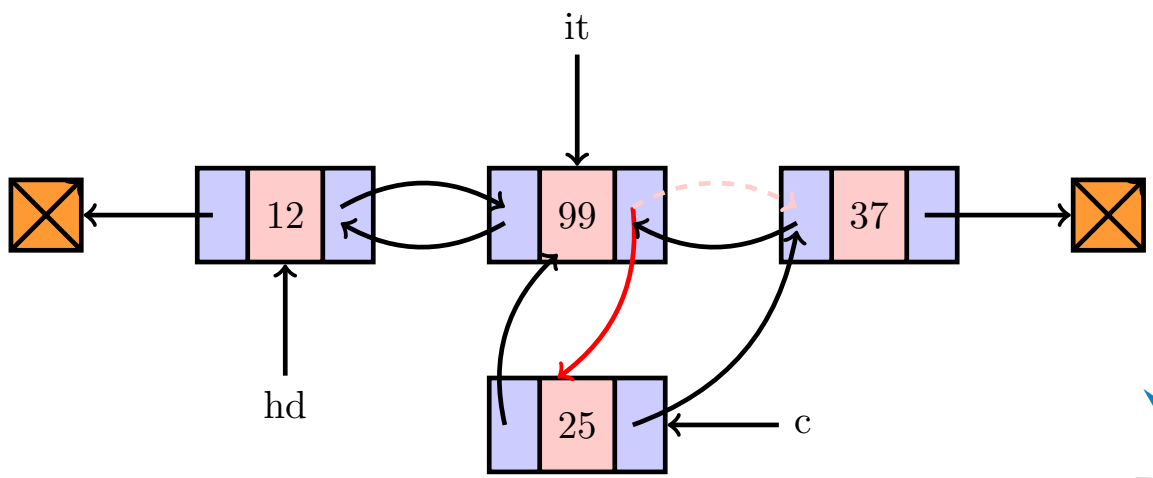
Listas duplamente encadeadas

$c.next = it.next$

$c.prev = it$



$it.next = c$

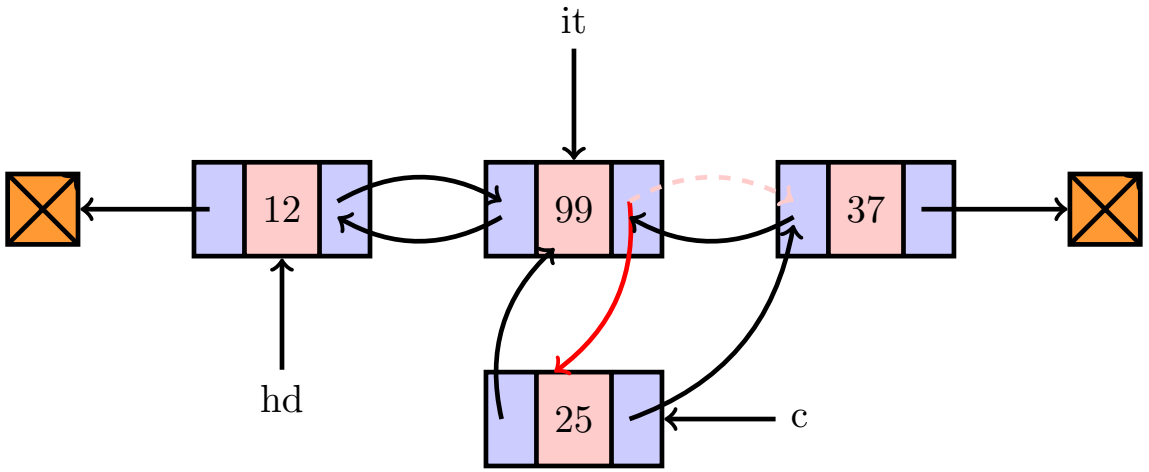


Inserção após a posição atual

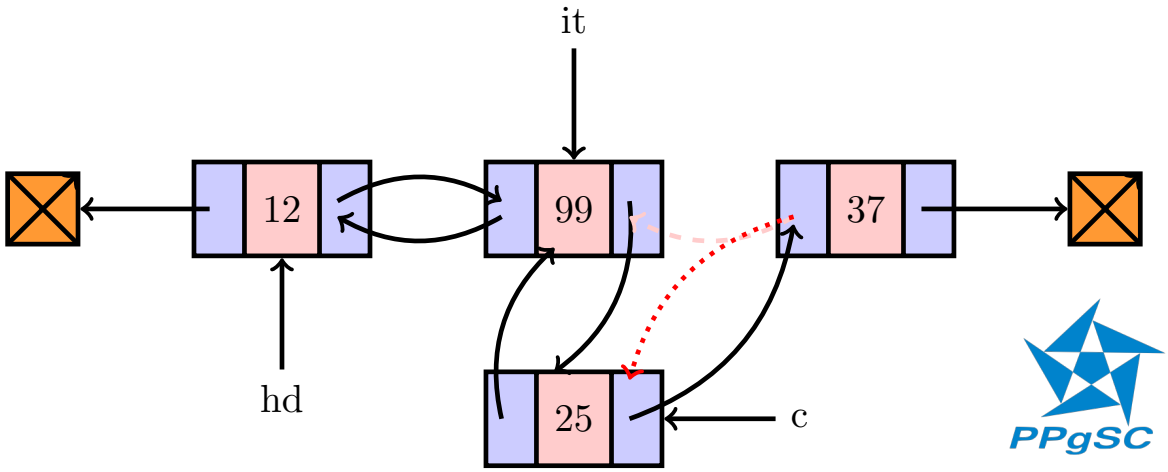
Listas duplamente encadeadas

$$c.next = it.next$$
$$c.prev = it$$

$$it.next = c$$



if $c.next \neq \text{NIL}$
$$c.next.prev = c$$



Inserção após a posição atual

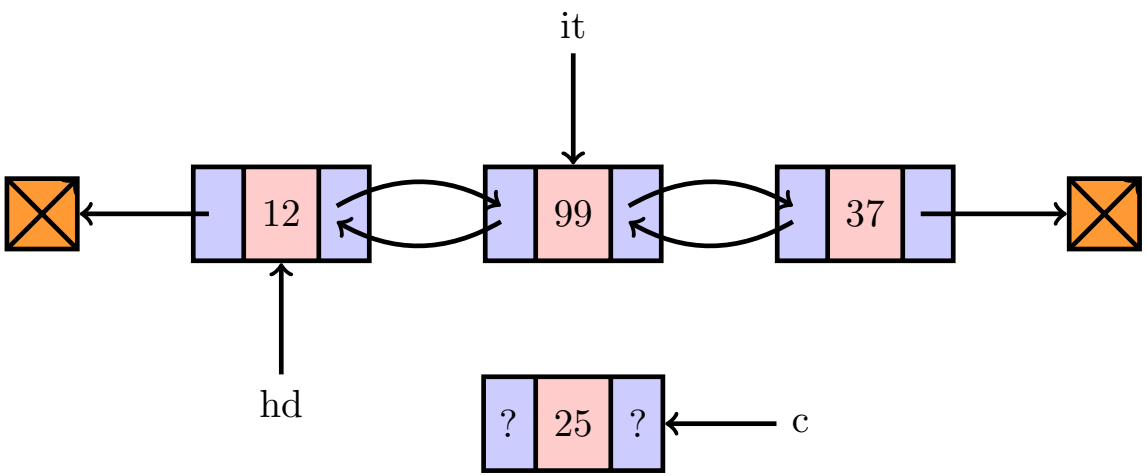
Listas duplamente encadeadas

$$
\begin{aligned}
&c.next = it.next \\
&c.prev = it \\
&it.next = c \\
&\textbf{if } c.next \neq \textsc{Nil} \\
&\qquad c.next.prev = c
\end{aligned}
$$

Inserção antes da posição atual

Listas duplamente encadeadas

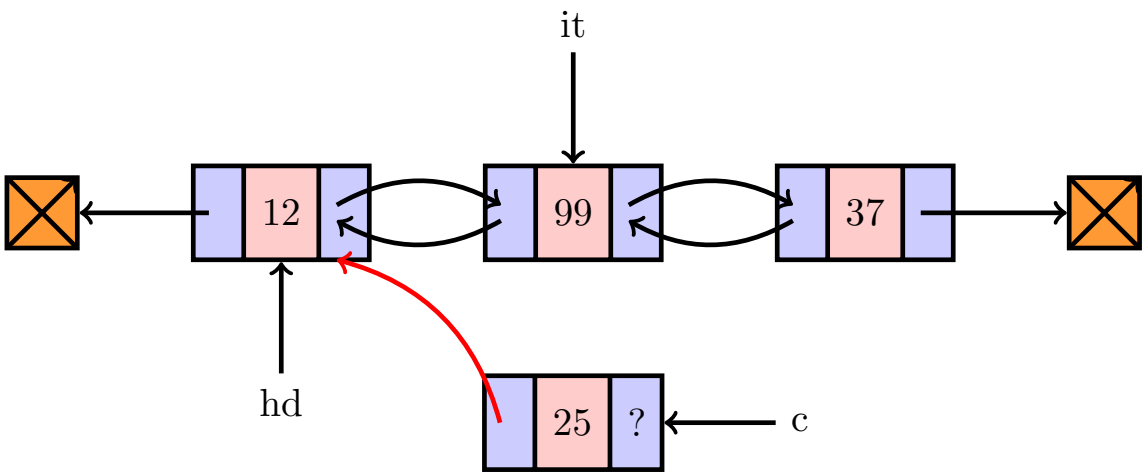
$c.next = it.next$
$c.prev = it$
$it.next = c$
if $c.next \neq \text{NIL}$
$\quad c.next.prev = c$



Inserção antes da posição atual

Listas duplamente encadeadas

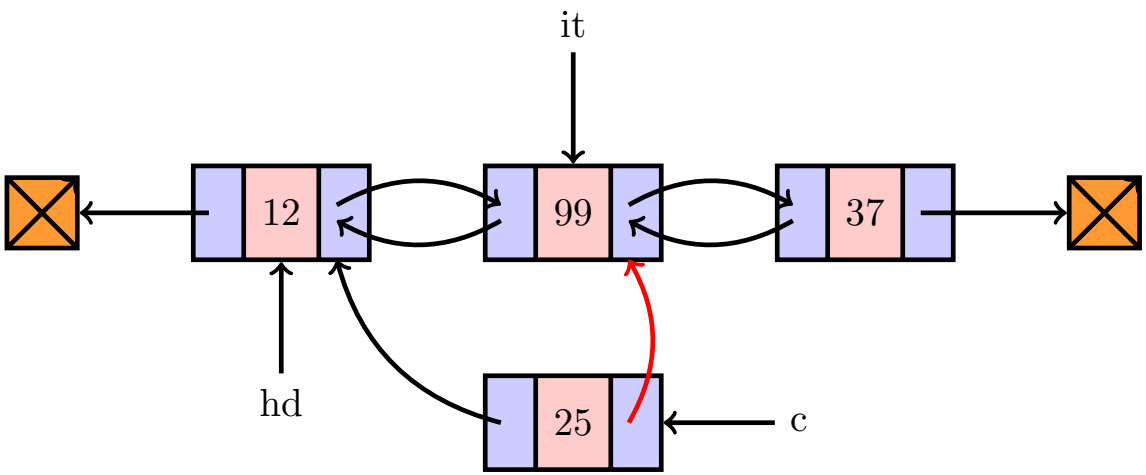
$$
\begin{aligned}
c.prev &= it.prev \\
c.prev &= it \\
it.next &= c \\
&\textbf{if } c.next \neq \textsc{Nil} \\
&\qquad c.next.prev = c
\end{aligned}
$$



Inserção antes da posição atual

Listas duplamente encadeadas

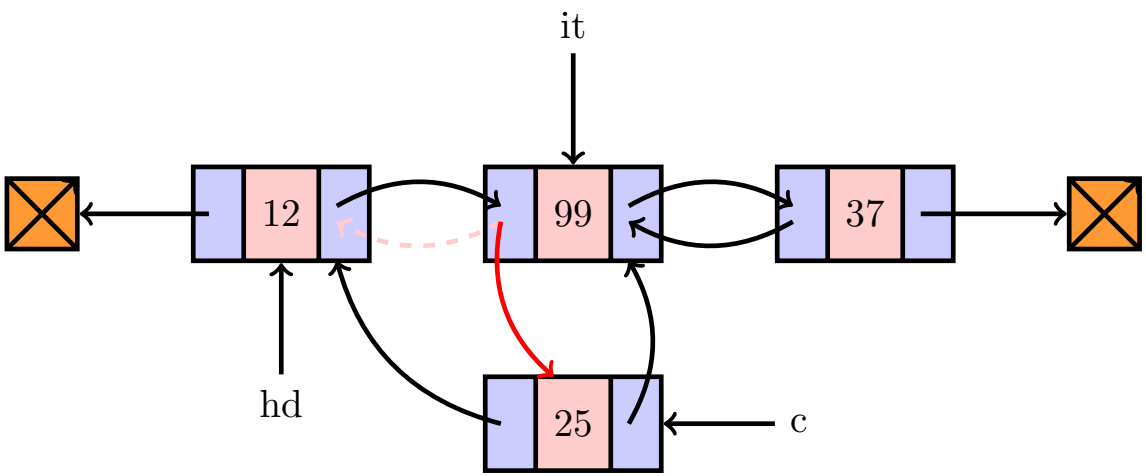
$$
\begin{aligned}
c.prev &= it.prev \\
c.next &= it \\
it.next &= c \\
\textbf{if } & c.next \neq \textsc{Nil} \\
& \quad c.next.prev = c
\end{aligned}
$$



Inserção antes da posição atual

Listas duplamente encadeadas

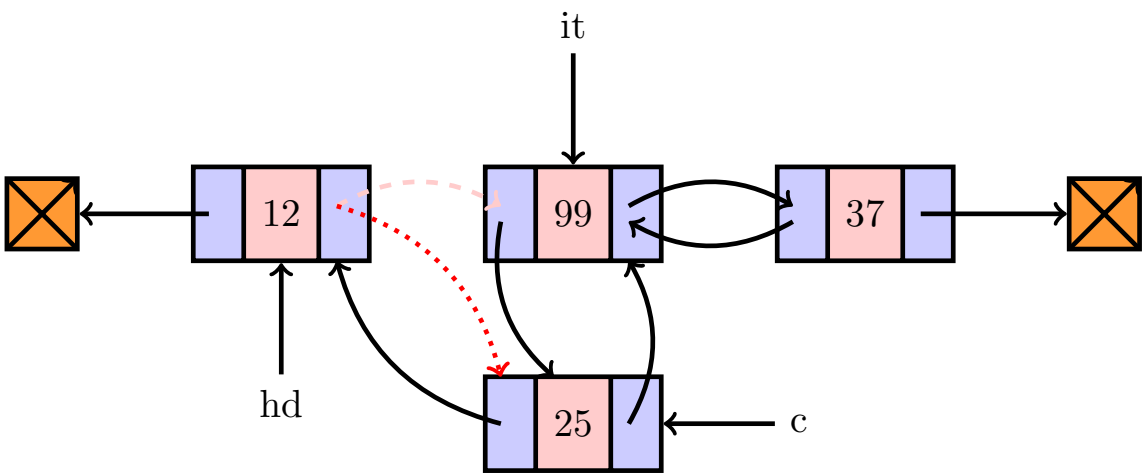
$$
\begin{aligned}
&c.prev = it.prev \\
&c.next = it \\
&it.prev = c \\
&\textbf{if } c.next \neq \textsc{Nil} \\
&\qquad c.next.prev = c
\end{aligned}
$$



Inserção antes da posição atual

Listas duplamente encadeadas

$$
\begin{aligned}
&c.prev = it.prev \\
&c.next = it \\
&it.prev = c \\
\textbf{if}\quad &c.prev \neq \textsc{Nil} \\
&\qquad c.prev.next = c
\end{aligned}
$$



Inserção antes da posição atual

Listas duplamente encadeadas

$$
\begin{aligned}
& c.prev = it.prev \\
& c.next = it \\
& it.prev = c \\
\textbf{if} \quad & c.prev \neq \textsc{Nil} \\
& \quad c.prev.next = c \\
\textbf{if} \ & it == hd \\
& \quad hd = c
\end{aligned}
$$

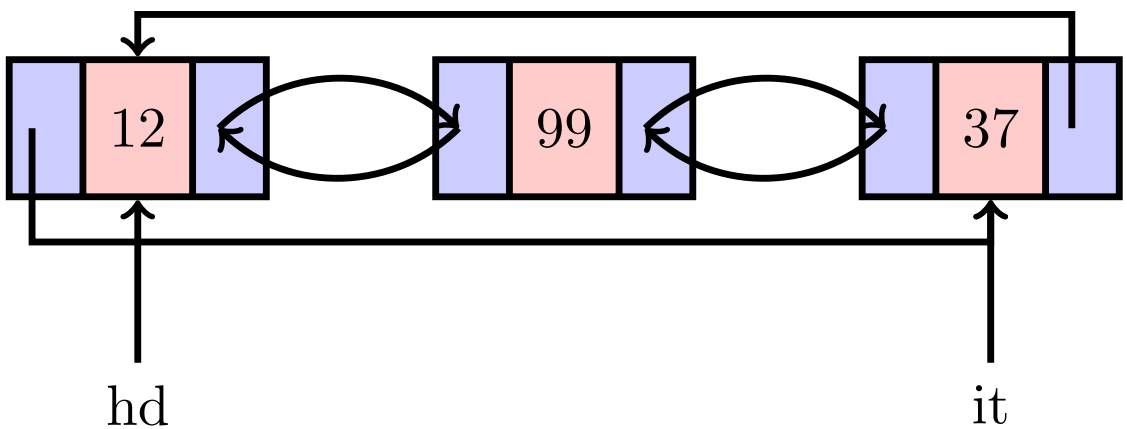
Listas duplamente encadeadas circulares

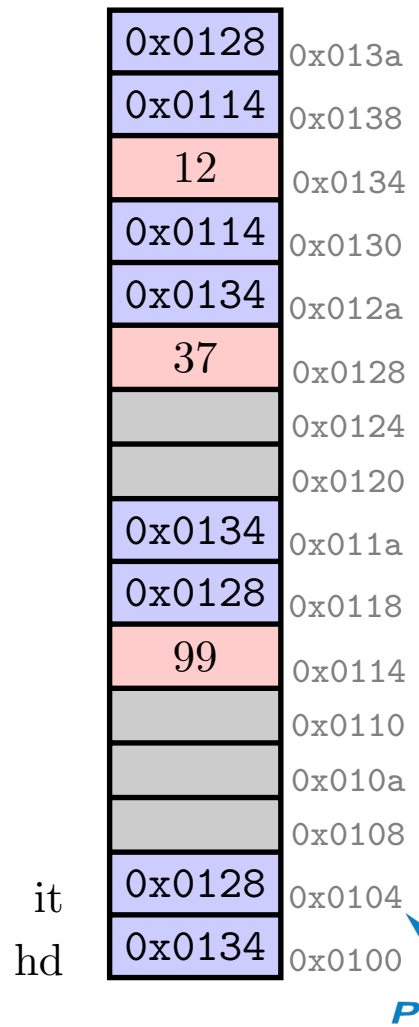
- a primeira célula referência a última

$$cell(v_1).prev = cell(v_n)$$

Implementação usando registros e ponteiros

Listas duplamente encadeadas circulares





Inserção

Listas duplamente encadeadas circulares

- Todas as células possuem sucessor e predecessor
- Algoritmos mais simples:
- Inserção de c após a posição it

$$c.next = it.next$$
$$c.prev = it$$
$$it.next = c$$
$$c.next.prev = c$$

- Inserção de c antes da posição it

$$c.prev = it.prev$$
$$c.next = it$$
$$it.prev = c$$
$$c.prev.next = c$$

PPgSC

Inserção

Listas duplamente encadeadas circulares

- Todas as células possuem sucessor e predecessor
- Algoritmos mais simples:
- Inserção de c após a posição it

$$c.next = it.next$$
$$c.prev = it$$
$$it.next = c$$
$$c.next.prev = c$$

- Inserção de c antes da posição it

$$c.prev = it.prev$$
$$c.next = it$$
$$it.prev = c$$
$$c.prev.next = c$$

- Exercício: identifique as dependências entre os quatro comandos formando cada operação de inserção.

- Remoção da célula na posição *it*

```
// quando a lista tem mais de um elemento
if it.next ≠ it
        c.next.prev = c.prev
        c.prev.next = c.next
        if it == hd
                hd = it.next
        // quando a lista tem um único elemento
else hd = NIL
FREE(it)
```

Elemento sentinela

- célula que não contem valor
- localizada no início ou no fim da lista
- simplifica os algoritmos de algumas operações
- consumo extra de memória

Formalização

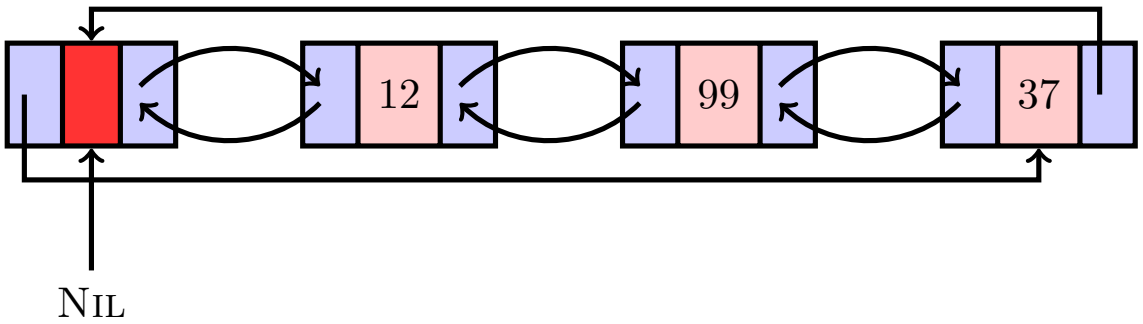
Elemento sentinela

- $l = \langle v_1, \ldots, v_n \rangle$
- $\forall i \cdot cell(v_i) \neq nil(l)$
- se $n \geq 1$
  - $nil(l).next = cell(v_1)$
  - $nil(l).prev = cell(v_n)$
  - $cell(v_1).prev = nil(l)$
  - $cell(v_n).next = nil(l)$
- se $n = 0$
  - $nil(l).next = nil(l)$
  - $nil(l).prev = nil(l)$

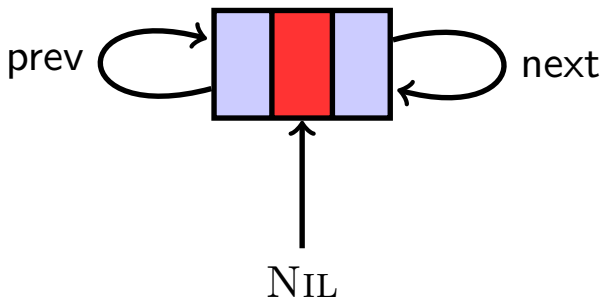
Ilustração

Elemento sentinela

$l = \langle 12, 99, 37 \rangle$



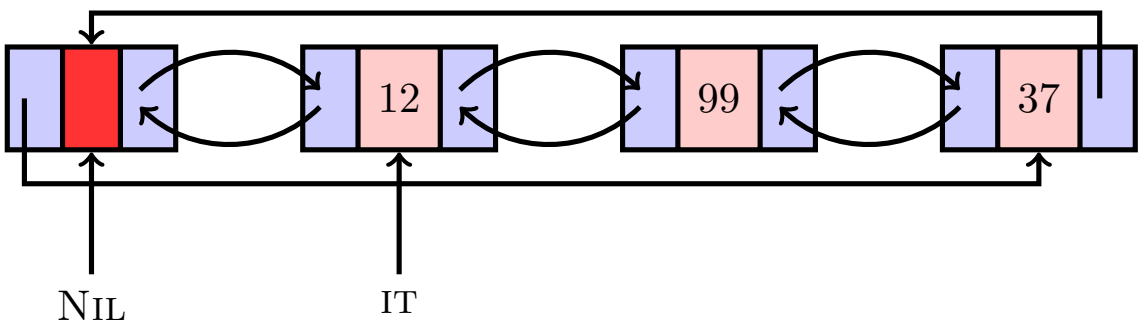
$l = \langle\rangle$



$it = Nil(l).next$
while $it \neq Nil(l)$
    // Processar $it.val$
    $it = it.next$

Remoção de um item na posição atual

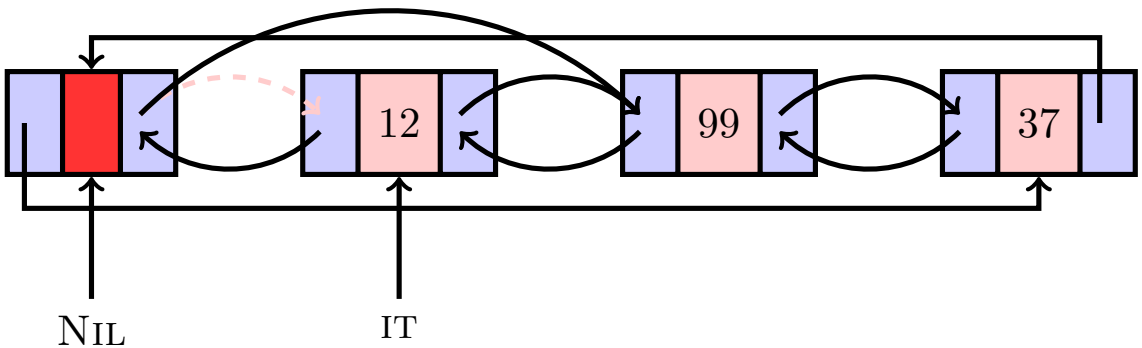
Listas duplamente encadeadas circulares com sentinela



Remoção de um item na posição atual

Listas duplamente encadeadas circulares com sentinela

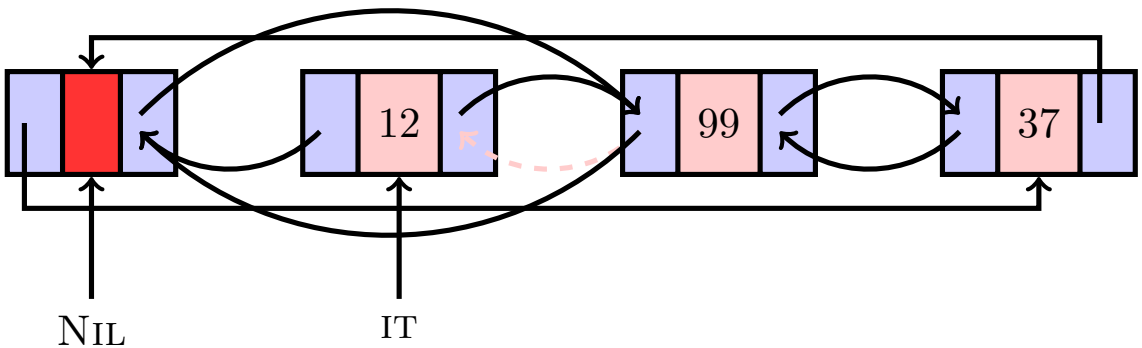
$$it.prev.next = it.next$$



Remoção de um item na posição atual

Listas duplamente encadeadas circulares com sentinela

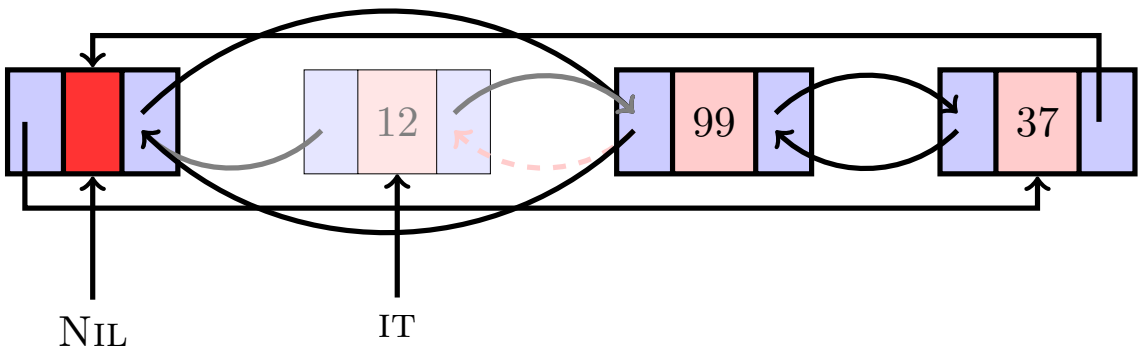
$$it.prev.next = it.next$$
$$it.next.prev = it.prev$$



Remoção de um item na posição atual

Listas duplamente encadeadas circulares com sentinela

$$it.prev.next = it.next$$
$$it.next.prev = it.prev$$
$$\textsc{Free}(it)$$



Exercícios

- Identifique qual tipo de lista é mais adequado para representar pilhas, filas e deques.
- Projete um algoritmo que, dada duas listas, concatena o conteúdo da segunda no fim da primeira.
  - Escolhe o tipo de lista que considerar mais adequado.

Listas encadeadas e arranjos

- O princípio de encadeamento não precisa ser restrito a ponteiros.
- Os atributos *next* (e *prev*) podem ser índices de um arranjo.
- Observação:
  - a memória do computador nada mais é que um arranjo, onde
  - os ponteiros são os índices deste arranjo.
- Nem todas as posições do arranjo são ocupadas
  - manter uma lista de posições livres.

PPgSC

Ilustração: lista simplesmente encadeada

$\langle 37, 12, 99 \rangle$

índice	valor	next
1		8
2		1
3	12	7
4		5
5		0
6	37	3
7	99	0
8		4

$hd = 6$
$free = 2$

if $free == 0$
    return
$n = free$
$free = A[free].next$
$A[n].next = hd$
$A[n].value = v$
$hd = n$

Exercício

- Escrever algoritmo para calcular tamanho de uma lista
- Escrever algoritmo para remover um valor
- Escrever um algoritmo para compactar a representação de uma lista em um arranjo: ao término, as posições ocupadas devem estar nas posições 1 até i.

PPgSC