



Aula 08 - Listas Estáticas Sequenciais

Prof. Me. Claudiney R. Tinoco

profclaudineytinoco@gmail.com

Faculdade de Computação (FACOM)
Bacharelado em Ciência da Computação (BCC)
Bacharelado em Sistemas de Informação (BSI)

Algoritmos e Estruturas de Dados 1 (AED1)
GBC024 - GSI006

Listas Estáticas/Sequenciais

- Implementada através de um **vetor**
- Apresenta as seguintes características:
 - Elementos armazenados em **posições consecutivas**
 - **Inserção** de um elemento **na posição i** causa o **deslocamento à direita** desde o elemento a_i , até o último
 - **Remoção** do elemento **na posição i** causa o **deslocamento à esquerda** desde o elemento a_{i+1} , até o último



Listas Estáticas/Sequenciais

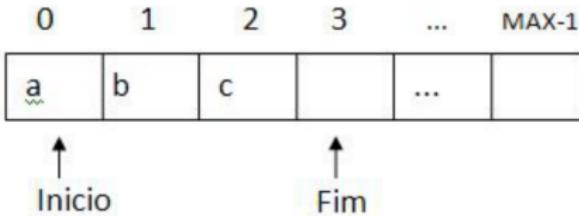
- **Vantagem:**
 - **Acesso direto** (indexado) aos elementos da lista
- **Desvantagens:**
 - **Movimentação de elementos** na inserção e remoção
 - Possibilidade de **super ou subestimação**
- **Recomendação de uso:**
 - Listas pequenas
 - Inserção e remoção no fim da lista
 - Tamanho máximo pré-definido
 - Aplicações fortemente baseadas em consultas



Lista Estática/Sequencial

- #### **• Estrutura de representação:**

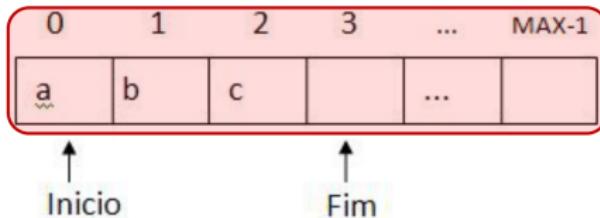
Ex: $L = \{a, b, c\}$



Listas Estáticas/Sequenciais

- Estrutura de representação:
 - Um **vetor de elementos** (ex: *int no[MAX]*)

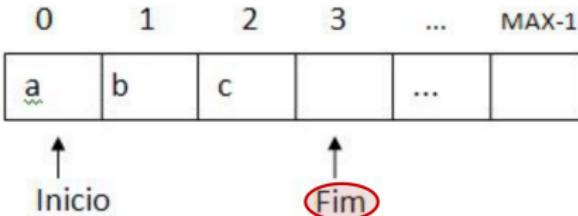
Ex: $L = \{a, b, c\}$



Listas Estáticas/Sequenciais

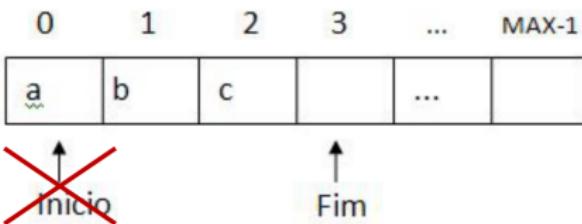
- Estrutura de representação:
 - Um **vetor de elementos** (ex: *int no[MAX]*)
 - Um campo p/ indicar o **final da lista** (ex: *int Fim*)
 - Pode indicar a **última posição ocupada** OU a **primeira posição livre**

Ex: $L = \{a, b, c\}$



Listas Estáticas/Sequenciais

- Estrutura de representação:
 - Um **vetor de elementos** (ex: *int no[MAX]*)
 - Um campo p/ indicar o **final da lista** (ex: *int Fim*)
 - Pode indicar a **última posição ocupada** OU a **primeira posição livre**





Estrutura de Representação em C

- Declaração de uma lista de inteiros no **lista.c**:

```
#define max 10 // Tamanho máximo
struct lista {
    int no[max];
    int Fim;
};
```

- Definição da interface da estrutura no **lista.h**:

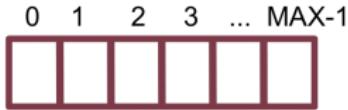
```
typedef struct lista * Lista;
```

Operação **cria_lista**

- **Criar uma lista**
 - Aloca uma instância da estrutura lista (*malloc*)

Operação **cria_lista**

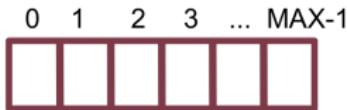
- **Criar uma lista**
 - Aloca uma instância da estrutura lista (*malloc*)



Fim = ?

Operação **cria_lista**

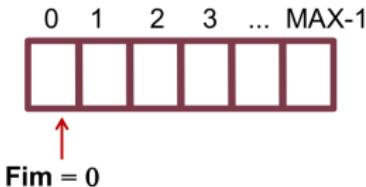
- **Criar uma lista**
 - Aloca uma instância da estrutura lista (*malloc*)
- **Colocar a lista no estado de vazia**
 - Campo **Fim** deve ser igual a 0
 - Estratégia que indica a próxima posição disponível



Fim = ?

Operação `cria_lista()`

- **Criar uma lista**
 - Aloca uma instância da estrutura lista (*malloc*)
- **Colocar a lista no estado de vazia**
 - Campo **Fim** deve ser igual a 0
 - Estratégia que indica a próxima posição disponível



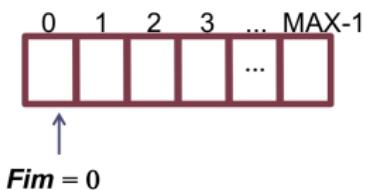
Operação `cria_lista()`

- Implementação em C:

```
Lista cria_lista() {  
    Lista lst;  
    lst = (Lista) malloc(sizeof(struct lista));  
  
    if (lst != NULL)  
        lst->Fim = 0; // Lista vazia  
  
    return lst;  
}
```

Operação lista_vazia()

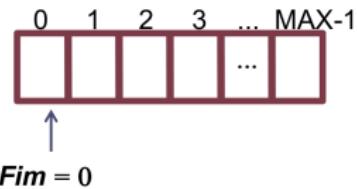
- Verifica se a lista está na condição de vazia:
 - Campo **Fim** deve ser igual a 0



Operação **lista_vazia()**

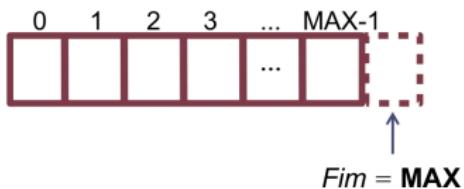
- Verifica se a lista está na condição de vazia:
 - Campo **Fim** deve ser igual a 0
- **Implementação em C:**

```
int lista_vazia(Lista lst) {  
    if (lst->Fim == 0)  
        return 1; // Lista vazia  
    else  
        return 0; // Lista NÃO vazia  
}
```



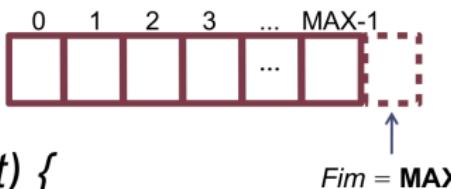
Operação lista_cheia()

- Verifica se a lista está na condição de cheia:
 - Campo **Fim** deve ser igual a **MAX**



Operação lista_cheia()

- Verifica se a lista está na condição de cheia:
 - Campo **Fim** deve ser igual a **MAX**
- **Implementação em C:**



```
int lista_cheia(Lista lst) {  
    if (lst->Fim == MAX)  
        return 1; // Lista cheia  
    else  
        return 0; // Lista NÃO cheia  
}
```

Operação de Inserção

- Afetada pelo **critério de ordenação**

- **Lista não ordenada:**

- Inserção na ordem de chegada
 - Insere no final da lista (**mais simples**)
 - Função: *insere_elem()*

- **Lista ordenada:**

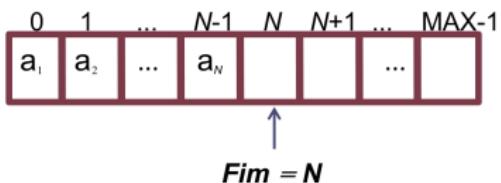
- Deve garantir que a lista permaneça ordenada
 - Inserção envolve **percorrimento da lista** para buscar a posição correta (**mais complexo**)
 - Função: *insere_ord()*

Operação de Inserção (Lista NÃO Ordenada)

- Usa a forma mais simples de inserção (**final da lista**)
 - Campo **Fim** já indica a posição a ser utilizada
 - Não envolve de percorrimento ou deslocamento
- A lista não pode estar cheia
 - **Fim** indica uma posição inválida (**Fim=MAX**)
- Campo **Fim** deve ser incrementado ao final da inserção

Operação de Inserção (Lista NÃO Ordenada)

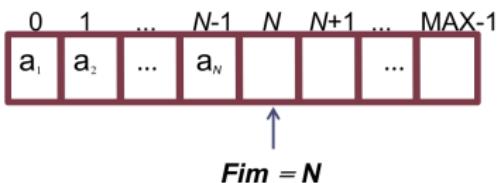
- Seja uma lista com N elementos:



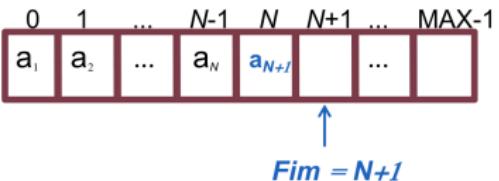


Operação de Inserção (Lista NÃO Ordenada)

- Seja uma lista com N elementos:



- Após a inserção do elemento a_{N+1} a lista fica:



Operação de Inserção (Lista NÃO Ordenada)

- Implementação em C:

```
int insere_elem(Lista lst, int elem) {  
    if (lst == NULL || lista_cheia(lst) == 1)  
        return 0;  
    
```

```
    lst->no[lst->Fim] = elem; // Insere elemento  
    lst->Fim++; // Avança o Fim  
    return 1;  
}
```

Operação de Inserção (Lista Ordenada)

- Inserção na posição correta
 - Envolve percorrimento
- Existem **5 casos** possíveis de inserção:
 - Lista está cheia
 - Lista está vazia
 - Novo elemento $<$ 1º nó da lista
 - Novo elemento \geq último nó da lista
 - Novo elemento entre o 1º e o último nó da lista

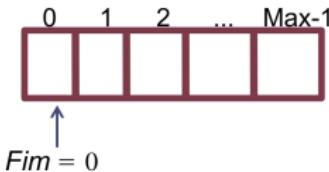
Operação de Inserção (Lista Ordenada)

- **Lista está cheia:**
 - Não é possível incluir novos elementos
 - Operação **falha**
 - Função retorna **ZERO**

Operação de Inserção (Lista Ordenada)

- **Lista está vazia:**

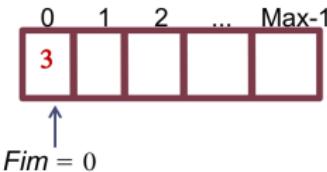
- Não precisa se preocupar com ordenação
- Insere elemento no 1º nó da lista (**posição 0**)



Operação de Inserção (Lista Ordenada)

- **Lista está vazia:**
 - Não precisa se preocupar com ordenação
 - Insere elemento no 1º nó da lista (**posição 0**)
 - Atribui o elemento ao vetor na posição indicada por **Fim**

Ex: inserir 3

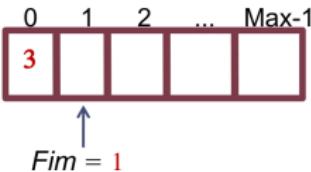


Operação de Inserção (Lista Ordenada)

- **Lista está vazia:**

- Não precisa se preocupar com ordenação
- Insere elemento no 1º nó da lista (**posição 0**)
 - Atribui o elemento ao vetor na posição indicada por **Fim**
 - Incrementa o campo **Fim**

Ex: inserir 3

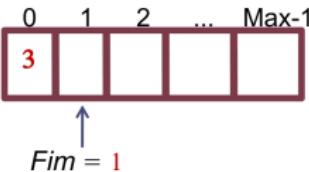


Operação de Inserção (Lista Ordenada)

- **Lista está vazia:**

- Não precisa se preocupar com ordenação
- Insere elemento no 1º nó da lista (**posição 0**)
 - Atribui o elemento ao vetor na posição indicada por **Fim**
 - Incrementa o campo **Fim**
 - Retorna 1 (**sucesso**)

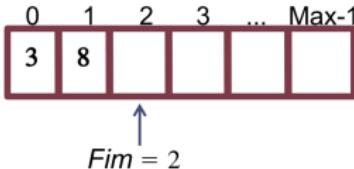
Ex: inserir 3



Operação de Inserção (Lista Ordenada)

- **Novo elemento < 1º nó da lista:**
 - Pode verificar **SEM percorrimento** da lista
 - Necessita **deslocar todos os elementos** da lista

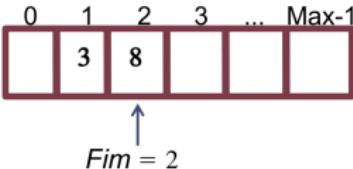
Ex: inserir 2



Operação de Inserção (Lista Ordenada)

- **Novo elemento < 1º nó da lista:**
 - Pode verificar **SEM percorrimento** da lista
 - Necessita **deslocar todos os elementos** da lista
 - Deslocar todos os nós **uma posição à direita**

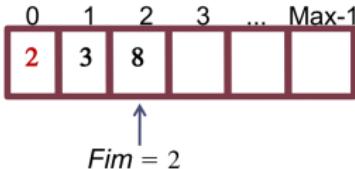
Ex: inserir 2



Operação de Inserção (Lista Ordenada)

- **Novo elemento < 1º nó da lista:**
 - Pode verificar **SEM percorrimento** da lista
 - Necessita **deslocar todos os elementos** da lista
 - Deslocar todos os nós **uma posição à direita**
 - Atribuir ao 1º nó da lista o valor do elemento

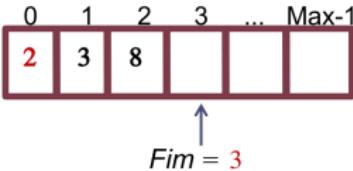
Ex: inserir 2



Operação de Inserção (Lista Ordenada)

- **Novo elemento < 1º nó da lista:**
 - Pode verificar **SEM percorrimento** da lista
 - Necessita **deslocar todos os elementos** da lista
 - Deslocar todos os nós **uma posição à direita**
 - Atribuir ao 1º nó da lista o valor do elemento
 - Incrementa **Fim**

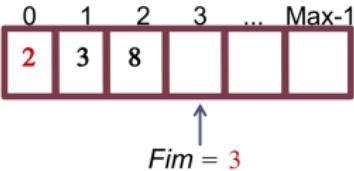
Ex: inserir 2



Operação de Inserção (Lista Ordenada)

- **Novo elemento < 1º nó da lista:**
 - Pode verificar **SEM percorrimento** da lista
 - Necessita **deslocar todos os elementos** da lista
 - Deslocar todos os nós **uma posição à direita**
 - Atribuir ao 1º nó da lista o valor do elemento
 - Incrementa **Fim**
 - Retorna 1 (**sucesso**)

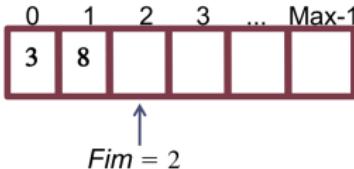
Ex: inserir 2



Operação de Inserção (Lista Ordenada)

- **Novo elemento \geq último nó da lista:**
 - Pode verificar **SEM percorrimento** da lista
 - Não envolve deslocamento (**insere no final**)

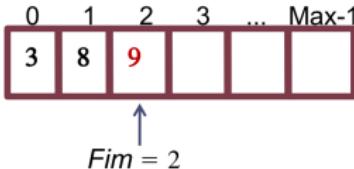
Ex: inserir 9



Operação de Inserção (Lista Ordenada)

- **Novo elemento \geq último nó da lista:**
 - Pode verificar **SEM percorrimento** da lista
 - Não envolve deslocamento (**insere no final**)
 - Atribui o elemento ao vetor na posição indicada por **Fim**

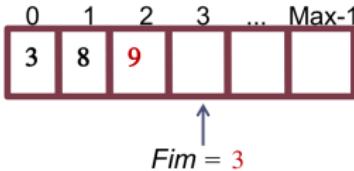
Ex: inserir 9



Operação de Inserção (Lista Ordenada)

- **Novo elemento \geq último nó da lista:**
 - Pode verificar **SEM percorrimento** da lista
 - Não envolve deslocamento (**insere no final**)
 - Atribui o elemento ao vetor na posição indicada por **Fim**
 - Incrementa o campo **Fim**

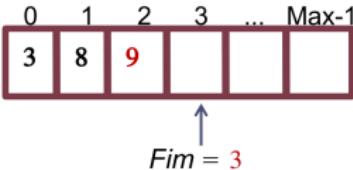
Ex: inserir 9



Operação de Inserção (Lista Ordenada)

- **Novo elemento \geq último nó da lista:**
 - Pode verificar **SEM percorrimento** da lista
 - Não envolve deslocamento (**insere no final**)
 - Atribui o elemento ao vetor na posição indicada por **Fim**
 - Incrementa o campo **Fim**
 - Retorna 1 (**sucesso**)

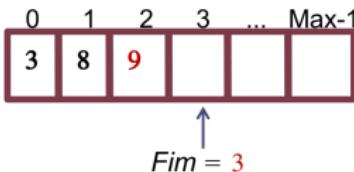
Ex: inserir 9



Operação de Inserção (Lista Ordenada)

- **Novo elemento \geq último nó da lista:**
 - Pode verificar **SEM percorrimento** da lista
 - Não envolve deslocamento (**insere no final**)
 - Atribui o elemento ao vetor na posição indicada por **Fim**
 - Incrementa o campo **Fim**
 - Retorna 1 (**sucesso**)

Ex: inserir 9

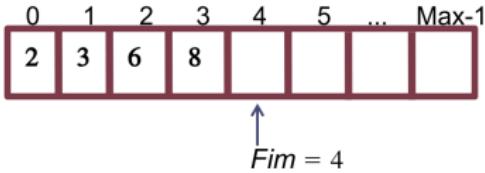


*Similar ao
lista vazia*

Operação de Inserção (Lista Ordenada)

- **Novo elemento entre 1º e último nó da lista:**
 - Necessita de **percorrimento** da lista
 - Deslocamento dos nós da lista que são maiores

Ex: inserir 5

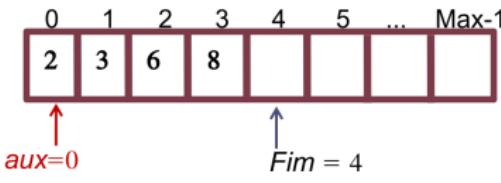


Operação de Inserção (Lista Ordenada)

- Novo elemento entre 1º e último nó da lista:
 - Necessita de **percorrimento** da lista
 - Deslocamento dos nós da lista que são maiores
 - Busca o 1º nó da lista maior que o novo elemento

Ex: inserir 5

**2 > 5 ? Não
(avançar)**

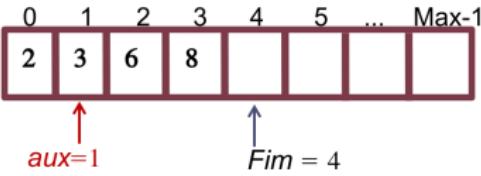


Operação de Inserção (Lista Ordenada)

- Novo elemento entre 1º e último nó da lista:
 - Necessita de **percorrimento** da lista
 - Deslocamento dos nós da lista que são maiores
 - Busca o 1º nó da lista maior que o novo elemento

Ex: inserir 5

**$3 > 5$? Não
(avançar)**

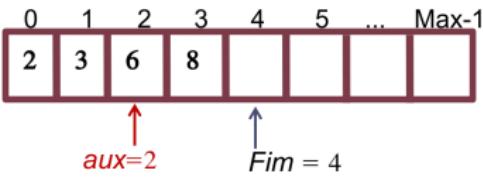


Operação de Inserção (Lista Ordenada)

- Novo elemento entre 1º e último nó da lista:
 - Necessita de **percorrimento** da lista
 - Deslocamento dos nós da lista que são maiores
 - Busca o 1º nó da lista maior que o novo elemento

Ex: inserir 5

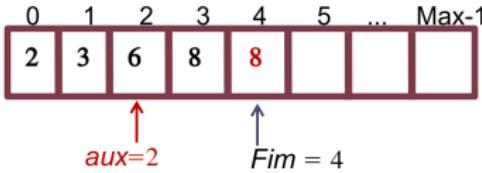
$6 > 5 ? Sim$
(parar)



Operação de Inserção (Lista Ordenada)

- **Novo elemento entre 1º e último nó da lista:**
 - Necessita de **percorrimento** da lista
 - Deslocamento dos nós da lista que são maiores
 - Busca o 1º nó da lista maior que o novo elemento
 - Desloca nós **1 posição à direita** do fim até a posição atual

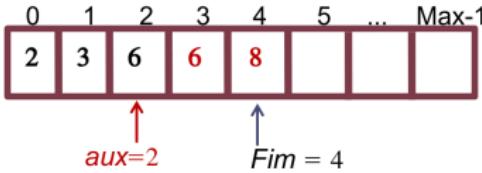
Ex: inserir 5



Operação de Inserção (Lista Ordenada)

- **Novo elemento entre 1º e último nó da lista:**
 - Necessita de **percorrimento** da lista
 - Deslocamento dos nós da lista que são maiores
 - Busca o 1º nó da lista maior que o novo elemento
 - Desloca nós **1 posição à direita** do fim até a posição atual

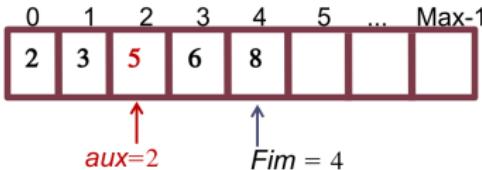
Ex: inserir 5



Operação de Inserção (Lista Ordenada)

- **Novo elemento entre 1º e último nó da lista:**
 - Necessita de **percorrimento** da lista
 - Deslocamento dos nós da lista que são maiores
 - Busca o 1º nó da lista maior que o novo elemento
 - Desloca nós **1 posição à direita** do fim até a posição atual
 - Atribui o elemento ao vetor na posição encontrada (**aux**)

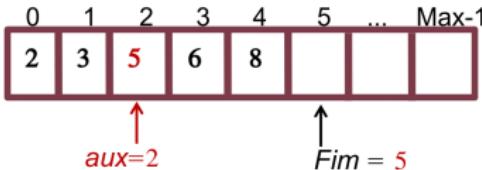
Ex: inserir 5



Operação de Inserção (Lista Ordenada)

- **Novo elemento entre 1º e último nó da lista:**
 - Necessita de **percorrimento** da lista
 - Deslocamento dos nós da lista que são maiores
 - Busca o 1º nó da lista maior que o novo elemento
 - Desloca nós **1 posição à direita** do fim até a posição atual
 - Atribui o elemento ao vetor na posição encontrada (**aux**)
 - Incrementa o campo **Fim**

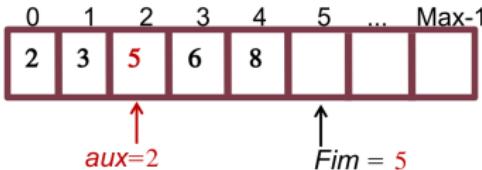
Ex: inserir 5



Operação de Inserção (Lista Ordenada)

- **Novo elemento entre 1º e último nó da lista:**
 - Necessita de **percorrimento** da lista
 - Deslocamento dos nós da lista que são maiores
 - Busca o 1º nó da lista maior que o novo elemento
 - Desloca nós **1 posição à direita** do fim até a posição atual
 - Atribui o elemento ao vetor na posição encontrada (**aux**)
 - Incrementa o campo **Fim**
 - Retorna 1 (**sucesso**)

Ex: inserir 5



Operação de Inserção (Lista Ordenada)

- **Implementação em C:**

```
int insere_ord(Lista lst, int elem) {  
    if (lst == NULL || lista_cheia(lst) == 1)  
        return 0; // Falha  
  
    // Trata lista vazia ou elemento ≥ último da lista  
    if (lista_vazia(lst) == 1 || elem >= lst->no[lst->Fim-1]) {  
        lst->no[lst->Fim] = elem; // Insere no final  
    }  
    ...
```



Operação de Inserção (Lista Ordenada)

...

```
else {  
    int i, aux = 0;  
    while (elem >= lst->no[aux]) // Percorimento  
        aux++;  
    for (i = lst->Fim; i > aux; i--) // Deslocamento  
        lst->no[i] = lst->no[i-1];  
    lst->no[aux] = elem; // Inclui o elemento na lista  
}  
lst->Fim++; // Avança o Fim  
return 1; // Sucesso  
}
```



Operação de Remoção

- Necessita de percorrimento da lista
 - Busca pelo elemento a ser removido
- Remoção no meio envolve movimentação dos nós (**deslocamento à esquerda**)
- **Critério de ordenação** afeta quando não existe o elemento na lista
 - **Lista não ordenada:** tem que percorrer **até o final da lista**
 - **Lista ordenada:** percorre **até achar nó maior**



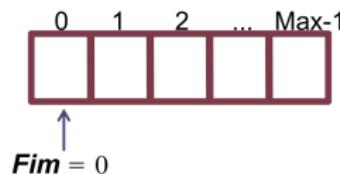
Operação de Remoção (Lista NÃO Ordenada)

- Existem **4 casos** possíveis de remoção:
 - Lista está vazia
 - Elemento não está na lista
 - Elemento é o último nó da lista
 - Elemento está entre o 1º e o penúltimo nó da lista

Operação de Remoção (Lista NÃO Ordenada)

- **Lista vazia:**

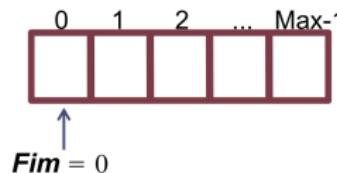
Ex: remover 5



Operação de Remoção (Lista NÃO Ordenada)

- **Listas vazias:**
 - Não existe o elemento

Ex: remover 5



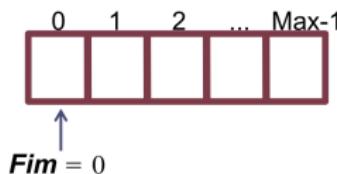
5

Operação de Remoção (Lista NÃO Ordenada)

- **Listagem vazia:**
 - Não existe o elemento
 - Retorna 0 (**operação falha**)

Ex: remover 5

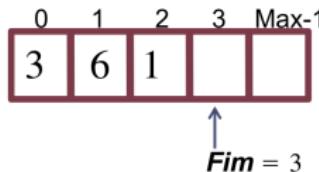
Retorna 0



Operação de Remoção (Lista NÃO Ordenada)

- Elemento não está na lista:

Ex: remover 5

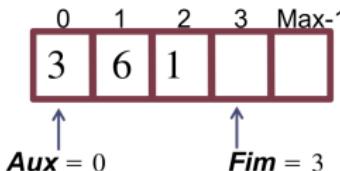


Operação de Remoção (Lista NÃO Ordenada)

- **Elemento não está na lista:**
 - Lista é percorrida até seu final (***Aux = Fim***)
 - Elemento é comparado com cada nó da lista para verificar sua existência

Ex: remover 5

**$3 \neq 5 ?$ Sim
(avançar)**

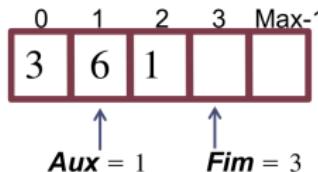


Operação de Remoção (Lista NÃO Ordenada)

- **Elemento não está na lista:**
 - Lista é percorrida até seu final ($\text{Aux} = \text{Fim}$)

Ex: remover 5

$6 \neq 5 ?$ Sim
(avançar)

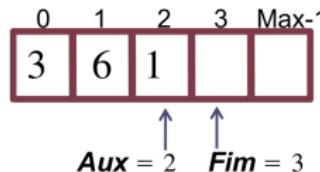


Operação de Remoção (Lista NÃO Ordenada)

- **Elemento não está na lista:**
 - Lista é percorrida até seu final ($\text{Aux} = \text{Fim}$)

Ex: remover 5

$1 \neq 5 ?$ Sim
(avançar)

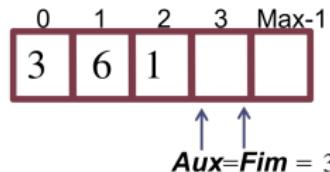


Operação de Remoção (Lista NÃO Ordenada)

- **Elemento não está na lista:**
 - Lista é percorrida até seu final ($\text{Aux} = \text{Fim}$)
 - Não existe o elemento

Ex: remover 5

Aux = Fim? Sim
(Fim da Lista)



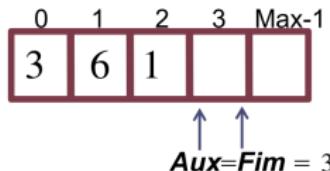
3 5

Operação de Remoção (Lista NÃO Ordenada)

- **Elemento não está na lista:**
 - Lista é percorrida até seu final ($\text{Aux} = \text{Fim}$)
 - Não existe o elemento
 - Retorna 0 (**operação falha**)

Ex: remover 5

Aux = Fim? Sim
(Fim da Lista)

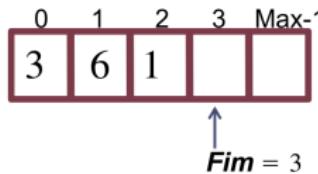


Retorna 0

Operação de Remoção (Lista NÃO Ordenada)

- Elemento é o último da lista:

Ex: remover 1

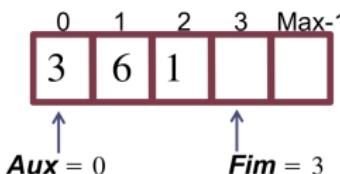


Operação de Remoção (Lista NÃO Ordenada)

- **Elemento é o último da lista:**
 - Lista é percorrida até encontrar o elemento ($Ist->no[Aux] = elem$)

Ex: remover 1

$3 \neq 1 ? Sim$
(avançar)

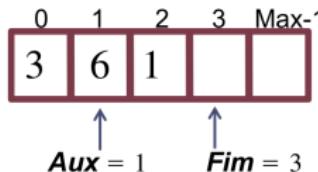


Operação de Remoção (Lista NÃO Ordenada)

- **Elemento é o último da lista:**
 - Lista é percorrida até encontrar o elemento ($Ist->no[Aux] = elem$)

Ex: remover 1

$6 \neq 1 ? Sim$
(avançar)

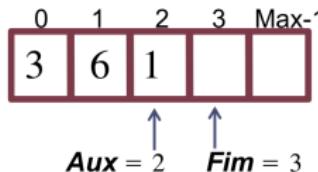


Operação de Remoção (Lista NÃO Ordenada)

- Elemento é o último da lista:
 - Lista é percorrida até encontrar o elemento ($Ist->no[Aux] = elem$)

Ex: remover 1

$1 \neq 1$? Não
(parar)

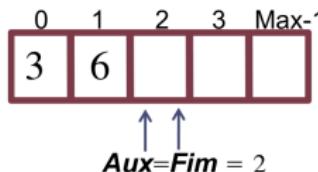


3 1

Operação de Remoção (Lista NÃO Ordenada)

- **Elemento é o último da lista:**
 - Lista é percorrida até encontrar o elemento ($Ist->no[Aux] = elem$)
 - Decrementar **Fim** (remoção do elemento)

Ex: remover 1

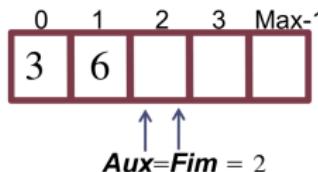


Operação de Remoção (Lista NÃO Ordenada)

- **Elemento é o último da lista:**
 - Lista é percorrida até encontrar o elemento ($Ist->no[Aux] = elem$)
 - Decrementar **Fim** (remoção do elemento)
 - Retornar 1 (operação bem sucedida)

Ex: remover 1

Retorna 1



Operação de Remoção (Lista NÃO Ordenada)

- Elemento entre o 1º e o penúltimo nó da lista:

Ex: remover 3

0	1	2	3	4	Max-1
1	3	9	2		

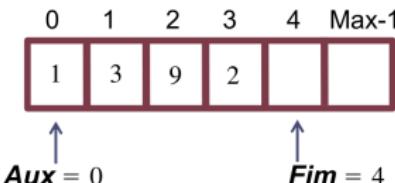
\uparrow
 $Fim = 4$

Operação de Remoção (Lista NÃO Ordenada)

- **Elemento entre o 1º e o penúltimo nó da lista:**
 - Lista é percorrida até encontrar o elemento
(Ist->no[Aux] = elem)

Ex: remover 3

**$1 \neq 3 ?$ Sim
(avançar)**



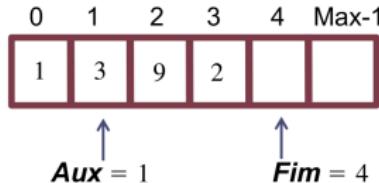
Operação de Remoção (Lista NÃO Ordenada)

- **Elemento entre o 1º e o penúltimo nó da lista:**
 - Lista é percorrida até encontrar o elemento
(Ist->no[Aux] = elem)

Ex: remover 3

**3 ≠ 3 ? Não
(parar)**

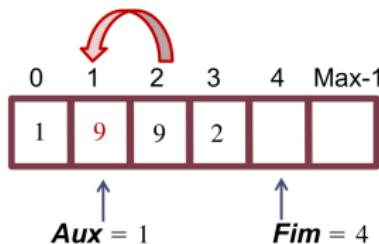
E 3



Operação de Remoção (Lista NÃO Ordenada)

- **Elemento entre o 1º e o penúltimo nó da lista:**
 - Lista é percorrida até encontrar o elemento
(Ist->no[Aux] = elem)
 - **Deslocar para a esquerda** do sucessor de **Aux** até o final da lista (de **Aux+1** até **Fim-1**)

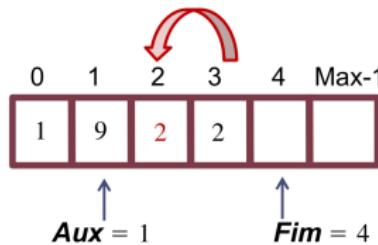
Ex: remover 3



Operação de Remoção (Lista NÃO Ordenada)

- **Elemento entre o 1º e o penúltimo nó da lista:**
 - Lista é percorrida até encontrar o elemento ($Ist->no[Aux] = elem$)
 - **Deslocar para a esquerda** do sucessor de **aux** até o final da lista (de $Aux+1$ até $Fim-1$)

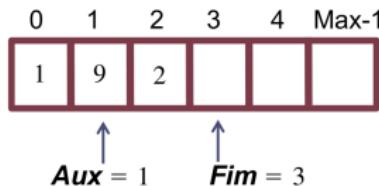
Ex: remover 3



Operação de Remoção (Lista NÃO Ordenada)

- **Elemento entre o 1º e o penúltimo nó da lista:**
 - Lista é percorrida até encontrar o elemento
(Ist->no[Aux] = elem)
 - **Deslocar para a esquerda** do sucessor de **aux** até o final da lista (de **Aux+1** até **Fim-1**)
 - Decrementar **Fim**

Ex: remover 3

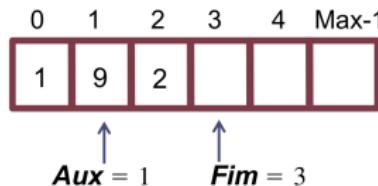


Operação de Remoção (Lista NÃO Ordenada)

- **Elemento entre o 1º e o penúltimo nó da lista:**
 - Lista é percorrida até encontrar o elemento ($Ist->no[Aux] = elem$)
 - **Deslocar para a esquerda** do sucessor de **aux** até o final da lista (de **Aux+1** até **Fim-1**)
 - Decrementar **Fim**
 - Retornar 1 (**operação bem sucedida**)

Ex: remover 3

Retorna 1



Operação de Remoção (Lista NÃO Ordenada)

- Implementação em C:

```
int remove_elem (Lista lst, int elem) {  
    if (lst == NULL || lista_vazia(lst) == 1)  
        return 0; // Falha  
    int i, Aux = 0;  
  
    // Percorimento até achar o elem ou final de lista  
    while (Aux < lst->Fim && lst->no[Aux] != elem)  
        Aux++;  
  
    ...
```

Operação de Remoção (Lista NÃO Ordenada)

- Implementação em C:

```
int remove_elem (Lista lst, int elem) {
    if (lst == NULL || lista_vazia(lst) == 1)
        return 0; // Falha
    int i, Aux = 0;

    // Percorimento até achar o elem ou final de lista
    while (Aux < lst->Fim && lst->no[Aux] != elem)
        Aux++;
    ...
    Teste invertido
```

Operação de Remoção (Lista NÃO Ordenada)

- Implementação em C:

```
int remove_elem (Lista lst, int elem) {  
    if (lst == NULL || lista_vazia(lst) == 1)  
        return 0; // Falha  
    int i, Aux = 0;  
  
    // Percorimento até achar o elem ou final de lista  
    while (Aux < lst->Fim && lst->no[Aux] != elem)  
        Aux++;  
    ...  
    Teste invertido
```

Operação de Remoção (Lista NÃO Ordenada)

- Implementação em C:

```
int remove_elem (Lista lst, int elem) {  
    if (lst == NULL || lista_vazia(lst) == 1)  
        return 0; // Falha  
    int i, Aux = 0;  
  
    // Percorimento até achar o elem ou final de lista  
    while (Aux < lst->Fim && lst->no[Aux] != elem)  
        Aux++;  
    ...  
    Teste invertido
```

Operação de Remoção (Lista NÃO Ordenada)

- Implementação em C:

```
...
if (Aux == lst->Fim) // Final de lista (não elem)
    return 0; // Falha

// Deslocamento à esq. do sucessor até o final da lista
for (i = Aux+1; i < lst->Fim; i++)
    lst->no[i-1] = lst->no[i];

lst->Fim--; // Decremento do campo Fim
return 1; // Sucesso
}
```

Operação de Remoção (Lista NÃO Ordenada)

- Implementação em C:

```
...
if (Aux == lst->Fim) // Final de lista (não elem)
    return 0; // Falha

// Deslocamento à esq. do sucessor até o final da lista
for (i = Aux+1; i < lst->Fim; i++)
    lst->no[i-1] = lst->no[i];

lst->Fim--; // Decremento do campo Fim
return 1; // Sucesso
}
```

Operação de Remoção (Lista NÃO Ordenada)

- Implementação em C:

...

```
if (Aux == lst->Fim) // Final de lista (não elem)
```

```
return 0; // Falha
```

```
// Deslocamento à esq. do sucessor até o final da lista
```

```
for (i = Aux+1; i < lst->Fim; i++)
```

```
lst->no[i-1] = lst->no[i];
```

```
lst->Fim--; // Decremento do campo Fim
```

```
return 1; // Sucesso
```

```
}
```

Operação de Remoção (Lista NÃO Ordenada)

- Implementação em C:

```
...
if (Aux == lst->Fim) // Final de lista (não elem)
    return 0; // Falha

// Deslocamento à esq. do sucessor até o final da lista
for (i = Aux+1; i < lst->Fim; i++)
    lst->no[i-1] = lst->no[i];
        Atual      Sucessor
lst->Fim--; // Decremento do campo Fim
return 1; // Sucesso
}
```

Operação de Remoção (Lista Ordenada)

- Existem **6 casos** possíveis de remoção:
 - Lista está vazia
 - Elemento é o último nó da lista
 - Elemento está entre o 1º e o penúltimo nó da lista
 - Elemento não está na lista

Operação de Remoção (Lista Ordenada)

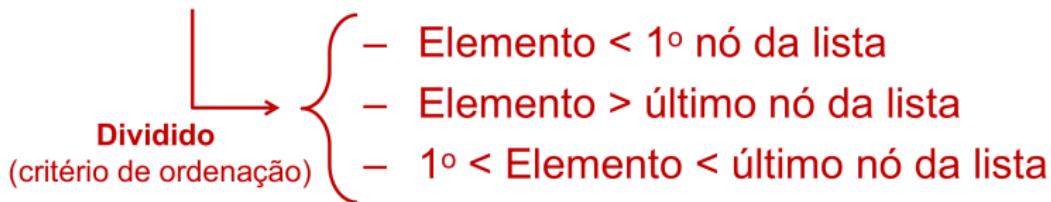
- Existem **6 casos** possíveis de remoção:

- Lista está vazia
- Elemento é o último nó da lista
- Elemento está entre o 1º e o penúltimo nó da lista
- Elemento não está na lista

} similar ao TAD
Lista NÃO Ordenada

Operação de Remoção (Lista Ordenada)

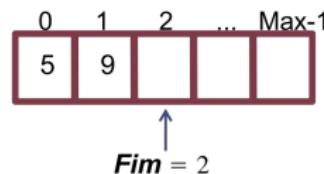
- Existem **6 casos** possíveis de remoção:
 - Lista está vazia
 - Elemento é o último nó da lista
 - Elemento está entre o 1º e o penúltimo nó da lista
 - ~~Elemento não está na lista~~



Operação de Remoção (Lista Ordenada)

- Elemento < 1º nó da lista:

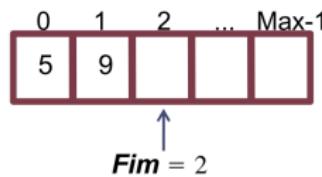
Ex: remover 2



Operação de Remoção (Lista Ordenada)

- **Elemento < 1º nó da lista:**
 - Não existe o elemento

Ex: remover 2

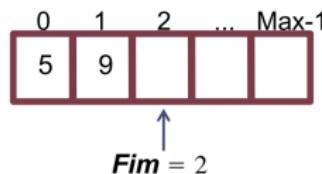


2

Operação de Remoção (Lista Ordenada)

- **Elemento < 1º nó da lista:**
 - Não existe o elemento
 - Retorna 0 (**operação falha**)

Ex: remover 2

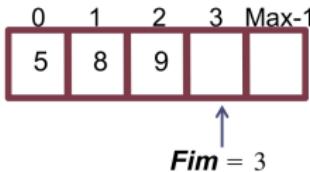


Retorna 0

Operação de Remoção (Lista Ordenada)

- Elemento > último nó da lista:

Ex: remover 12

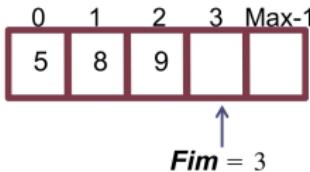


Operação de Remoção (Lista Ordenada)

- **Elemento > último nó da lista:**
 - Compara elemento com o **último da lista**
(*elem > lst->no[lst->fim-1]*)

Ex: remover 12

9 < 12 ?

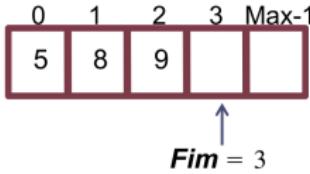


Operação de Remoção (Lista Ordenada)

- **Elemento > último nó da lista:**
 - Compara elemento com o **último da lista** (***elem > lst->no[lst->fim-1]***)
 - Não existe o elemento

Ex: remover 12

9 < 12 ? Sim
(12)

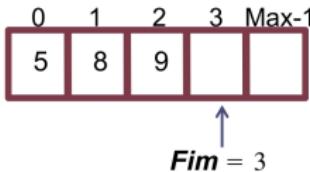


Operação de Remoção (Lista Ordenada)

- **Elemento > último nó da lista:**
 - Compara elemento com o **último da lista** (***elem > Ist->no[Ist->fim-1]***)
 - Não existe o elemento
 - Retorna 0 (**operação falha**)

Ex: remover 12

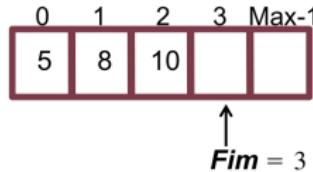
retorna 0



Operação de Remoção (Lista Ordenada)

- **1º < Elemento < último nó da lista:**

Ex: remover 9

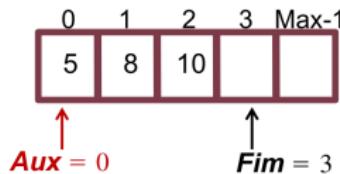


Operação de Remoção (Lista Ordenada)

- **1º < Elemento < último nó da lista:**
 - Lista é percorrida até encontrar um nó na lista maior que o elemento desejado ($Ist->no[Aux] > elem$)

Ex: remover 9

**$9 \leq 5 ?$ Não
(avançar)**

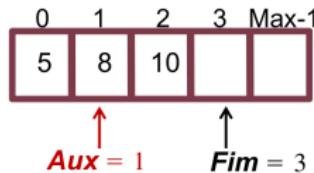


Operação de Remoção (Lista Ordenada)

- **1º < Elemento < último nó da lista:**
 - Lista é percorrida até encontrar um nó na lista maior que o elemento desejado (*Ist->no[Aux] > elem*)

Ex: remover 9

**$9 \leq 8 ?$ Não
(avançar)**

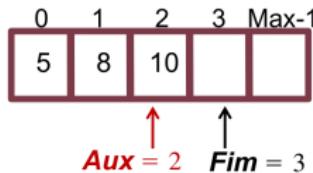


Operação de Remoção (Lista Ordenada)

- **1º < Elemento < último nó da lista:**
 - Lista é percorrida até encontrar um nó na lista maior que o elemento desejado (*Ist->no[Aux] > elem*)

Ex: remover 9

**$9 \leq 10 ?$ Sim
(parar)**



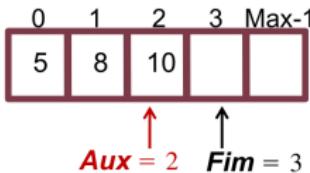
Operação de Remoção (Lista Ordenada)

- **1º < Elemento < último nó da lista:**
 - Lista é percorrida até encontrar um nó na lista maior que o elemento desejado ($Ist->no[Aux] > elem$)
 - Não existe o elemento

Ex: remover 9

$9 \leq 10 ?$ Sim
(parar)

9

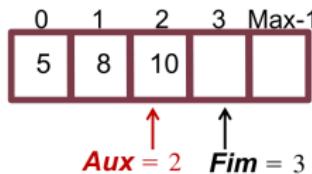


Operação de Remoção (Lista Ordenada)

- **1º < Elemento < último nó da lista:**
 - Lista é percorrida até encontrar um nó na lista maior que o elemento desejado ($Ist->no[Aux] > elem$)
 - Não existe o elemento
 - Retorna 0 (operação falha)

Ex: remover 9

retorna 0



Operação de Remoção (Lista Ordenada)

- Implementação em C:

```
int remove_ord (Lista lst, int elem) {
    if (lst == NULL || lista_vazia(lst) == 1 ||
        elem < lst->no[0] || elem > lst->no[lst->fim-1])
        return 0; // Falha
    int i, Aux = 0;

    // Percorre até achar o elem ou nó maior, ou final de lista
    while (Aux < lst->Fim && lst->no[Aux] < elem)
        Aux++;
}
```



Operação de Remoção (Lista Ordenada)

- Implementação em C:

```
int remove_ord (Lista lst, int elem) {
    if (lst == NULL || lista_vazia(lst) == 1 ||
        elem < lst->no[0] || elem > lst->no[lst->fim-1])
        return 0; // Falha
    int i, Aux = 0;

    // Percorre até achar o elem ou nó maior, ou final de lista
    while (Aux < lst->Fim && lst->no[Aux] < elem)
        Aux++;
}
```

Operação de Remoção (Lista Ordenada)

- Implementação em C:

```
...
if (Aux == lst->Fim || lst->no[Aux] > elem) // Não elem
    return 0; // Falha

// Deslocamento à esq. do sucessor até o final da lista
for (i = Aux+1; i < lst->Fim; i++)
    lst->no[i-1] = lst->no[i];

lst->Fim--; // Decremento do campo Fim
return 1; // Sucesso
}
```

Operação de Remoção (Lista Ordenada)

- Implementação em C:

```
...
if (Aux == lst->Fim || lst->no[Aux] > elem) // Não elem
    return 0; // Falha

// Deslocamento à esq. do sucessor até o final da lista
for (i = Aux+1; i < lst->Fim; i++)
    lst->no[i-1] = lst->no[i];

lst->Fim--; // Decremento do campo Fim
return 1; // Sucesso
}
```

Exercícios

1. Implementar, utilizando a alocação estática e o acesso seqüencial, o TAD lista linear não ordenada de números inteiros. Nessa implementação a lista deve ter no máximo 20 elementos e deve contemplar as operações básicas: *criar_lista*, *lista_vazia*, *lista_cheia*, *insere_elem*, *remove_elem* e *obtem_valor_elem*. Além disso, desenvolva um programa aplicativo que permita ao usuário inicializar uma lista, inserir e remover elementos e imprimir a lista.

Teste este programa com a seguinte seqüência de operações:

- Inicialize a lista
- Imprima a lista
- Insira os elementos {4,8,-1,19,2,7,8,5,9,22,45};
- Imprima a lista
- Remova o elemento 8
- Imprima a lista
- Inicialize a lista
- Imprima a lista

2. Repita a implementação acima para o TAD lista ordenada.

Exercícios

3. Altere a implementação do exercício 1 para contemplar uma lista não ordenada de bebidas, com a seguinte estrutura:

Nome char[20]	Volume (ml) int	Preço float
------------------	--------------------	----------------

Crie um programa aplicativo similar àquele desenvolvido nos exercícios de alocação dinâmica, ou seja, com as seguintes opções:

- [1] Inserir registro
- [2] Apagar último registro
- [3] Imprimir tabela
- [4] Sair



Referências

✓ Básica

- CELES, W., CERQUEIRA, R. e RANGEL, J. L. “Introdução a estruturas de dados”. Campus Elsevier, 2004.
- TENENBAUM, A. M., LANGSAM, Y. e AUGENSTEIN, M.J. “Estrutura de Dados Usando C”. Makron Books.

✓ Extra

- BACKES, André. “Programação Descomplicada Linguagem C”. Projeto de extensão que disponibiliza vídeo-aulas de C e Estruturas de Dados. Disponível em: <https://www.youtube.com/user/progdescomplicada>. Acessado em: 25/04/2022.

✓ Baseado nos materiais dos seguintes professores:

- Prof. André Backes (UFU)
- Prof. Bruno Travençolo (UFU)
- Prof. Luiz Gustavo de Almeida Martins (UFU)



Dúvidas?

Prof. Me. Claudiney R. Tinoco
profclaudineytinoco@gmail.com

Faculdade de Computação (FACOM)
Universidade Federal de Uberlândia (UFU)