Estruturas de Dados: **Listas**

Helena Graziottin Ribeiro
hgrib@ucs.br

UCS
UNIVERSIDADE
DE CAXIAS DO SUL

Listas

Definição: seqüência finita de itens de dados (elementos).

Listas

Definição: seqüência finita de itens de dados (elementos).

Na lista, os elementos podem estar ordenados, ou não.

Listas

Definição: seqüência finita de itens de dados (elementos).

Na lista, os elementos podem estar ordenados, ou não.

Na lista, os elementos podem ser todos do mesmo tipo, ou não.

Listas

- **Lista vazia:** não contém elementos.
- **comprimento (*length*):** número de elementos na lista.
- **Cabeça ou início (*head*):** início da lista
- **fim (*tail*):** final da lista
- **Listas ordenadas:** elementos posicionados em ordem (crescente/decrescente) de valor.
- **Listas não ordenadas:** não há relações entre valores e posições.

Listas

Exemplo de uma notação para representar listas:

L = (e0, e1, e2, e3, ... , en-1)

n = comprimento da lista

para todo elemento e da lista e i>0, ei precede ei+1 e segue ei-1

Lista vazia: ( )

Listas

Exemplo de uma notação para representar listas:

L = (e0, e1, e2, e3, ... , en-1)

n = comprimento da lista

para todo elemento e da lista e i>0, ei precede ei+1 e segue ei-1

Lista vazia: ( )

Listas

Operações básicas que uma implementação deve suportar (em geral):

- Listas podem crescer e diminuir: é possível inserir e retirar elementos
- Devemos ter acesso a qualquer elemento para efetuar operações: ler e modificar.
- Deve ser possível criar e destruir (reinicializar) listas.

Listas - TAD

TAD Lista{

Dados: itens

quantidade

Operações:

...

}

TAD item{

Dados: valor

Operações:

...

}

Listas - como implementar?

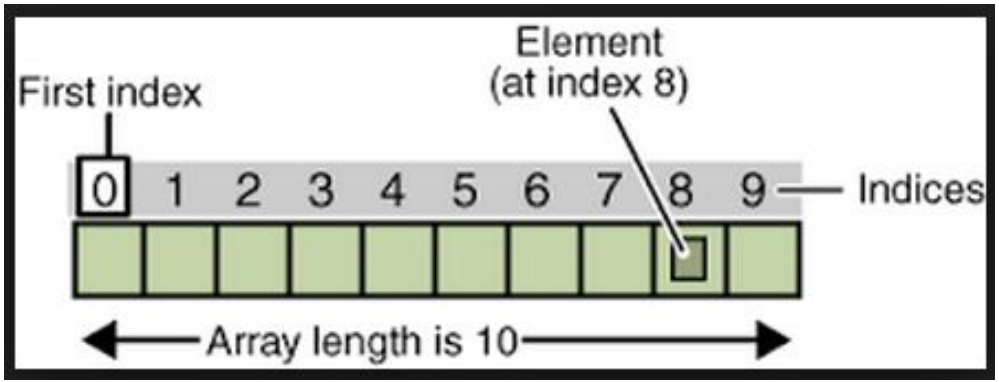
Duas abordagens básicas de implementação:

- através de **vetores** (arrays, ou sequencial)
- **listas encadeadas** (com referências ou apontadores)

Listas - como implementar? Vetores

Através de **vetores** (arrays, ou sequencial):

- **usa índices para acesso a posições sequenciais de memória**



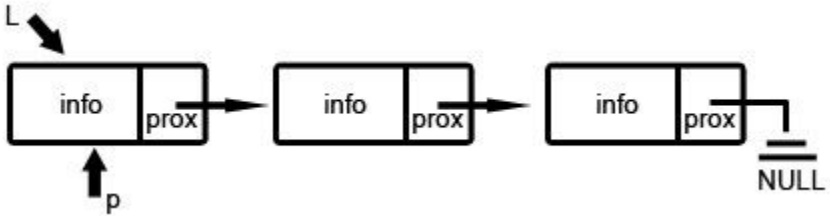
Listas - como implementar? Vetores

- ocupam um espaço contíguo de memória (**alocados estaticamente na memória**)
- permitem acesso randômico aos elementos (direto na posição)
- devem ser dimensionados com um número máximo de elementos

Listas - como implementar? Listas encadeadas

Através de **listas encadeadas** (com referências ou apontadores):

- cada elemento tem a referência (endereço de memória) do próximo elemento, porque eles não estão em sequência na memória



Listas - como implementar? Listas encadeadas

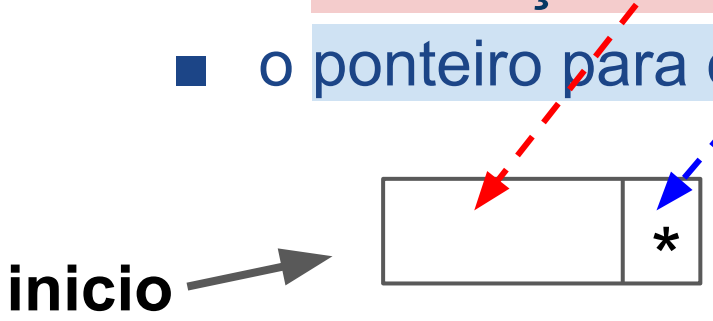
- estruturas que usam **alocação dinâmica de memória**
- crescem (ou decrescem) à medida que elementos são inseridos (ou removidos)
- acesso aos elementos de forma sequencial
  -

Listas - como implementar? Listas encadeadas

- Lista encadeada:
  - seqüência encadeada de elementos, chamados de nós (ou nodos) da lista
  - cada nó da lista é representado por dois campos:
    - a informação armazenada e
    - o ponteiro para o próximo elemento da lista
- a lista é representada por um **ponteiro para o primeiro nó**
- o ponteiro do último elemento é NULL

Listas - como implementar? Listas encadeadas

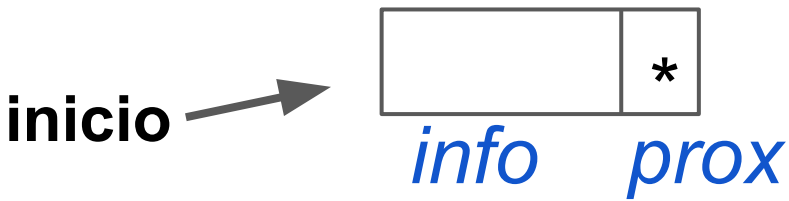
- cada nó da lista é representado por dois campos:
  - a informação armazenada e
  - o ponteiro para o próximo elemento da lista



- a lista é representada por um **ponteiro para o primeiro nó (inicio)**
- o ponteiro do último elemento é NULL (representado por *)

Listas encadeadas

```
struct elemento {
     int info;
     struct elemento *prox;
};
typedef struct elemento Elemento;
```



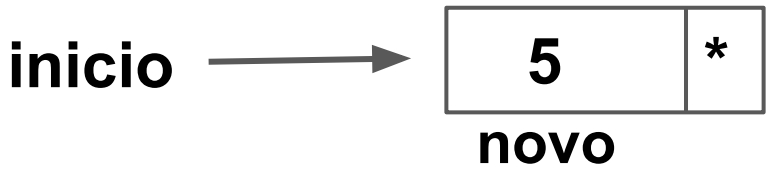
Listas encadeadas

Operações em listas:

- inserção:
  - do primeiro e único
  - do primeiro
  - do último
  - no “meio”

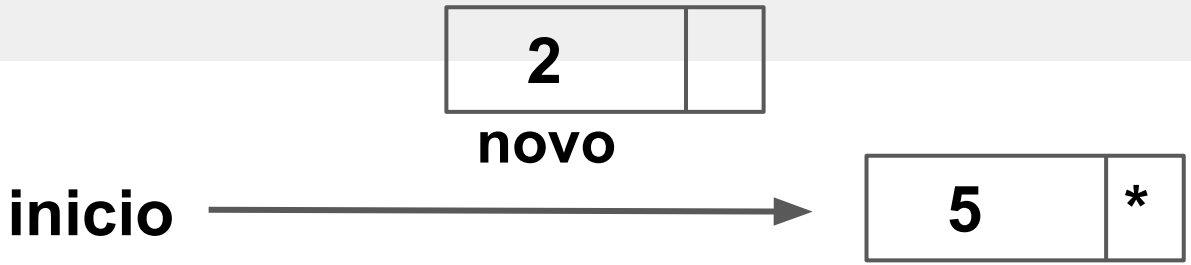
Listas encadeadas: inserção (1º e único)

```
Elemento *inicio, *novo;
inicio = NULL; /* inicialização da lista */
...
novo = (Elemento*) malloc(sizeof(Elemento));
novo->info = 5;
novo->prox = NULL;
inicio = novo;
```



Listas encadeadas: inserção (primeiro)

```
Elemento *novo;
novo = (Elemento*) malloc(sizeof(Elemento));
novo->info = 2;
novo->prox = inicio;
inicio = novo;
```

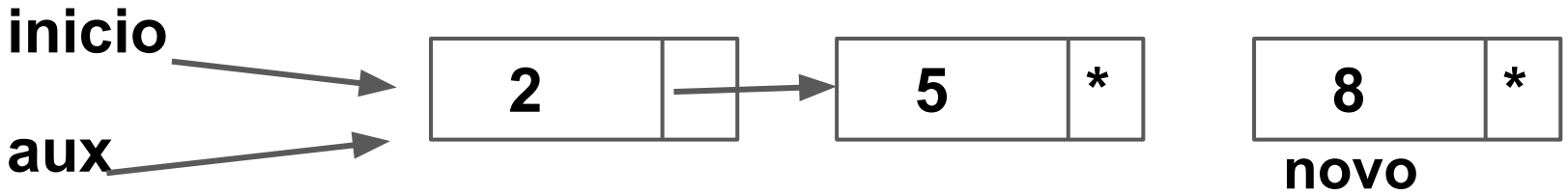


Listas encadeadas: inserção (primeiro)

```
Elemento *novo;
novo = (Elemento*) malloc(sizeof(Elemento));
novo->info = 2;
novo->prox = inicio;
inicio = novo;
```

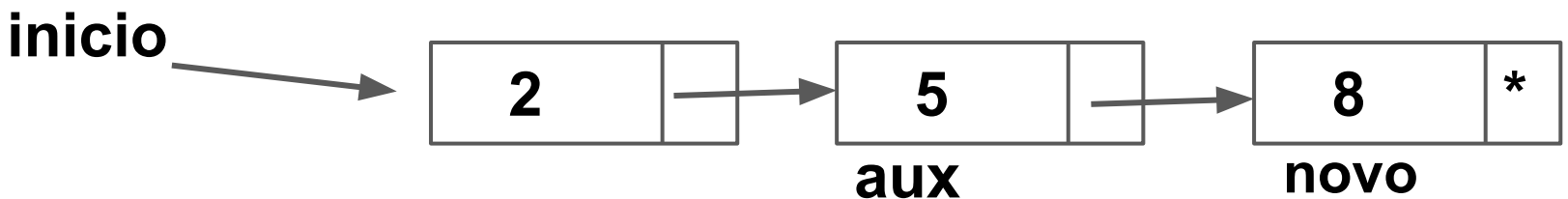
Listas encadeadas: inserção (último)

```
Elemento *novo, *aux=inicio;
novo = (Elemento*) malloc(sizeof(Elemento));
novo->info = 8;
novo->prox = NULL;
while (aux->prox != NULL)
      aux = aux->prox;
aux->prox = novo;
```



Listas encadeadas: inserção (último)

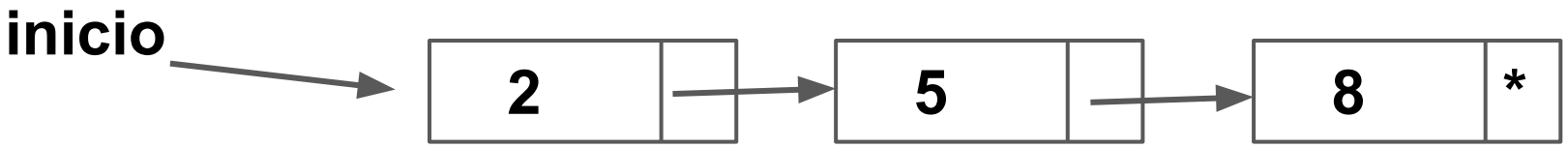
```
Elemento *novo, *aux=inicio;
novo = (Elemento*) malloc(sizeof(Elemento));
novo->info = 8;
novo->prox = NULL;
while (aux->prox != NULL)
      aux = aux->prox;
aux->prox = novo;
```



Listas encadeadas: inserção (no meio)

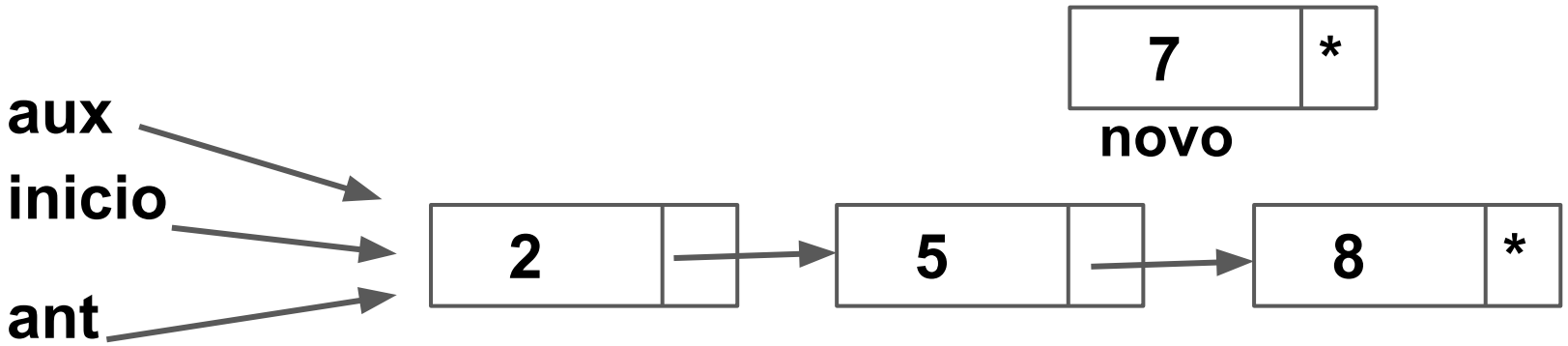
- Deve haver algum critério para caracterizar a inserção no meio, por exemplo:
  - inserção ordenada
  - inserção em posição determinada

Exemplo: inserir o 7, em ordem crescente de valores



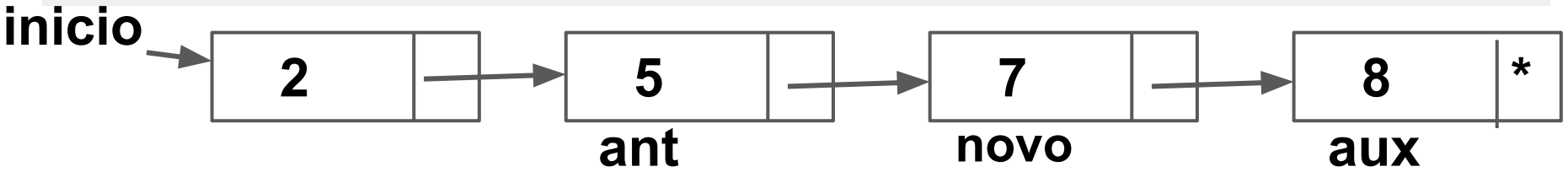
Listas encadeadas: inserção (no meio)

```
Elemento *novo, *aux=inicio, *ant=inicio;
novo = (Elemento*) malloc(sizeof(Elemento));
novo->info = 7;
novo->prox = NULL;
...
```



Listas encadeadas: inserção (no meio)

```
Elemento *novo, *aux=inicio, *ant=inicio;
novo = (Elemento*) malloc(sizeof(Elemento));
novo->info = 7;
novo->prox = NULL;
while (aux != NULL && aux->info < novo->info ){
      ant = aux;
      aux = aux->prox;}
ant->prox = novo;
novo->prox = aux;
```



Listas encadeadas - inserção

```
/* inserção no início: retorna a lista atualizada */
Elemento* lst_insere (Elemento* lst, int val)
{
 Elemento* novo = (Elemento*) malloc(sizeof(Elemento));
 novo->info = val;
 novo->prox = lst;
 return novo;
}
...
inicio = lst_insere (inicio, 9);
```