

Estruturas de Dados I

Listas Estáticas

Estrutura de Dados Lista - Definição



■ Seqüência 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 uma 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.

Estrutura de Dados Lista - Definição



Uma Estrutura de Dados Lista é um conjunto de dados dispostos e/ou acessáveis em uma seqüência 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 que este conjunto seja uma lista, ele deve possuir operações e informações adicionais que permitam que seja tratado como tal (Lista Encadeada).

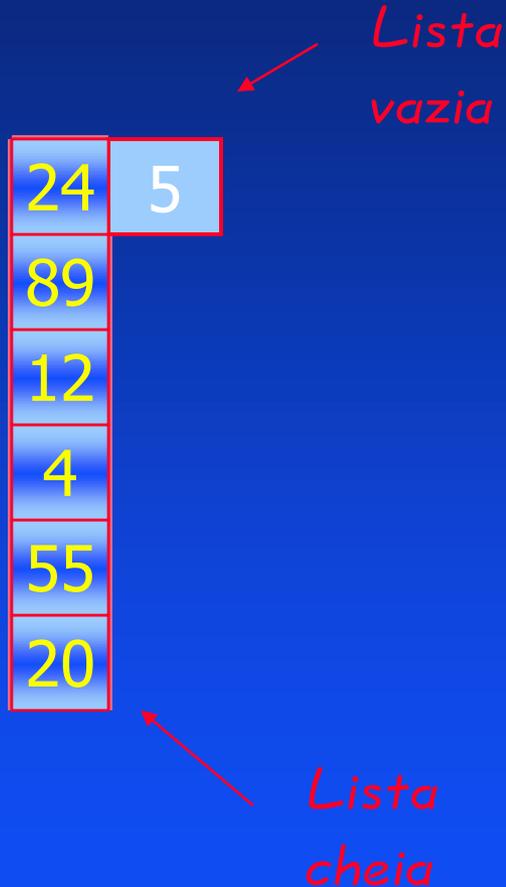
Listas

- A Lista é uma estrutura de dados cujo funcionamento é inspirado no de uma lista “natural”

4
12
20
24
55
89

- Uma lista pode ser Ordenada ou não.
 - Quando for ordenada, pode o ser por alguma **característica intrínseca** dos dados (ex: ordem alfabética).
 - Pode também refletir a **ordem cronológica** (ordem de inserção) dos dados).

Listas usando Vetores – Listas estáticas



- Vetores possuem um espaço limitado para armazenamento de dados.
- Precisamos definir um espaço grande o suficiente para a lista.
- Precisamos 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 (Array).
 - ◆ Veremos por enquanto somente os algoritmos.

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.

■ Algoritmo para o TAD:

```
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.
- ◆ Inicializa-la e garantir a ordem dos elementos.

Modelagem da Lista

- Exemplo de Operações: Colocar e retirar dados da lista:
 - ◆ Adiciona(dado)
 - ◆ AdicionaNoInício(dado)
 - ◆ AdicionaNaPosição(dado, posição)
 - ◆ AdicionaEmOrdem(dado)
 - ◆ Retira()
 - ◆ RetiraDoInício()
 - ◆ RetiraDaPosição(posição)
 - ◆ RetiraEspecífico(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;
```

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

Ou seja, foi declarada no início:
nometipolista **aLista**;

Algoritmo DestroiLista

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

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

Algoritmo ListaCheia

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

Pergunta: Existe tipo booleano em C?

Adote o padrão: 0 para falso, e diferente de 0 (-1 ou 1) para verdadeiro.

Algoritmo ListaVazia

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

Algoritmo Adiciona

24	5
89	
12	
4	
55	
20	

■ Procedimento:

- ◆ Testamos se há espaço.
- ◆ Incrementamos o último.
- ◆ Adicionamos 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;

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 AdicionaNoInício



■ Procedimento:

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

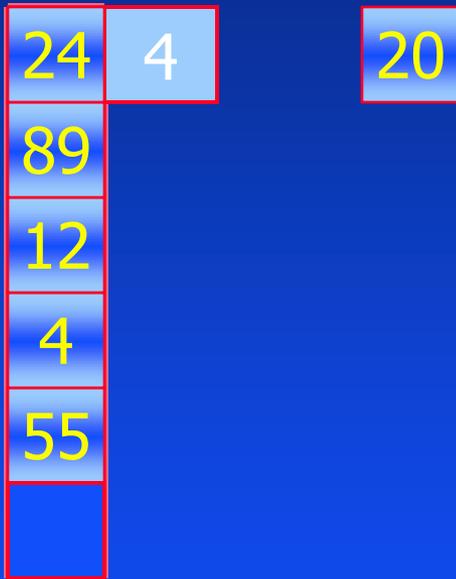
■ Parâmetros:

- ◆ O dado a ser inserido.
- ◆ Lista (global).

Algoritmo AdicionaNoInício

```
Inteiro FUNÇÃO adicionaNoInício(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:

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

■ Parâmetros:

- ◆ Lista (global).

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 frente”  
        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.
- ◆ Empuramos tudo para trás a partir da posição.
- ◆ Adicionamos o novo dado na posição dada.

■ Parâmetros:

- ◆ O dado a ser inserido.
- ◆ A posição onde inserir.
- ◆ Lista (global).

Algoritmo AdicionaNaPosição

```
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.
- ◆ Decrementamos o último.
- ◆ Salvamos elemento na posição.
- ◆ Empuramos tudo para frente até posição.

■ Parâmetros:

- ◆ O dado a ser inserido.
- ◆ A posição onde inserir.
- ◆ 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 **RetiraEspecífico**

- Retira um dado específico da 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;
```

Algoritmos Contem(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;
```

Exercícios

- Converter todos os algoritmos apresentados nesta aula em programas em C.