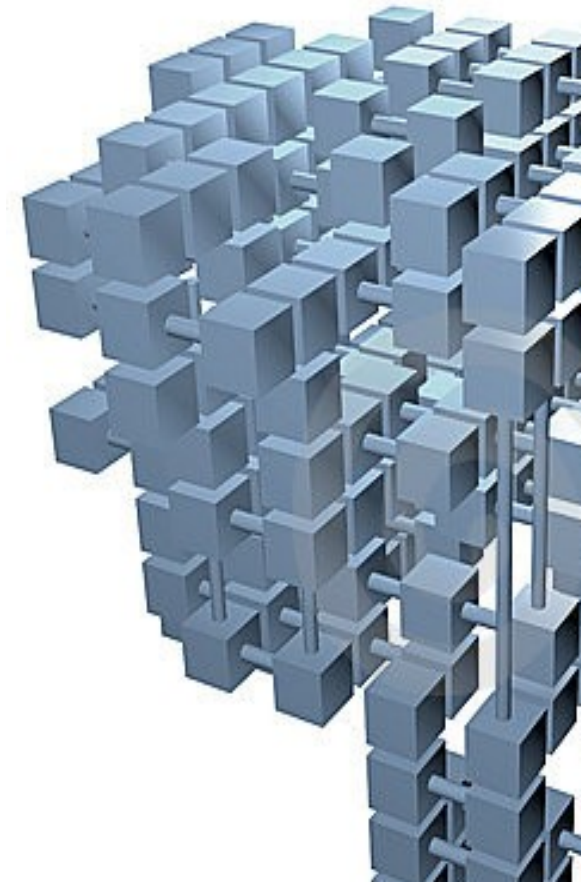


SISTEMAS DE INFORMAÇÃO

Estrutura de Dados 1

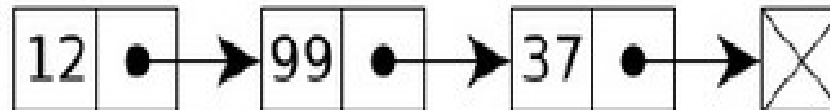
Listas Encadeadas

Prof. Ivan José dos Reis Filho
ivanfilhoreis@gmail.com



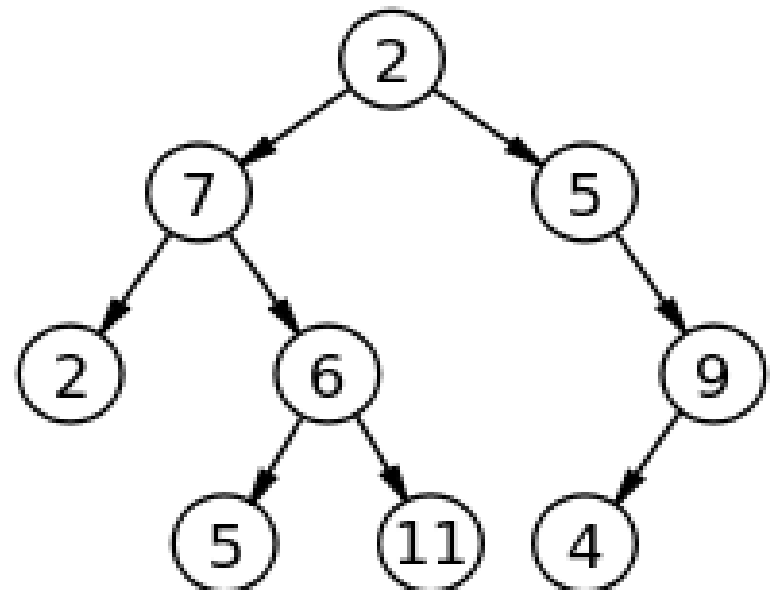
Lista encadeada

- O que é?
 - É um tipo abstrato de dados
 - Organizar os dados de forma **Linear** ou **Dinâmico**
 - Pode pensar visualmente assim:



Como assim Linear?

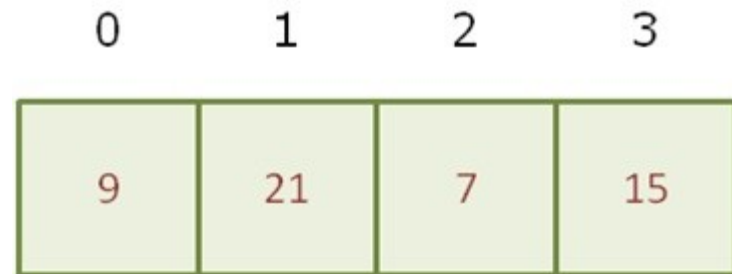
- Existem estruturas de dados em que os elementos estão em níveis diferentes
- Por exemplo, um árvore binária



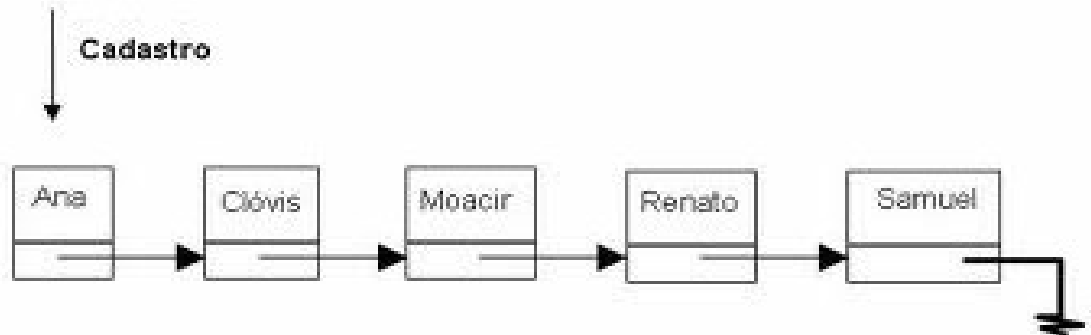
Como assim Dinâmico?

- **Vetores** são estruturas que precisam ter seu tamanho bem definidos!

```
int num[3];
```



- Lista podem crescer até a memória estourar!



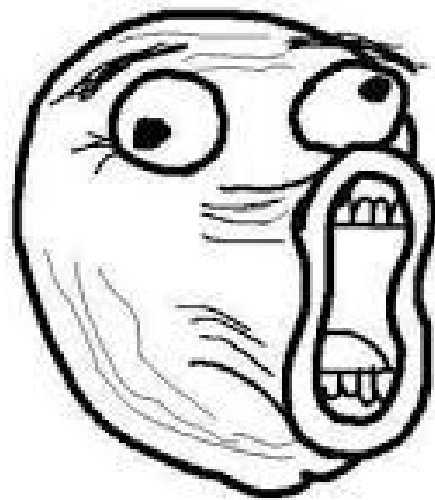
Motivação

- Para quer usar?

Motivação

- Para quer usar?

Tirar nota 10



LOL

Motivação

- Para quer usar?
 - Listas crescem e decrescem a medida em que o programa precisa!
 - Ganha mais espaço na memória!
 - Fácil de Implementar em relação as outras TAD's

Lista - Definição

- Sequência de um ou mais itens

x_1, x_2, \dots, x_n , na qual x_i é de um determinado tipo e n representa o tamanho da lista linear.

- Sua principal propriedade estrutural envolve as posições relativas dos itens em um dimensão

Assumindo $n \geq 1$, x_1 é o primeiro item da lista e x_n é o último item da lista.

x_i precede x_{i+1} para $i = 1, 2, \dots, n - 1$

x_i sucede x_{i-1} para $i = 2, 3, \dots, n$

o elemento x_i é dito estar na i -ésima posição da lista.

Lista - Definição

- Uma estrutura de dados Lista é um conjunto de dados dispostos e/ou acessíveis em um sequência determinada
 - Este conjunto de dados pode possuir uma ordem intrínseca (Lista ordenada) ou não;
 - Este conjunto de dados pode ocupar espaços de memória fisicamente consecutivos, espelhando a sua ordem, ou não.
 - Se os dados estiverem dispersos fisicamente, para este conjunto seja uma lista, ele deve possuir operações adicionais que permitam que seja tratado como tal.

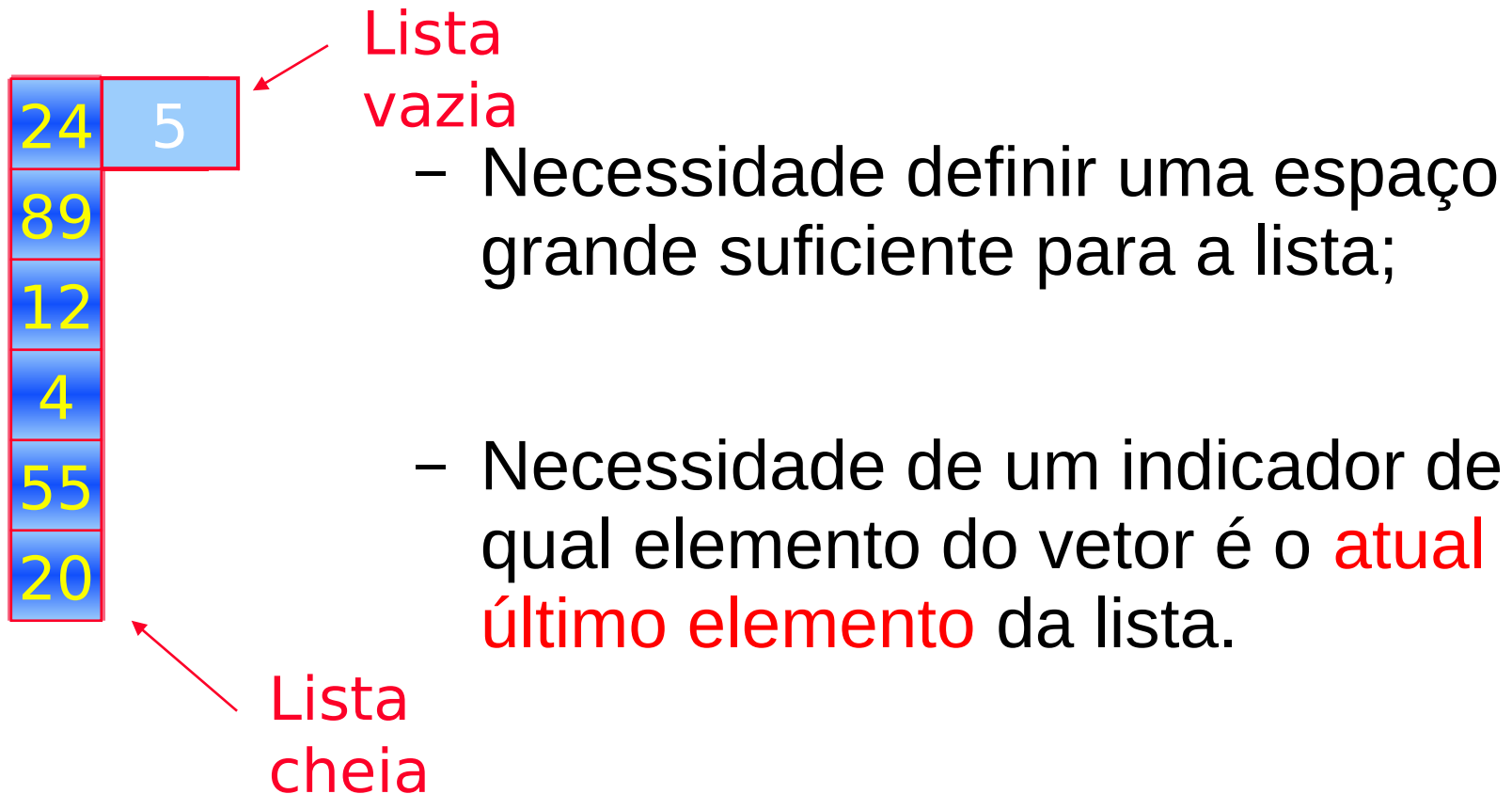
Listas

4
12
20
24
55
89

- A lista é uma estrutura de dados
 - Inspirado numa lista natural
- A lista pode ser ordenada ou não
 - Quando ordenada, pode ser por alguma **característica intrínseca**.
 - Pode também refletir a **ordem cronológica**.

Listas usando vetores – Listas estáticas

- Vetores possuem um espaço limitado para armazenamento.



- Necessidade definir uma espaço grande suficiente para a lista;
- Necessidade de um indicador de qual elemento do vetor é o **atual último elemento** da lista.

Modelagem de listas estáticas utilizando algoritmo estruturado

- Modelaremos a estrutura de dados Lista utilizando a técnica da programação estruturada e como forma de armazenamento um Vetor.

Veremos por enquanto apenas o algoritmo.

Modelagem da Lista

- Aspecto estrutural
 - Necessitamos de um vetor para armazenar as informações
 - Necessitamos de um indicador da posição atual do **último elemento** da lista.
 - Necessitamos de uma constante que nos diga quando a lista está cheia e duas outras para codificar erros.
 - Criaremos uma lista estática de inteiros para exemplo.

```
constantes MaxLista = 100;  
tipo Lista{  
    inteiro dados [MaxLista];  
    inteiro ultimo;  
}nomeTipoLista;
```

Modelagem da Lista

- Aspecto Funcional
 - Colocar e retirar dados da lista.
 - Testar se a lista está vazia ou cheia e outros testes.
 - Inicializá-la e garantir a ordem do elementos

Modelagem da Lista

- Exemplo de Operações: Colocar e retirar dados da lista:
 - Adciona(dado)
 - AdcionaNoInicio(dado)
 - AdicionaNaPosição(dado, posição)
 - AdcionaEmOrdem(dado)
 - Retira()
 - RetiraDoInicio()
 - RetiraDaPosição(posição)
 - RetiraEspecifico(dado)

Modelagem da Lista

- Exemplo de Operações: Testar a lista e outros testes:
 - ListaCheia
 - ListaVazia
 - Posicao(dado)
 - Contem(dado)
 - Igual(dado1, dado2)
 - Maior(dado1, dado2)
 - Menor(dado1, dado2)
- Exemplo de Operações: Inicializar ou limpar:
 - InicializaLista;
 - DestroiLista;

Algoritmo inicializaLista

```
FUNÇÃO inicializaLista()  
    início  
        aLista.ultimo := -1;  
    Fim;
```

Obs: Este e os próximos algoritmos pressupõem que foi definida uma variável global tipo Lista denominada aLista.

Algoritmo DestroiLista

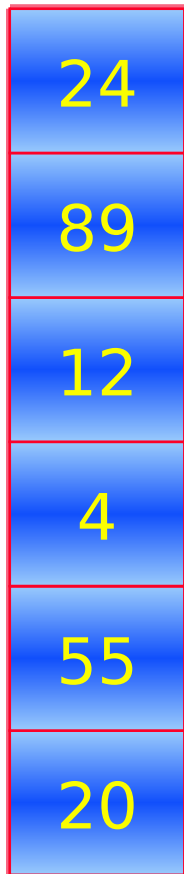
```
FUNÇÃO destroiLista()  
    início  
        aLista.ultimo := -1;  
    fim;
```

- **Obs:** Este algoritmo parece redundante. Colocar a sua semântica em separado da inicialização, porém, é importante. Mais tarde veremos que, utilizando alocação dinâmica de memória, possuir um destrutor para a lista é muito importante.

Algoritmo ListaVazia

```
Booleano FUNÇÃO listaVazia()  
    início  
        SE (aLista.ultimo = -1) ENTÃO  
            RETORNE(Verdade)  
        SENÃO  
            RETORNE(Falso);  
    fim;
```

Algoritmo Adiciona



24
89
12
4
55
20

- Procedimento
 - Testamos se há espaço
 - Incrementamos o último
 - Adcionamos o novo dado
- Parâmetros
 - O dado a ser inserido
 - Lista (Global)

Algoritmo Adiciona

Constantes

```
    ErroListaCheia    = -1;  
    ErroListaVazia    = -2;  
    ErroPosição       = -3;
```

Inteiro FUNÇÃO adiciona(inteiro dado)

início

SE (listaCheia) ENTÃO

RETORNE(ErroListaCheia);

SENÃO

aLista.ultimo := aLista.ultimo + 1.

aLista.dados[aLista.ultimo] := dado;

RETORNE(aLista.ultimo);

FIM SE

fim;

DÚVIDAS ?



Algoritmo Retira

- Procedimento
 - Testamos se há elementos
 - Decrementamos o último.
 - Devolvemos o último elemento.
- Parâmetros
 - Lista(global)

Algoritmo Retira

```
Inteiro FUNÇÃO retira()  
    início  
        SE (listaVazia) ENTÃO  
            RETORNE(ErroListaVazia);  
        SENÃO  
            aLista.ultimo := aLista.ultimo - 1.  
  
            RETORNE(aLista.dados[aLista.ultimo+1]);  
  
        FIM SE  
    fim;
```

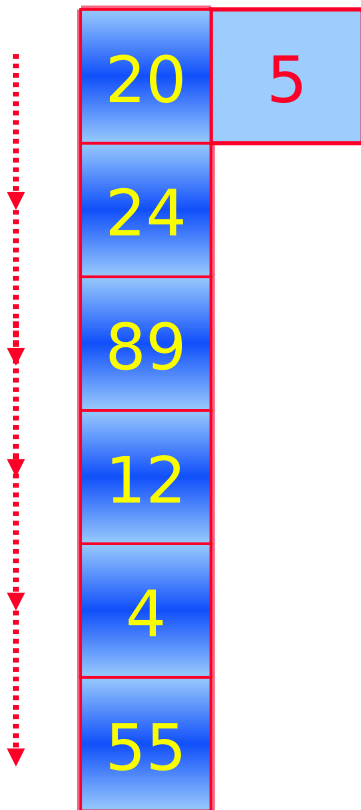

Algoritmo AdicionaNoInicio

- Procedimento

- Testamos se há espaço.
- Incrementamos o último
- Empurramos tudo para trás
- Adicionamos o novo dado na primeira posição

- Parâmetros

- O dado a ser inserido
- Lista (global)



Algoritmo AdicionaNoInicio

```
Inteiro FUNÇÃO adicionaNoInicio(inteiro dado)
variáveis
    inteiro posição; //“Variável auxiliar para percorrer”
início
    SE (listaCheia) ENTÃO
        RETORNE(ErroListaCheia);
    SENÃO
        aLista.ultimo := aLista.ultimo + 1;
        posição := aLista.ultimo;
        ENQUANTO (posição > 0) FAÇA
            //“Empurrar tudo para tras”
            aLista.dados[posicao] := aLista.dados[posicao-1];
            posição := posição - 1;
        FIM ENQUANTO
        aLista.dados[0] := dado;
        RETORNE(0);
    FIM SE
fim;
```

Algoritmo AdicionaNoInicio

```
Inteiro FUNÇÃO adicionaNoInicio(inteiro dado)
variáveis
    inteiro posição; //“Variável auxiliar para percorrer”
início
    SE (listaCheia) ENTÃO
        RETORNE(ErroListaCheia);
    SENÃO
        aLista.ultimo := aLista.ultimo + 1;
        posição := aLista.ultimo;
        ENQUANTO (posição > 0) FAÇA
            //“Empurrar tudo para tras”
            aLista.dados[posicao] := aLista.dados[posicao-1];
            posição := posição - 1;
        FIM ENQUANTO
        aLista.dados[0] := dado;
        RETORNE(0);
    FIM SE
fim;
```

Algoritmo AdicionaNoInicio

```
Inteiro FUNÇÃO adicionaNoInicio(inteiro dado)
  variáveis
    inteiro posição; //“Variável auxiliar para percorrer”
  início
    SE (listaCheia) ENTÃO
      RETORNE(ErroListaCheia);
    SENÃO
      aLista.ultimo := aLista.ultimo + 1;
      posição := aLista.ultimo;
      ENQUANTO (posição > 0) FAÇA
        //“Empurrar tudo para tras”
        aLista.dados[posicao] := aLista.dados[posicao-1];
        posição := posição - 1;
      FIM ENQUANTO
      aLista.dados[0] := dado;
      RETORNE(0);
    FIM SE
  fim;
```

Algoritmo RetiraDoInício

- Procedimento

20

- Testamos se há elementos
- Decrementamos o último
- Salvamos o primeiro elemento
- Empurramos tudo para a frente

- Parâmetros

- Lista(Global)

24	4
89	
12	
4	
55	

Algoritmo RetiraDoInício

```
Inteiro FUNÇÃO retiraDoInício()  
  variáveis  
    Inteiro posição, valor;  
  início  
    SE (listaVazia) ENTÃO  
      RETORNE(ErroListaVazia);  
    SENÃO  
      aLista.ultimo := aLista.ultimo - 1;  
      valor := aLista.dados[0];  
      posição := 0;  
      ENQUANTO (posição <= aLista.ultimo) FAÇA  
        //“Empurrar tudo para traz”  
        aLista.dados[posicao] := aLista.dados[posicao + 1];  
        posição := posição + 1;  
      FIM ENQUANTO  
  
      RETORNE(valor);  
  
    FIM SE  
  fim;
```

Algoritmo RetiraDoInício

```
Inteiro FUNÇÃO retiraDoInício()
    variáveis
        Inteiro posição, valor;
    início
        SE (listaVazia) ENTÃO
            RETORNE(ErroListaVazia);
        SENÃO
            aLista.ultimo := aLista.ultimo - 1;
            valor := aLista.dados[0];
            posição := 0;
            ENQUANTO (posição <= aLista.ultimo) FAÇA
                //“Empurrar tudo para traz”
                aLista.dados[posicao] := aLista.dados[posicao + 1];
                posição := posição + 1;
            FIM ENQUANTO

            RETORNE(valor);

        FIM SE
    fim;
```

Algoritmo AdicionaNaPosição

- Procedimento
 - Testamos se há espaço e se a posição existe
 - Incrementamos o último
 - Empurramos tudo para trás a partir da posição
 - Adicionamos o novo dado na posição
- Parâmetros
 - O dado a ser inserido
 - A posição onde inserir
 - Lista (global)

Algoritmo AdcionaNaPosição

```
Inteiro FUNÇÃO adicionaNaPosição(inteiro dado, destino)
variáveis
    inteiro posição;
início
    SE (listaCheia) ENTÃO
        RETORNE(ErroListaCheia);
    SENÃO
        SE (destino > aLista.ultimo + 1) ENTÃO
            RETORNE(ErroPosição);
    FIM SE

    aLista.ultimo := aLista.ultimo + 1;
    posição := aLista.ultimo;
    ENQUANTO (posição > destino) FAÇA
        // "Empurrar tudo para tras"
        aLista.dados[posicao] := aLista.dados[posicao - 1];
        posição := posição - 1;
    FIM ENQUANTO
    aLista.dados[destino] := dado;
    RETORNE(destino);
    FIM SE
fim;
```

Algoritmo AdcionaNaPosição

```
Inteiro FUNÇÃO adicionaNaPosição(inteiro dado, destino)
variáveis
    inteiro posição;
início
    SE (listaCheia) ENTÃO
        RETORNE(ErroListaCheia);
    SENÃO
        SE (destino > aLista.ultimo + 1) ENTÃO
            RETORNE(ErroPosição);
    FIM SE

    aLista.ultimo := aLista.ultimo + 1;
    posição := aLista.ultimo;
    ENQUANTO (posição > destino) FAÇA
        // "Empurrar tudo para tras"
        aLista.dados[posicao] := aLista.dados[posicao - 1];
        posição := posição - 1;
    FIM ENQUANTO
    aLista.dados[destino] := dado;
    RETORNE(destino);
    FIM SE
fim;
```

Algoritmo RetiraDaPosição

- Procedimento
 - Testamos se há elementos e se a posição existe
 - Decremento o último
 - Salvamos elemento na posição
 - Empuramos tudo para frente até a posição
- Parâmetros
 - A posição a retirar o valor
 - Lista (global)

Algoritmo RetiraDaPosição

```
Inteiro FUNÇÃO retiraDaPosição(inteiro fonte)
  variáveis
  Inteiro posição, valor;
  início
    SE (fonte > aLista.ultimo) ENTÃO
      RETORNE(ErroPosição);
    SENÃO
      SE (listaVazia) ENTÃO
        RETORNE(ErroListaVazia);
      SENÃO
        aLista.ultimo := aLista.ultimo - 1;
        valor := aLista.dados[fonte];
        posição := fonte;
        ENQUANTO (posição <= aLista.ultimo) FAÇA
          // "Empurrar tudo para frente"
          aLista.dados[posicao] := aLista.dados[posicao + 1];
          posição := posição + 1;
        FIM ENQUANTO
        RETORNE(valor);
      FIM SE
    FIM SE
  fim;
```

Algoritmo RetiraDaPosição

```
Inteiro FUNÇÃO retiraDaPosição(inteiro fonte)
  variáveis
  Inteiro posição, valor;
  início
    SE (fonte > aLista.ultimo) ENTÃO
      RETORNE(ErroPosição);
    SENÃO
      SE (listaVazia) ENTÃO
        RETORNE(ErroListaVazia);
      SENÃO
        aLista.ultimo := aLista.ultimo - 1;
        valor := aLista.dados[fonte];
        posição := fonte;
        ENQUANTO (posição <= aLista.ultimo) FAÇA
          // "Empurrar tudo para frente"
          aLista.dados[posicao] := aLista.dados[posicao + 1];
          posição := posição + 1;
        FIM ENQUANTO
        RETORNE(valor);
      FIM SE
    FIM SE
  fim;
```

Algoritmo RetiraEspecifico

- Retira um dado específico na lista
- Procedimento
 - Testamos se há elementos
 - Testamos se o dado existe e qual sua posição
 - Necessitamos de uma função **Posição(dado)**
 - Chamamos **RetiraDaPosição**.
- Parâmetros
 - O dado a ser retirado
 - Lista (Global)

Algoritmo Posição

```
Inteiro FUNÇÃO posição(inteiro dado)
variáveis
inteiro posição;
início
    posição := 0;
    ENQUANTO (posição <= aLista.ultimo E
        NÃO(IGUAL(dado, aLista.dados[posição])))
        FAÇA
            posição := posição + 1;
    FIM ENQUANTO
    SE (posição > aLista.ultimo) ENTÃO
        RETORNE(ErroPosição);
    SENÃO
        RETORNE(posição);
    FIM SE
fim;
```

Algoritmo RetiraEspecífico

Inteiro FUNÇÃO retiraEspecífico(inteiro
dado)

variáveis
inteiro posição;

início
 SE (listaVazia) ENTÃO
 RETORNE(ErroListaVazia);
 SENÃO
 posição := posição(dado);
 SE (posição < 0) ENTÃO
 RETORNE(ErroPosição);
 SENÃO
 RETORNE(retiraDaPosição(posição));
 FIM SE
FIM SE
fim;

Algoritmo Contém(dado)

- Procedimento
 - Testamos se um dado específico existe na lista
- Parâmetros
 - O dado a ser procurado
 - Lista (global)

Algoritmo Contém

Booleano FUNÇÃO Contém(inteiro dado)

variáveis

inteiro posição;

Início

 posição := 0;

 ENQUANTO (posição <= aLista.ultimo E
 NÃO(IGUAL(dado, aLista.dados[posição])))

 FAÇA

 posição := posição + 1;

 FIM ENQUANTO

 SE (posição > aLista.ultimo) ENTÃO

 RETORNE(Falso);

 SENÃO

 RETORNE(Verdadeiro);

 FIM SE

fim;

DÚVIDAS ?

