

## Aula 3

### Estrutura de Dados

Prof. Vinicius Pozzobon Borin

### Conversa Inicial

- O objetivo desta aula é apresentar os conceitos que envolvem a estrutura de dados do tipo lista
- Será mostrado como realizar as manipulações dos dados dentro de uma estrutura de lista, como a construção, inserção e manipulação de dados

- Os tipos de listas e suas características serão
  - Lista simplesmente encadeada (circular e não circular)
  - Lista duplamente encadeada (circular e não circular)

- Também serão apresentadas outras duas estruturas de dados que podem ser construídas a partir de listas, mas que apresentam características únicas de operação
  - Pilhas
  - Filas

### Conceitos de listas encadeadas

- Existem estruturas de dados com alocação sequencial na memória.  
Exemplo: vetores, matrizes e registros
- Existem estruturas de dados com alocação não sequencial, ou seja, cada dado da estrutura pode estar posicionado em qualquer parte da memória destinada ao programa em execução

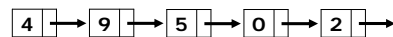
- Como o programa consegue localizar cada dado da estrutura, uma vez que eles estão posicionados aleatoriamente na memória?
  - Solução: ponteiros

- Não existe o conceito de índice em uma lista
- Na lista, conhecemos somente o endereço do primeiro elemento
- Sem índice, como acessamos cada dado da lista? Na lista, cada elemento contém o endereço do próximo

#### Vetor

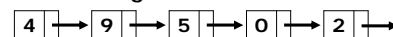
| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 4 | 9 | 5 | 0 | 2 |

#### Lista

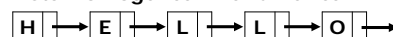


- Vantagem da lista
  - Alocação dinâmica de memória
- Desvantagem da lista
  - Tempo de acesso ao elemento de um vetor:  $O(1)$
  - Tempo de acesso ao elemento da lista:  $O(n)$

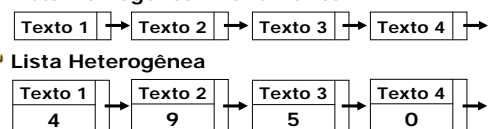
#### Lista Homogênea Numérica



#### Lista Homogênea Alfanumérica



#### Lista Heterogênea



- Aplicações
  - Agenda de contatos
  - Programa de solução de equações matemáticas
  - Reprodutor de música

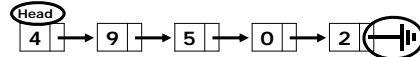
## Lista simplesmente encadeada (linked list)

### Listas encadeadas simples

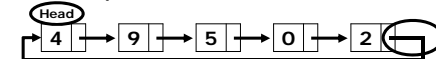
- Cada elemento (nó) de uma lista conterá um dado (ou um conjunto de dados) e um ponteiro contendo o endereço para o próximo elemento da lista
- A lista encadeada simples funciona como se fosse uma via de mão única
- Sendo assim, cada elemento conhece somente o seu subsequente, não sendo possível retornar para o elemento imediatamente anterior

### Criação dos elementos da lista

- Lista Simplesmente Encadeada e não Circular

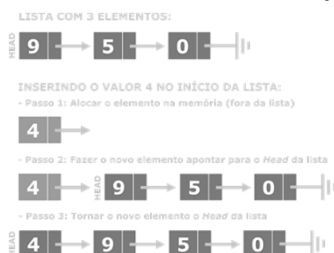


- Lista Simplesmente Encadeada e Circular



```
registro ElementoDaLista_Simples
dado: inteiro
prox: ElementoDaLista[->]
fimregistro
```

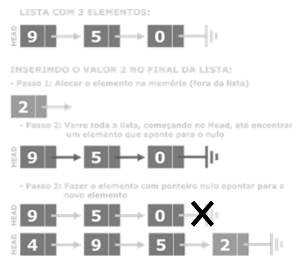
### Inserindo no início da lista simples



### Inserindo no início da lista simples

```
1 //Cria um Procedimento que recebe como parâmetro o dado a ser inserido no início
2 função InsereInicio (numero inteiro); sem retorno
3 var
4 //Declara um novo nó do tipo REGISTRO
5 NovoElemento[->]: registro ElementoDaLista_Simples
6 Inicio
7 //Insere no novo nó o dado recebido como parâmetro
8 NovoElemento->dado := numero
9
10 //Verifica se o HEAD está vazio
11 se (head = NULO) então
12 //Se HEAD está vazio, a lista está vazia!
13 //Portanto, novo elemento será o HEAD!
14 head := NovoElemento
15 head->prox := NULO
16 senão
17 //Se HEAD não está vazio, existem dados na lista
18 //Portanto, novo elemento aponta para o HEAD
19 NovoElemento->prox := head
20 //Novo elemento vira o HEAD da lista
21 head := NovoElemento
22 fimse
23 fimfunção
```

## Inserindo no final da lista simples



## Inserindo no final da lista simples

```

1 //Cria um procedimento que recebe como parâmetro o dado a ser inserido na lista
2 função InsereSim (numero inteiro): sem retorno
3 var
4     NovoElemento: registro ElementoDaLista_Simples
5     NovoElemento->registro ElementoDaLista_Simples
6     //Declara um novo nó da lista
7     ElementoVarredura->registro ElementoDaLista_Simples
8     //Declara um novo nó da lista
9     //Declara um novo nó da lista
10    //Declara um novo nó da lista
11    //Declara um novo nó da lista
12    //Declara um novo nó da lista
13    //Declara um novo nó da lista
14    //Declara um novo nó da lista
15    //Declara um novo nó da lista
16    //Declara um novo nó da lista
17    //Declara um novo nó da lista
18    //Declara um novo nó da lista
19    //Declara um novo nó da lista
20    //Declara um novo nó da lista
21    //Declara um novo nó da lista
22    //Declara um novo nó da lista
23    //Declara um novo nó da lista
24    //Declara um novo nó da lista
25    //Declara um novo nó da lista
26    //Declara um novo nó da lista
27    //Declara um novo nó da lista
28    //Declara um novo nó da lista
29    //Declara um novo nó da lista
30    //Declara um novo nó da lista
31    //Declara um novo nó da lista
32    //Declara um novo nó da lista
33    //Declara um novo nó da lista
34    //Declara um novo nó da lista
35    //Declara um novo nó da lista
36    //Declara um novo nó da lista
37    //Declara um novo nó da lista
38    //Declara um novo nó da lista
39    //Declara um novo nó da lista
40    //Declara um novo nó da lista
41    //Declara um novo nó da lista
42    //Declara um novo nó da lista
43    //Declara um novo nó da lista
44    //Declara um novo nó da lista
45    //Declara um novo nó da lista
46    //Declara um novo nó da lista
47    //Declara um novo nó da lista
48    //Declara um novo nó da lista
49    //Declara um novo nó da lista
50    //Declara um novo nó da lista
51    //Declara um novo nó da lista
52    //Declara um novo nó da lista
53    //Declara um novo nó da lista
54    //Declara um novo nó da lista
55    //Declara um novo nó da lista
56    //Declara um novo nó da lista
57    //Declara um novo nó da lista
58    //Declara um novo nó da lista
59    //Declara um novo nó da lista
60    //Declara um novo nó da lista
61    //Declara um novo nó da lista
62    //Declara um novo nó da lista
63    //Declara um novo nó da lista
64    //Declara um novo nó da lista
65    //Declara um novo nó da lista
66    //Declara um novo nó da lista
67    //Declara um novo nó da lista
68    //Declara um novo nó da lista
69    //Declara um novo nó da lista
70    //Declara um novo nó da lista
71    //Declara um novo nó da lista
72    //Declara um novo nó da lista
73    //Declara um novo nó da lista
74    //Declara um novo nó da lista
75    //Declara um novo nó da lista
76    //Declara um novo nó da lista
77    //Declara um novo nó da lista
78    //Declara um novo nó da lista
79    //Declara um novo nó da lista
80    //Declara um novo nó da lista
81    //Declara um novo nó da lista
82    //Declara um novo nó da lista
83    //Declara um novo nó da lista
84    //Declara um novo nó da lista
85    //Declara um novo nó da lista
86    //Declara um novo nó da lista
87    //Declara um novo nó da lista
88    //Declara um novo nó da lista
89    //Declara um novo nó da lista
90    //Declara um novo nó da lista
91    //Declara um novo nó da lista
92    //Declara um novo nó da lista
93    //Declara um novo nó da lista
94    //Declara um novo nó da lista
95    //Declara um novo nó da lista
96    //Declara um novo nó da lista
97    //Declara um novo nó da lista
98    //Declara um novo nó da lista
99    //Declara um novo nó da lista
100   //Declara um novo nó da lista

```

## Inserindo no meio da lista simples



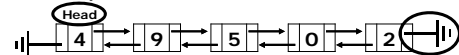
## Lista duplamente encadeada (doubly linked list)

## Listas encadeadas duplas

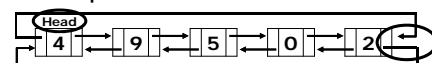
- Cada elemento (nó) de uma lista conterá um dado (ou um conjunto de dados) e um ponteiro contendo o endereço para o próximo e o elemento anterior da lista
- A lista encadeada dupla funciona como se fosse uma via de mão dupla
- Sendo assim, cada elemento conhece o seu subsequente e seu antecessor

## Criação dos elementos da lista

### ■ Lista Duplamente Encadeada e não Circular



### ■ Lista Duplamente Encadeada e Circular



- ☐ Ponteiro para o próximo
- ☐ Ponteiro para o anterior

```

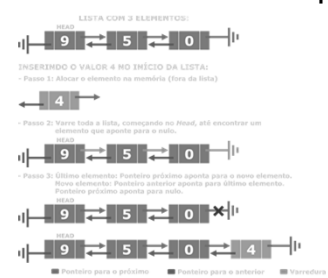
registro ElementoDaLista_Dupla
dado: inteiro
prox: ElementoDaLista[->]
ant: ElementoDaLista[->]
fimregistro

```

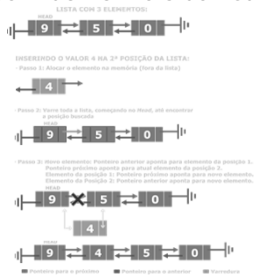
### Inserindo no início da lista dupla



### Inserindo no final da lista dupla



### Inserindo no meio da lista dupla



### Pilha (*stack*)

### First in last out (FILO)

- Em uma pilha, só é possível inserir um elemento ou remover esse elemento do topo da pilha
- O primeiro elemento que entra é o último que sai



### First in last out (FILO)



## Criação dos elementos da pilha

- Podemos implementar utilizando vetores ou listas
- Para lista, temos

```
registro Pilha
  dado: inteiro
  prox: Pilha[->)
fimregistro
```

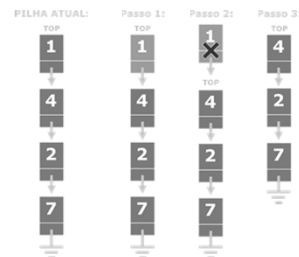
## Inserindo na pilha (*push*)



## Inserindo na pilha (*push*)

```
24 função push (numero: inteiro)
25 var
26   ElementoNovo: Pilha[->) ← PASSO 1
27 início
28   //Insere o valor no novo elemento que será inserido na pilha
29   NovoElemento->dado = numero ← PASSO 2
30   //Verifica que existe algo na pilha
31   se (Top == NULO) então ← PASSO 3
32     //Se a pilha está vazia, o novo elemento será a pilha
33     //e apontará para nulo
34     NovoElemento->prox = NULO
35   senão
36     //Se a pilha já tem algo, novo elemento aponta para o topo
37     NovoElemento->prox = Top ← PASSO 2
38   fimse
39   //Novo elemento vira o topo, pois a inserção é sempre no topo
40   Top = NovoElemento ← PASSO 3
41 fimfunção
```

## Removendo da pilha (*pop*)



## Removendo da pilha (*pop*)

```
43 //Não passa nenhum valor como parâmetro,
44 //pois a remoção é sempre do valor do topo
45 função pop ()
46 var
47   ElementoParaRemover: Pilha[->) ← PASSO 1
48 início
49   //Verifica que existe algo na pilha
50   se (Top <> NULO) então ← PASSO 2
51     //Existe algo para remover então
52     //Salva temporariamente o topo atual
53     ElementoParaRemover = Top
54     //Incrementa o top (passa para o próximo nó)
55     Top = Top->prox ← PASSO 2
56     //Limpa da memória o topo antigo
57     libera(ElementoParaRemover) ← PASSO 3
58   fimse
59 fimfunção
```

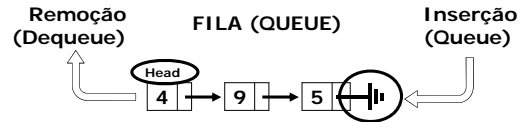
## Fila (*queue*)

### First in first out (FIFO)

- Só é possível inserir um elemento no final da fila
- Só é possível remover um elemento do início da fila
- O primeiro elemento que entra é o primeiro que sai



### First in first out (FIFO)



### Criação dos elementos da fila

- Podemos implementar utilizando vetores ou listas
- Para lista, temos

```
registro Fila
  dado: inteiro
  prox: Fila[->]
fimregistro
```

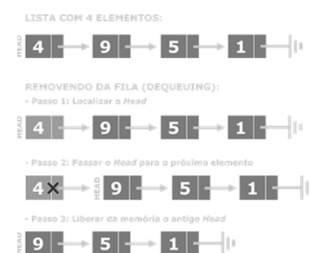
### Inserindo na fila (queuing)



### Inserindo na fila (queuing)

```
24 função queue (numero inteiro)
25 var
26   ElementoNovo: Fila[->] ← PASSO 1
27   ElementoVarredura: Fila[->]
28   inicio
29   //Insere o valor no novo elemento que será inserido na fila
30   NovoElemento->dado ← numero
31   //Verifica se existe algo na fila
32   se (Head = NULO) então
33     //Se a fila está vazia, o novo elemento vira o Head
34     Head ← NovoElemento
35   senão
36     //Se a fila já tem algo, novo elemento entra no final
37     ElementoVarredura ← Head ← PASSO 2
38     //Varre um elemento por vez procurando o ponteiro nulo
39     enquanto (ElementoVarredura->prox ≠ NULO)
40       ElementoVarredura ← ElementoVarredura->prox
41     fimenquanto
42     //Após encontrar o ponteiro nulo,
43     //fixa o último elemento da fila apontar para o novo nó
44     ElementoVarredura->prox ← NovoElemento
45     //Novo nó agora terá o ponteiro nulo (final da lista)
46     NovoElemento->prox ← NULO ← PASSO 3
47   fimse
48 fimfunção
```

### Removendo da fila (dequeuing)



### Removendo da fila (*dequeuing*)

```
50 //Não passa nenhum valor como parâmetro,  
51 //pois a remoção é sempre do valor do head  
52 função dequeue ()  
53 var  
54     ElementoParaRemover: Fila[->) ← PASSO 1  
55 início  
56     //Verifica que existe algo na pilha  
57     se (Head <> NULO) então  
58         //Existe algo para remover então  
59         //Salva temporariamente o head atual  
60         ElementoParaRemover = Head  
61         //Incrementa o head (passa para o próximo nó)  
62         Head = Head->prox ← PASSO 2  
63         //Limpa da memória o head antigo  
64         libera(ElementoParaRemover) ← PASSO 3  
65     fimse  
66 fimfunção
```

### Referências

- ASCENCIO, A. F. G. Estrutura de dados: algoritmos, análise da complexidade e implementações em Java e C/C++. São Paulo: Pearson, 2011.
- \_\_\_\_\_. Fundamentos da programação de computadores: Algoritmos, Pascal, C/C++ (padrão ANSI) JAVA. 3. ed. São Paulo: Pearson, 2012.
- CORMEN, T. H. Algoritmos: teoria e prática. 3. ed. Rio de Janeiro: Elsevier, 2012.

- PUGA, S.; RISSETI, G. Lógica de programação e estrutura de dados. 3. ed. São Paulo: Pearson, 2016.
- MIZRAHI, V. V. Treinamento em linguagem C. 2. ed. São Paulo: Pearson, 2008.
- LAUREANO, M. Estrutura de dados com algoritmos e C. São Paulo: Brasport, 2008.