SCC-223 Estruturas de Dados I

Lista Linear Sequencial

Profa. Elaine Parros Machado de Sousa

ICMC USP SÃO CARLOS

Aula baseada em material dos professores Adenilso Simão e Rudinei Goularte.

Grupo de Bases de Dados e Imagens
USP - São Carlos

Lista Linear

- **Lista Linear** ⇒ estrutura que armazena uma sequência de elementos (itens) do mesmo tipo
  - sequência de zero ou mais elementos
    - $L = (e_1, e_2, ..., e_n)$
    - e_i é do tipo **T**
    - n representa o tamanho da lista
    - **ex:** lista de nomes, lista de CPFs, lista de emails, lista de URLs, ...

Lista Linear

- **Lista Linear**
  - ...
  - principal característica estrutural ⇒ **posição relativa dos elementos**
    - elementos podem ser linearmente dispostos de acordo com sua posição na lista
    - para $n \geq 1$, e_1 é o primeiro item e e_n é o último
    - e_{i+1} é sucessor de e_i

Lista Linear

- Diversos tipos de aplicação
  - Listas de compras
  - Itens de estoque
  - Notas de alunos
  - Lista telefônica
  - Lista de tarefas
  - Informações sobre funcionários de uma empresa
  - Grafos
  - Compiladores
  - Similações
  - ...

Lista Linear

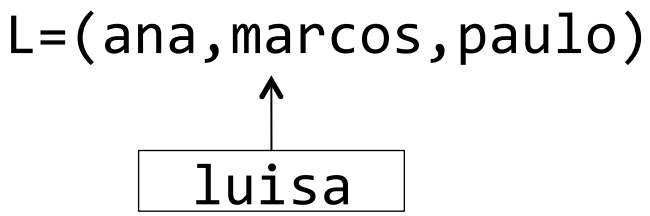
- Principais operações
  - Criar lista
  - Inserir item
  - Inserir item na *i*-ésima posição
  - Remover item
  - Remover *i*-ésimo item
  - Buscar item
  - Buscar *i*-ésimo item
  - Contar número de itens
  - Verificar se a lista está vazia
  - Verificar se a lista está cheia
  - Imprimir lista
  - ...

Lista Linear - Ordenação

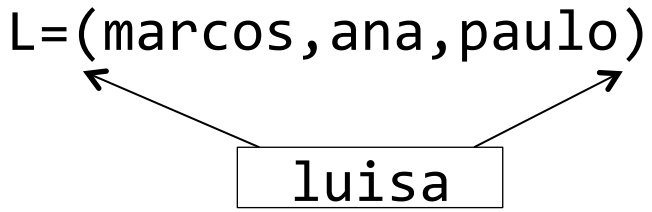
- **Lista ordenada** – posição dos elementos na lista é definida de acordo com algum critério
  - ordem alfabética
  - ordem numérica crescente ($e_i \leq e_{i+1}$)
  - ordem numérica decrescente ($e_i \geq e_{i+1}$)
  - …
- **Lista não ordenada** – não existe critério para definir a posição dos elementos na lista

Lista Linear - Ordenação

- **Lista ordenada vs. Lista não ordenada**
  - característica que afeta, principalmente, operações de **inserção**, **busca** e **remoção**
  - **Lista ordenada**
    - inserção realizada em posição definida pela ordenação



  - **Lista não ordenada**
    - inserção realizada nas extremidades (baixo custo)



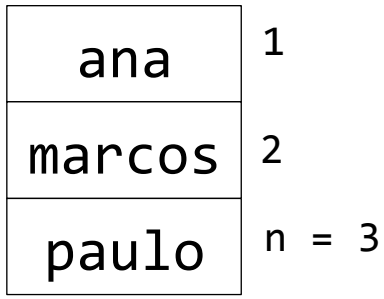
Lista Linear

Organização em Memória

- **Lista sequencial**
  - **organização física** ⇒ e_i e e_{i+1} armazenados de modo consecutivo em memória (endereços consecutivos)

L = (ana,marcos,paulo)

Memória Principal

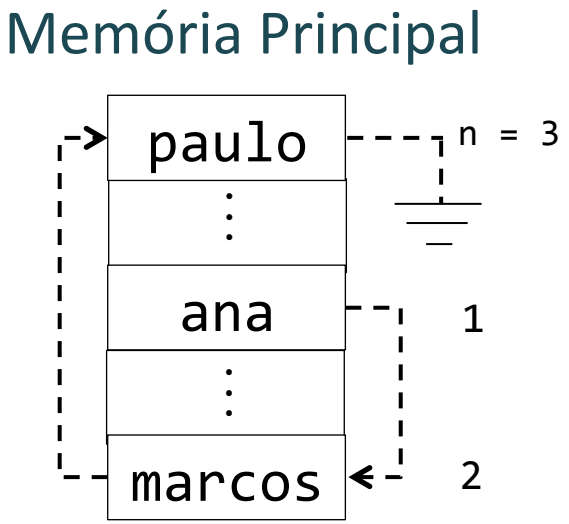


Lista Linear

Organização em Memória

- **Lista encadeada**
  - **organização lógica** - e_i e e_{i+1} armazenados de modo não consecutivo em memória

L = (ana,marcos,paulo)



Lista Linear

Alocação de Memória

- **Alocação Estática**: "reserva" de memória em tempo de compilação
- **Alocação Dinâmica**: em tempo de execução (na *heap*)

Lista Linear

Organização vs. Alocação de Memória

1) Sequencial e estática: array

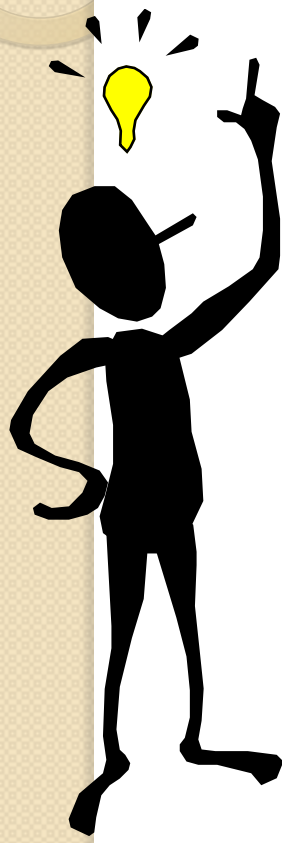
2) Encadeada e dinâmica: ponteiros

3) Encadeada e estática: array simulando encadeamento

4) Sequencial e dinâmica: alocação dinâmica de array

Alocação da Memória \ Organização da Memória	Sequencial	Encadeada
Estática	1	3
Dinâmica	4	2

O que usar?



Lista ordenada ou não ordenada?

Lista sequencial ou encadeada?

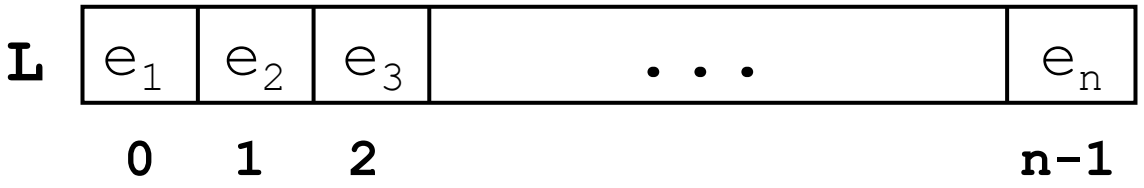
Alocação estática ou dinâmica?

- Definição e implementação da lista dependem de fatores como:
  - propósito da lista na aplicação
  - tamanho (estimado) da lista
  - operações mais frequentes
  - eficiência
  - restrições de armazenamento
  - …

Lista Sequencial

Lista Sequencial

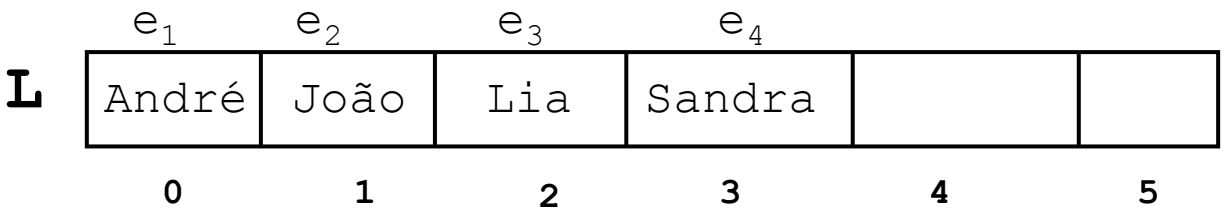
- Implementação como *array* - sequência de elementos contíguos na memória



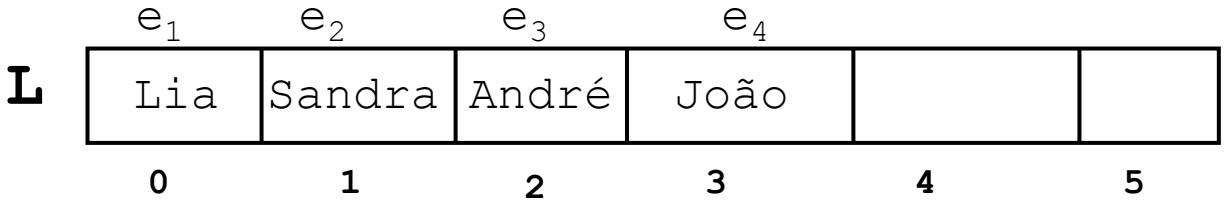
- Elemento e_i armazenado na posição de índice $i-1$ do *array* (em C)
  - acesso direto a e_i – índice $i-1$ do *array*
    - `primeiro = L[0]`
    - `ultimo = L[n-1]` (se lista cheia)
    - e_i `= L[i-1]`

Exemplo

- Lista ordenada

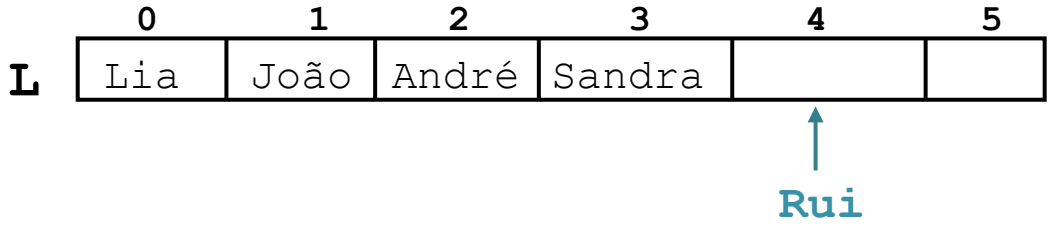


- Lista não ordenada



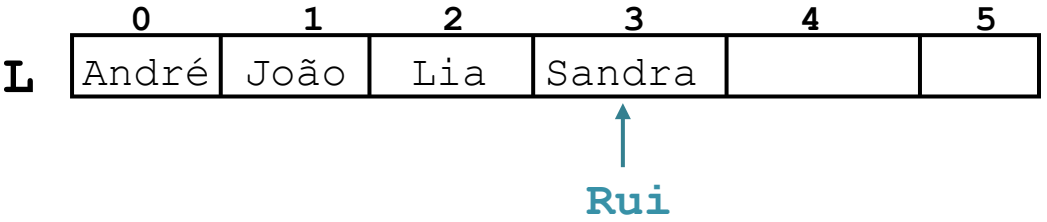
Inserção de item

- **Lista não ordenada** – inserção no final da lista



✓ Complexidade?

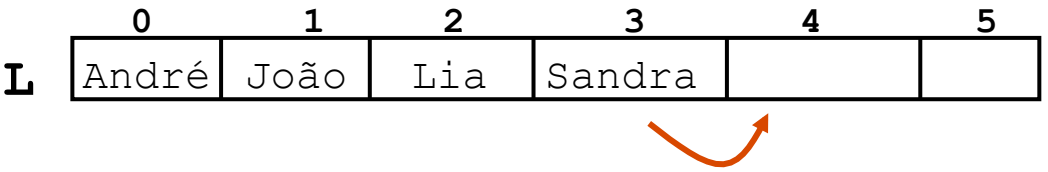
- **Lista ordenada** – inserção ordenada



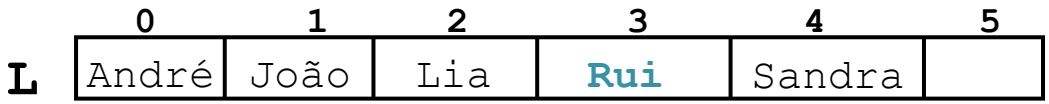
✓ Operação dominante?

Deslocamento de elementos…

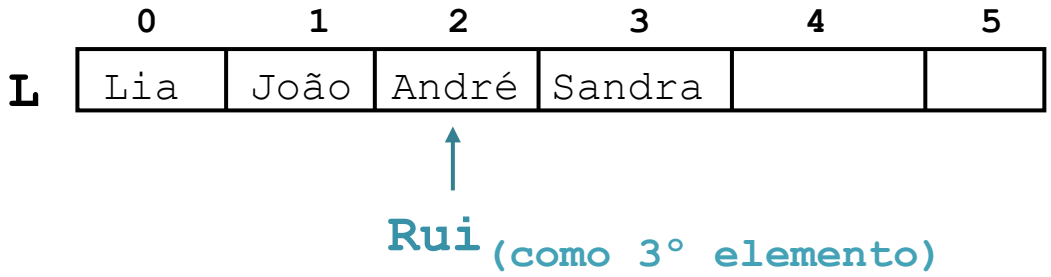
✓ Pior caso?



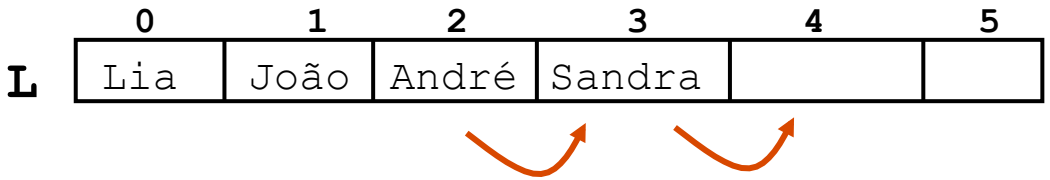
✓ Complexidade?

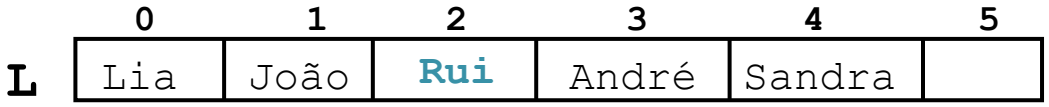


Inserção na i-ésima posição (índice i-1)



Deslocamento de elementos…





- ✓ Operação dominante?
- ✓ Pior caso?
- ✓ Complexidade?

Busca

- **Busca pelo `i`-ésimo item**
  - trivial => acesso direto (índice `i-1`)
  - complexidade?

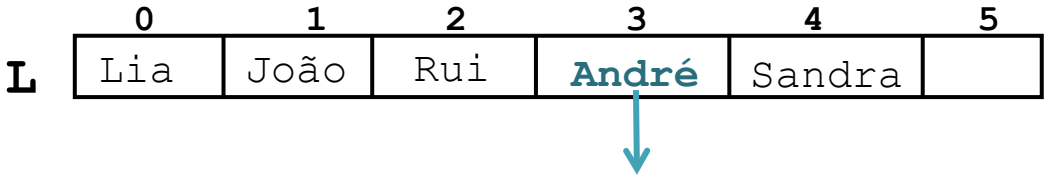
- **Busca por item**
  - **Lista não ordenada**
    - BUSCA SEQUENCIAL
      - pior caso?
      - complexidade?

Busca

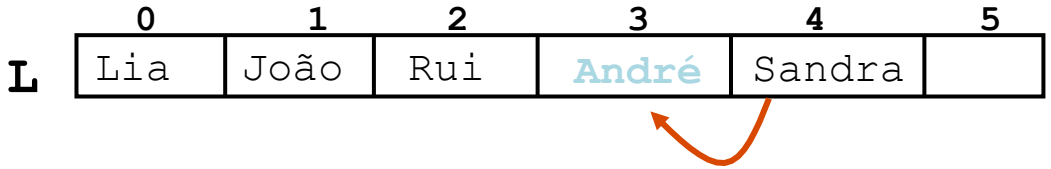
- Busca por item (continuação)
  - **Lista ordenada**
    - BUSCA SEQUENCIAL
      - mais eficiente que busca sequencial em lista não ordenada?
      - pior caso?
      - complexidade?
    - BUSCA BINÁRIA
      - somente para vetores ordenados
      - pior caso?
      - complexidade?

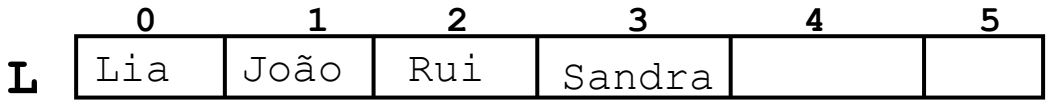
Remoção de item

- **Busca + Remoção**



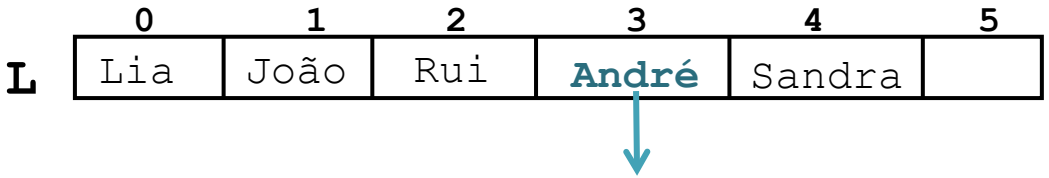
Deslocamento de elementos...



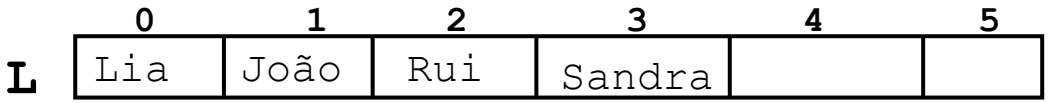


- ✓ Operação dominante?
- ✓ Pior caso?
- ✓ Complexidade?

Remoção do i-ésimo elemento (índice i-1)

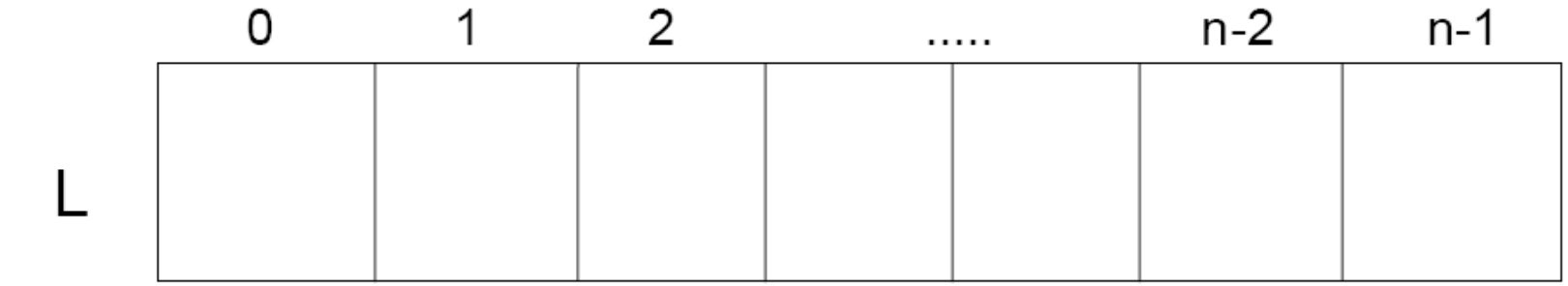


Deslocamento de elementos…

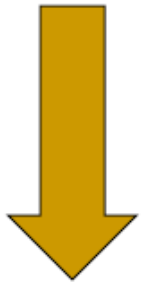


- ✓ Operação dominante?
- ✓ Pior caso?
- ✓ Complexidade?

TAD Lista – implementação sequencial



ITEM Lista[n];



`struct`

ITEM

Pode ser um registro com um número qualquer de campos

TAD Lista Não Ordenada – Exemplo de Definição da Interface

Arquivo Lista.h

```
1  #ifndef LISTA_H
2  #define LISTA_H
3
4  #define TAM_MAX 100 /*estimativa do tamanho máximo da lista*/
5  #define boolean int /*define tipo booleano – não existe em C*/
6  #define FALSE 0
7  #define TRUE 1
8  #define inicial 0
9  #define ERRO -32000
10
11 typedef int ITEM; /*Tipo ITEM (da lista) é um inteiro*/
12
13 typedef struct lista_ LISTA;
14
...
```

Arquivo Lista.h

```
...
15
16
17 LISTA *lista_criar(void);
18 boolean lista_apagar(LISTA *lista);
19 int lista_inserir(LISTA *lista, ITEM item);
20 boolean lista_inserir_pos(LISTA *lista, int pos, ITEM item);
21 boolean lista_remover(LISTA *lista, int chave);
22 boolean lista_remover_pos(LISTA *lista, int pos);
23 int lista_busca(int chave, LISTA *lista);
24 int lista_tamanho(LISTA *lista);
25 boolean lista_vazia(LISTA *lista);
26 boolean lista_cheia(LISTA *lista);
27 void lista_imprimir(LISTA *lista);
28
29 #endif
```

TAD Lista Não Ordenada – Exemplo de Implementação Sequencial

Arquivo Lista.c

```
1 #include "lista.h"
2
3 struct lista_ {
4    ITEM lista[TAM_MAX];
5    int inicio;  //inicio da lista
6    int fim;     //fim da lista - 1ª posicao livre para insercao
                  // e tamanho da lista
7 };
8
9 /*Cria logicamente uma lista, inicialmente vazia*/
16 LISTA *lista_criar(void){
17     LISTA *lista = (LISTA *) malloc(sizeof(LISTA));
18     if (lista != NULL){
19        lista->inicio = inicial;
20        lista->fim = lista->inicio; /*lista vazia*/
21     }
22   return (lista);
23 }
24
25
```

TAD Lista Não Ordenada – Exemplo de Implementação Sequencial

Arquivo Lista.c

```
1 /*Insere um elemento no fim da fila.*/
2 boolean lista_inserir(LISTA *lista, ITEM item){
3
4  if ((lista != NULL) && !lista_cheia(lista)){
5     (lista->lista[lista->fim]) = item;
6     lista->fim++;
7     return(TRUE);
8  }
9  return(FALSE);
10 }
11
...
```

Arquivo Lista.c

```
1 boolean lista_inserir_pos(LISTA *lista, int pos, ITEM item) {
2  int i;
3  // pos indica posição relativa do item na lista => índice-1
4  // verifica se existe espaço e se a posição está na lista
5  if (!lista_cheia(lista) && (pos-1 <= lista->fim)) {
6    for (i = (lista->fim-1); i >= pos-1; i--) { //move os itens
7       lista->lista[i + 1] = lista->lista[i];
8    }
9
10   lista->lista[pos-1] = item; //insere novo item
11   lista->fim++; //incrementa tamanho
12
13    return (TRUE);
14  } else  return (FALSE);
15
16
17 }
```

TAD Lista Não Ordenada – Exemplo de Implementação Sequencial

Arquivo Lista.c

```
1 int lista_busca(int chave, LISTA *lista) {
2   int i;
3
4   for (i = 0; i < lista->fim; i++)
5     if (lista->lista[i] == chave)
6         return (i+1); //retorna posicao relativa do item na lista
7   return (ERRO);
8 }
```

Lista Sequencial

- **Pontos Fortes**:
  - Acesso direto indexado a qualquer elemento da lista, dada a posição `i`
  - Tempo constante para acessar o `i-ésimo` elemento
- **Pontos Fracos**:
  - Movimentação para inserção e remoção do `i-ésimo` elemento
  - Tamanho máximo da lista pré-estimado
    - pode faltar espaço para armazenar elementos
    - pode sobrar espaço - desperdício de espaço alocado
- **Quando usar?**
  - Listas pequenas
  - Inserção/remoção no fim da lista
  - Tamanho máximo bem definido

Exercícios (não é preciso entregar)

- Implemente e teste as demais operações para o TAD Lista Não Ordenada (implementação sequencial)

- Implemente e teste o TAD Lista Ordenada (implementação sequencial)