

# *Tecnologia em Análise e Desenvolvimento de Sistemas – TADS*

## ***Estrutura de Dados***

***Prof. Luciano Vargas Gonçalves***  
*E-mail: [luciano.goncalves@riogrande.ifrs.edu.br](mailto:luciano.goncalves@riogrande.ifrs.edu.br)*



# Estrutura da Dados

## **Aula 4 – Estruturas Dados Dinâmicas**

- **Listas**
- **Pilhas**
- **Filas**

## **Listas Lineares**

- Simplesmente Encadeada – LSE**
- Duplamente Encadeada - LDE**

# Estrutura de Dados - Listas

- **LISTA LINEAR :**

- É uma coleção de **elementos** (Nós) do mesmo tipo, dispostos linearmente, que podem ou não seguir determinada organização.
  - Por exemplo: lista de telefone, lista carros, lista filmes, etc;
- A lista dinâmica é composta por **um número infinito** de elementos ( $E_0, E_1, E_2, \dots, E_n$ );
  - Onde “n” é a quantidade de elementos ( $n \geq 0$ );
- O fim da lista é marcado pela terminador (“**NULL**”);
- O **ponteiro próximo** e/ou anterior determinam o encadeamento dos elementos.



Lista Simplesmente Encadeada de Elementos

# Estrutura de dados - Listas

- **Tipos de Listas Encadeadas:**

- **Simples** (Deslocamento em apenas *um sentido* – *Um ponteiro = “Próximo”*)



- **Duplamente** (Deslocamento em *ambos sentidos* – *dois ponteiros “Anterior e Próximo”*)

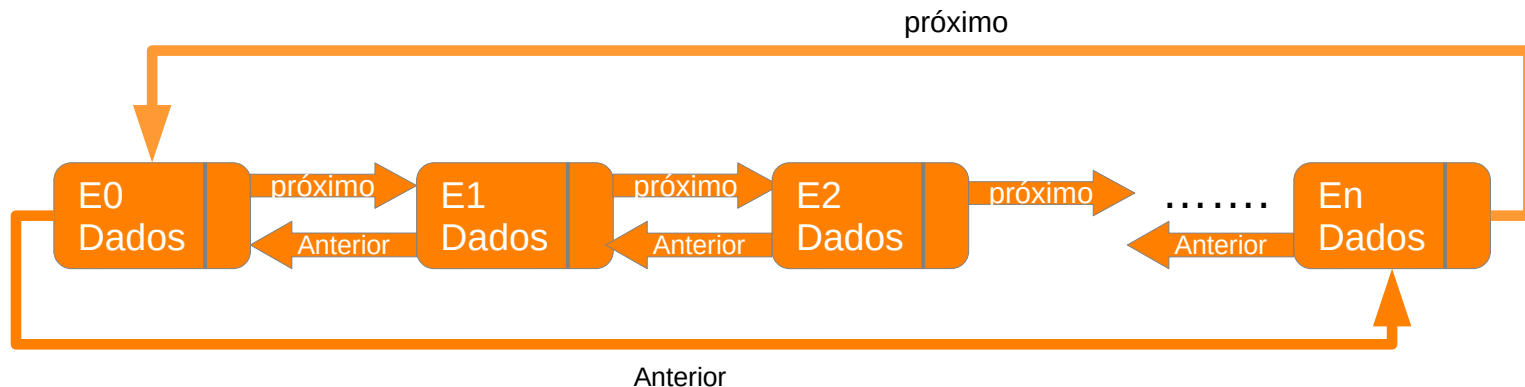


# Estrutura de dados - Listas

- **Tipos de Listas Simplesmente Encadeadas:**

- **Circulares**

- Ponteiros para Anterior e Próximo não apontam para NULL nos extremos, apontam para o primeiro e último.
    - Não possuem terminadores (NULL)



# Estrutura de dados - Listas

- **Quando falamos de listas dinâmicas:**
  - Precisamos pensar nos **elementos da estrutura (informações a serem armazenadas)**:
    - Ex:
      - Lista de Chamada: os elementos são os alunos matriculados;
      - Lista de Carros: os elementos são os carros;
  - Precisamos pensar **na interface da lista**:
    - Ex:
      - Lista de Chamada: a estrutura é definida pelas marcações, que determinam **início e fim** da lista,
      - Funções de Inserção / Remoção e Consulta de elementos.

# Elemento de Lista - Nó

- **Elemento de Lista ou Nó da lista**

- É uma estrutura de dados que irá armazenar:

- **DADOS:** As informações do elemento da lista (dados);

- **PONTEIRO(S): Anterior e/ou Próximo da lista;**

- Exemplo:

- A Lista de chamada armazena Alunos;

- A Lista de Filmes armazena títulos de filmes;

- A Lista de tarefas armazena as atividades pendentes;



Elemento de Lista



# Elemento de Lista

- Exemplo: Lista de Chamada de Alunos
  - O elemento será Aluno;
  - Onde iremos armazenar as informações de um Aluno:
    - Nome, idade, matricula;

Aluno
Nome Idade Matrícula

Elemento Aluno

# Elemento de Lista

- Elementos na memória RAM;
  - Os elementos ficam dispersos na memória quando alocados dinamicamente;
  - Cada elemento é alocado no espaço disponível no momento da alocação;
  - Exemplo:
    - Alunos: João, Maria e Paula

Endereço	Dados
xxx01	Aluno João
xxx02	....
xxx03	.....
.....	
xxx50	....
xxx51	Aluna Maria
.....	
.....	...
xxx81	Aluna Paula
xxx82	

Ex: Memória RAM

# Listas Simplesmente Encadeada - LSE

- **Lista Simplesmente Encadeada – LSE**

- Características:

- Navegação em apenas **um sentido**;
    - Apenas **um ponteiro** para o encadeamento (**Anterior ou Próximo**)
    - Exemplo:

- Lista Crescente ponteiro “\*Próximo”

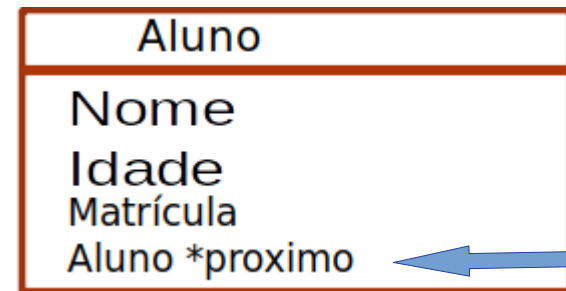


- Lista Decrescente ponteiro “\*Anterior”



# Listas Simplesmente Encadeada - LSE

- **LSE - Ponteiro próximo:**
  - Para apontar para o próximo elemento iremos criar um apontador que irá armazenar uma referência para o “**próximo**” elemento da lista (LSE).
  - Por exemplo, no caso de uma lista de Alunos (\*próximo).
    - **Próximo** é um ponteiro de encadeamento (ligação) para formação da lista.



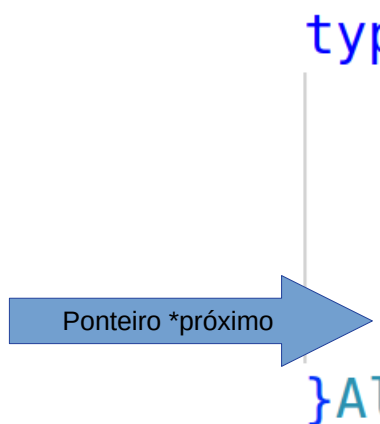
```
typedef struct Aluno{  
    char nome [20];  
    int idade;  
    int matricula;  
    struct Aluno *proximo;  
}Aluno;
```

# Listas Simplesmente Encadeada - LSE

- **Estrutura de Dados para armazenar Alunos**

- Elemento **Aluno** armazena os dados de um aluno e o ponteiro para o próximo **Aluno** (*\*proximo*);

```
typedef struct Aluno{  
    char nome [20];  
    int idade;  
    int matricula;  
    struct Aluno *proximo;  
}Aluno;
```



Exemplo de estrutura para armazenar dados de um Aluno

# Listas Simplesmente Encadeada - LSE

- **Estrutura para Listas Simplesmente Encadeada – LSE**

- Tamanho da lista ( *$n^{\circ}$  de elementos*)
  - Lista pode ter de 0 ou mais elementos encadeados ( $n$ );
- *Pontos de acesso e a navegação :*
  - **Ponteiro \*Primeiro** marca o início da Lista;
  - **Ponteiro \*Último** marca o fim da lista (facultativo)
  - **N = quantidade de elementos**



# Estrutura de Dados para LSE

- **Definição da Estrutura da LSE**
  - Estruturas de dados (LSE) para gerenciar uma Lista Simplesmente Encadeada.
    - **Exemplo 1, apenas um ponteiro para o início da lista:**

```
typedef struct LSE{  
    Aluno *primeiro;  
    int n;  
}LSE;
```

Início da lista (ponteiro \*primeiro )

Quantidade de elementos

# Criar uma nova Lista(LSE)

- **Declarar uma nova lista como Estrutura**
  - Uma nova lista LSE para turma de matemática.

```
//declaração da Lista  
LSE matematica;
```

```
//declarar um ponteiro de lista e alocar memória  
LSE *portugues = (LSE *) malloc(sizeof(LSE));
```



# Função para uma LSE

- Criar uma Lista (criaLista)
  - Inicializa os parâmetros de nova Lista alocada estaticamente
  - Recebe um ponteiro de Lista (\*ls)
  - Inicializa o ponteiro **\*primeiro** e **quantidade** de elementos para lista vazia.

```
void criaLista(LSE *ls){  
    ls->primeiro = NULL;  
    ls->n = 0;  
}
```

# Programa Principal – Main ()

- Novo Elemento de lista “Joao”
  - Declara e inicializa um novo aluno;

```
//declara e inicializa um novo elemento de lista de alunos  
Aluno joao = {"Joao Pedro", 23, 12345};  
joao.proximo = NULL;
```

# Função para mostrar um Aluno

- Função Mostra Aluno
  - Recebe um aluno e mostra os dados do aluno;

```
void mostraAluno(Aluno aluno){  
    printf("\n Dados do Aluno:");  
    printf("\n\t Nome: %s",aluno.nome);  
    printf("\n\t Idade: %d",aluno.idade);  
    printf("\n\t Matricula: %d\n",aluno.matricula);  
}
```

# Programa Principal – Main ()

- Função Mostra Aluno

```
void mostraAluno(Aluno aluno){  
    printf("\n Dados do Aluno:");  
    printf("\n\t Nome: %s",aluno.nome);  
    printf("\n\t Idade: %d",aluno.idade);  
    printf("\n\t Matricula: %d\n",aluno.matricula);  
}
```

Nome:Pedro

Idade:44

Matricula:1123301

Saída no terminal

# Cria um novo Elemento de Lista

- **Função cadastraAluno()**

- Recebe um ponteiro Aluno e os dados do novo aluno (nome,idade,matricula)

```
void cadastraAluno (Aluno *aluno, char nome[],int idade, int matricula)
{
    strcpy(aluno->nome, nome);
    aluno->idade = idade;
    aluno->matricula = matricula;
    aluno->proximo = NULL;
}
```

# Cria um novo Elemento de Lista

- **Função cadastraAluno()**

- Alocar um novo aluno e preencher os dados

```
//novo aluno declarado  
Aluno paulo;
```

```
//casdatrar um novo aluno por meio de uma função  
cadastraAluno(&paulo,"Paulo Roberto",34,232323);
```

```
void cadastraAluno (Aluno *aluno, char nome[],int idade, int matricula)  
{  
    strcpy(aluno->nome, nome);  
    aluno->idade = idade;  
    aluno->matricula = matricula;  
    aluno->proximo = NULL;  
}
```

# Interface da LSE

- A interface da lista implementa as funções de:

- **Funções de Inserção:**

- InsereNoInicio(), InsereNoFim(), InsereNaPosição()

- **Funções de Remoção:**

- RemoveNoInicio(), RemoveNoFim(),  
RemoveNaPosição();

- **Funções de Consulta:**

- MostraElemento( ), MostraLista()

- **Funções de Exclusão:**

- ApagaElemento(), ApagaLista()

- **OBS:**

***A Lista LSE não possui restrição para inserção e remoção.***

# Funções de Inserção

- Como Inserir um novo elemento na Lista, Funções:
  - **InserNoInicio():**
    - Insere um novo elemento na primeira **posição (E0)**;
  - **InserNoFim()**
    - Insere um novo elemento na última **posição (En)**;
  - **InserNaPosição(posicao)**
    - Insere um novo elemento na **posição (posição)**;
- **ATENÇÃO:** Condições a serem avaliadas na inserção de elementos:
  - Lista vazia ( $n\_elementos = 0$ );
  - Lista com elementos ( $n\_elementos > 0$ );

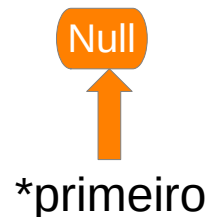


# Interface da LSE – Insere no Início

- Método para inserir um **novο** elemento no **Início** na Lista:
  - Duas situações precisam ser avaliadas;

## 1) Lista vazia;

- Ponteiro (**\*primeiro == NULL**);
- Número elementos (**n == 0**)



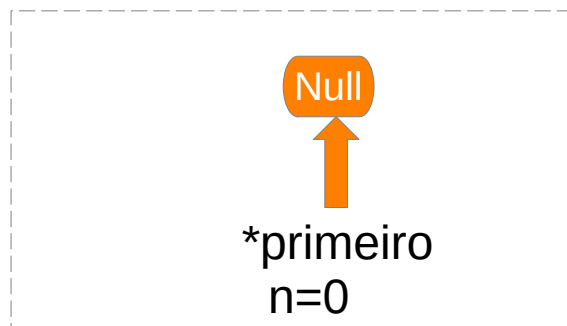
## 2) Lista com elementos;

- Ponteiro (**\*primeiro != NULL**);
- Número de elementos (**n > 0**);

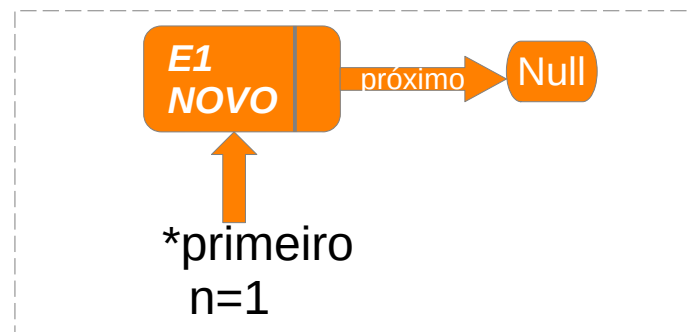


# Interface da LSE – Insere no Início

- Método para inserir um **novο** elemento no **Início** na Lista
  - Estado Atual – Lista Vazia



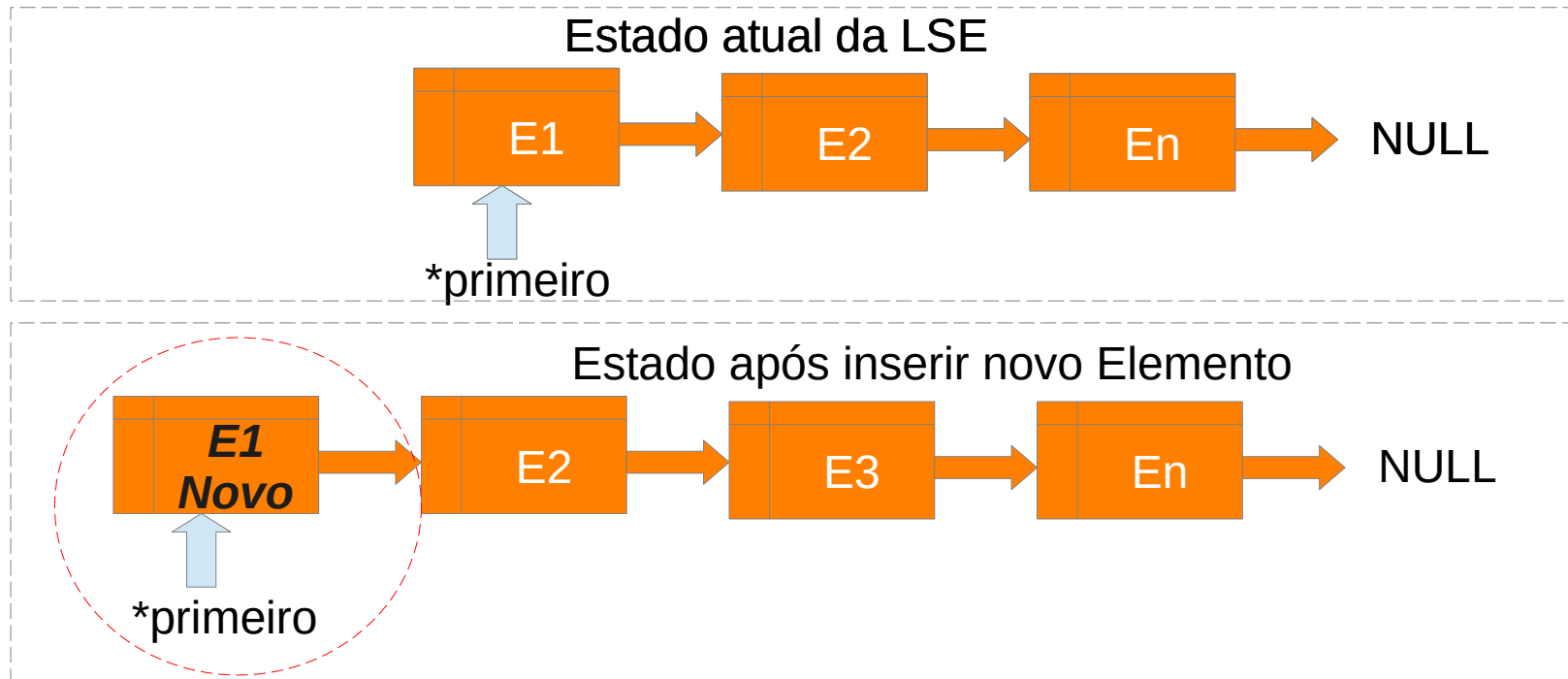
Antes da Inserção



Após inserção

# Interface da LSE – Insere no Início

- Método para inserir um **novο** elemento no **Início** na Lista
  - Estado Atual – Lista COM ELEMENTOS ( $N > 0$ );



# Interface da LSE – Insere no Início

## Método para inserir um *novο* elemento no *Início* na Lista:

- A função recebe um ponteiro de lista (*\*ls*) e um ponteiro de elemento (*\*aluno*)
- Situações a serem avaliadas;

### 1) Lista vazia;

- Ponteiro (\*primeiro == NULL);
- Número elementos (n == 0)

### 2) Lista com elementos;

- Ponteiro (\*primeiro != NULL);
- Número de elementos (n > 0);

```
void insereInicio(LSE *lista, Aluno *novo){  
    if(lista->primeiro == NULL){  
        //Lista Vazia  
        novo->proximo = NULL;  
    }else{  
        //Lista com elementos  
        novo->proximo = lista->primeiro;  
    }  
    lista->primeiro = novo;  
    lista->n_elementos++;  
}
```

- A função recebe dois ponteiros, um para lista (\*ls) e um para novo elemento (\*novo);

# Interface da LSE – Insere no Início

- **Método para inserir um *novο* elemento no *Início* na Lista:**
  - A função insere no início reduzida..
  - A validação para lista vazia na função insere no início é facultativa.

```
void insereInicioReduzida(LSE *lista, Aluno *novo){  
    novo->proximo = lista->primeiro;  
    lista->primeiro = novo;  
    lista->n_elementos++;  
}
```

# Interface da LSE – Insere no Início

- Método para inserir um **novο** elemento no **Início** na Lista
  - Estado Atual – Lista Vazia

Ls - Antes da Inserção



```
void insereInicioReduzida(LSE *lista, Aluno *novo){  
    novo->proximo = lista->primeiro;  
    lista->primeiro = novo;  
    lista->n_elementos++;  
}
```



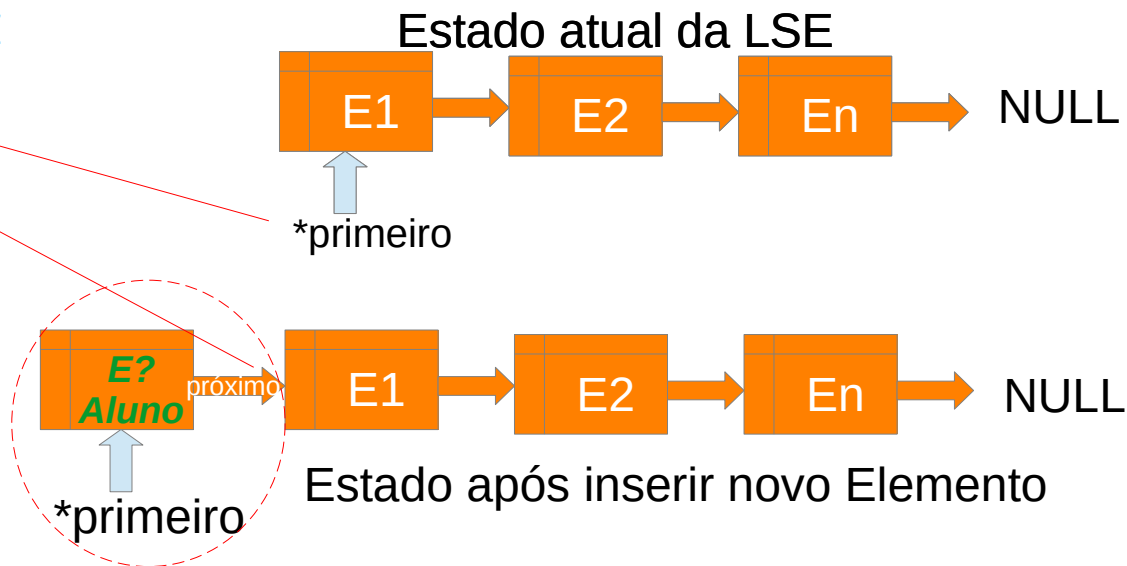
\*primeiro

Ls - Novo estado após inserção

# Interface da LSE – Insere no Início

- Método para inserir um **novο** elemento no **Início** na Lista
  - Estado Atual – Lista COM ELEMENTOS ( $n > 0$ )

```
void insereInicioReduzida(LSE *lista, Aluno *novo){  
    novo->proximo = lista->primeiro;  
    lista->primeiro = novo;  
    lista->n_elementos++;  
}
```



# Interface da LSE – Insere no Início

- Programa principal inserir novo elemento na LSE;

```
//função para inserir um novo aluno na lista de matemática
```

```
insereInicio(&matematica,&joao);
```

```
insereInicio(&matematica,&paulo);
```

```
void insereInicio(LSE *lista, Aluno *novo){  
    if(lista->primeiro == NULL){  
        //Lista Vazia  
        novo->proximo = NULL;  
    }else{  
        //Lista com elementos  
        novo->proximo = lista->primeiro;  
    }  
    lista->primeiro = novo;  
    lista->n_elementos++;  
}
```



# Vetor de Alunos

- Criado um vetor com 4 novo alunos

```
Aluno novosAlunos [4];  
cadastraAluno(&novosAlunos[0], "Luiz", 25, 12133);  
cadastraAluno(&novosAlunos[1], "Julio", 28, 12132);  
cadastraAluno(&novosAlunos[2], "Pedro", 29, 12134);  
cadastraAluno(&novosAlunos[3], "Maria", 22, 12135);
```

# Adicionado os novos alunos na Lista

- 4 novos alunos adicionados na lista de matemática

```
insereInicio(&matematica,&novosAlunos[0]);  
insereInicio(&matematica,&novosAlunos[1]);  
insereInicio(&matematica,&novosAlunos[2]);  
insereInicio(&matematica,&novosAlunos[3]);
```

# Interface da LSE – Insere no Início

- Método para inserir um *novο* elemento no *Início* na Lista

Estado atual da LSE



\*primeiro

Estado atual da LSE – Após inserção



\*primeiro

# Interface da LSE – Mostra Lista

- **MostraLista( )**

- Mostra todos elementos a partir do primeiro elemento;
- Necessário para varrer a lista (iterar);

```
void mostraLista(LSE lista){  
    printf("\n Mostra Lista Simplesmente Encadeada; \n");  
    Aluno *aux; //ponteiro auxiliar  
    aux = lista.primeiro;  
    for(int i=0;i<lista.n_elementos;i++){  
        mostraAluno(*aux);  
        aux = aux->proximo;  
    }  
    printf("\n Fim da Lista!\n");  
}
```

# Interface da LSE – Mostra Lista

- **MostraLista( )**

```
void mostraLista(LSE lista){  
    printf("\n Mostra Lista Simplesmente Encadeada; \n");  
    Aluno *aux; //ponteiro auxiliar  
    aux = lista.primeiro;  
    for(int i=0;i<lista.n_elementos;i++){  
        mostraAluno(*aux);  
        aux = aux->proximo;  
    }  
    printf("\n Fim da Lista!\n");  
}
```

Mostra Lista Simplesmente Encadeada;

Aluno Nome: Maria Idade 21 Matricula 334343

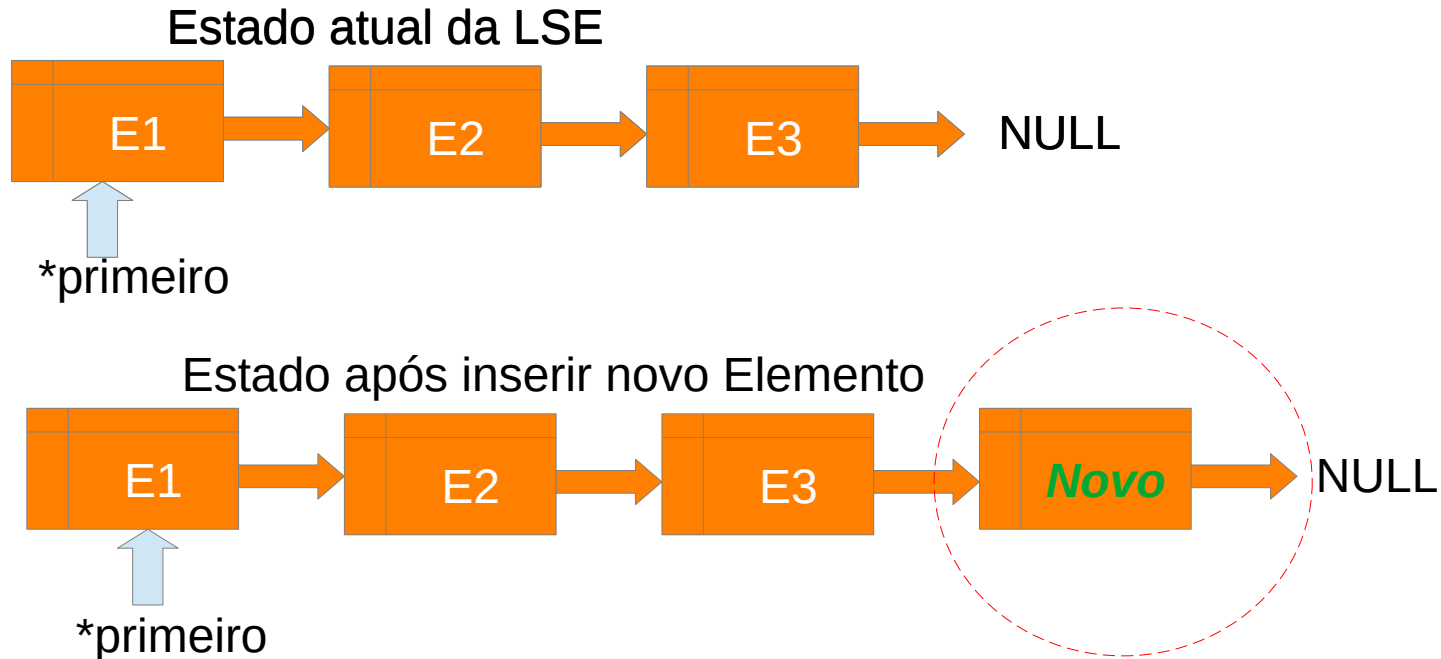
Aluno Nome: Paulo Roberto Idade 34 Matricula 232323

Aluno Nome: Joao Pedro Idade 23 Matricula 12345

Fim da Lista!

# Interface da LSE – Insere no FIM

- Método para inserir um **novο** elemento no **Final** na Lista

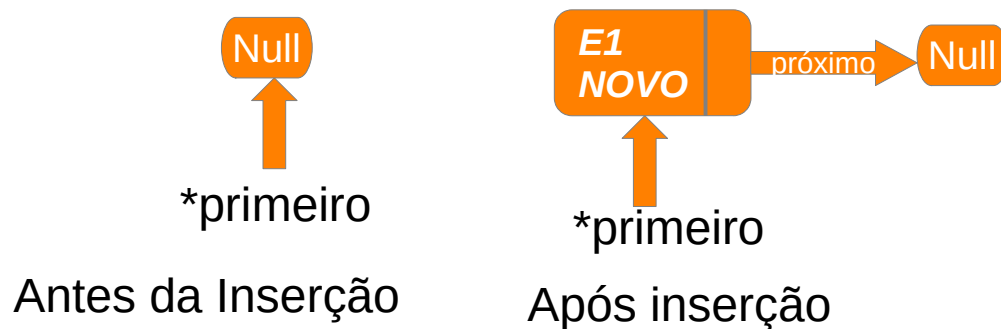


# Interface da LSE – Insere no FIM

- **Inserir um elemento no *FIM* da Lista**
  - Função InsereNoFim()
    - Recebe um ponteiro de lista *\*lista* e um ponteiro de Aluno *\*novo*;
    - Função sem retorno;
    - Duas possibilidades para inserção :
      - 1) Lista Vazia antes da inserção (*n = 0*);
      - 2) Lista com elementos (*n > 0*);

# Interface da LSE – Insere no FIM

- Método para inserir um **novο** elemento no **FIM** na Lista
  - Estado Atual – Lista Vazia
  - Procedimento idêntico ao Insere no Início

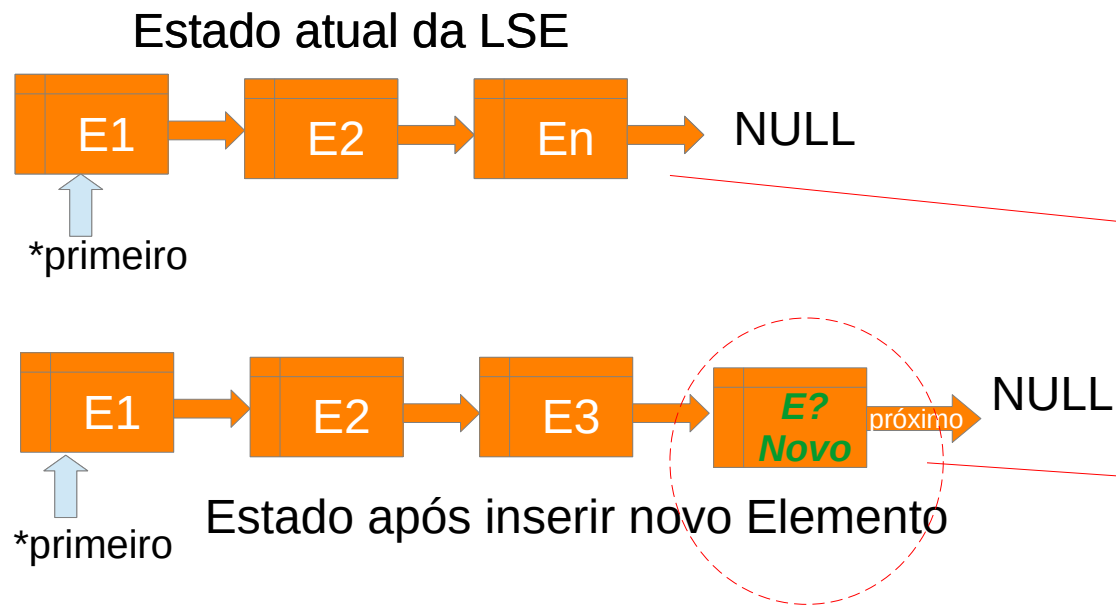


```
void insereFim(LSE *lista, Aluno *novo){
    novo->proximo = NULL;
    if(lista->primeiro == NULL){
        //Lista Vazia
        lista->primeiro = novo;
    }else{
        //Lista com elementos
        ...
    }
}
```



# Interface da LSE – Insere no FIM

- Método para inserir um **novο** elemento no **FIM** na Lista
  - Estado Atual – Lista COM ELEMENTOS



```
void insereFim(LSE *lista, Aluno *novo){
    novo->proximo = NULL;
    if(lista->primeiro == NULL){
        //Lista Vazia
        lista->primeiro = novo;
    }else{
        //Lista com elementos
        Aluno *aux = lista->primeiro;
        while(aux->proximo!=NULL){
            aux = aux->proximo;
        }
        aux->proximo = novo;
    }
    lista->n_elementos++;
}
```

# Funções de Remoção

- Como remover um novo elemento na Lista, Funções:
  - **Elemento \* RemoçãoNoInicio(Lista \*):**
    - Remove um elemento no início da lista;
  - **Elemento \* RemoçãoNoFim(Lista \*):**
    - Remove um elemento no fim da lista;
  - **Elemento \* RemoçãoNaPosicao(Lista ls\*, int pos):**
    - Remove um elemento em uma posição específica (pos);
  - **Condições a serem avaliadas:**
    - Lista vazia ( $n = 0$ );
    - Lista com elementos ( $n > 0$ );

- **Atenção:**

Remover é diferente de excluir um elemento, remover retorna o elemento removido.

# Interface da LSE – Insere no FIM

- **RemoveNoInicio ( )**

- Remove o primeiro elemento da Lista e retorna o endereço do elemento removido;

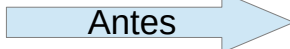
```
Aluno* removeInicio(LSE *lista){  
    Aluno *aux = lista->primeiro;  
    if(aux != NULL){  
        //lista com um ou mais elementos//  
        lista->primeiro = aux->proximo;  
        lista->n_elementos--;  
    }  
    return aux;  
}
```

# Listas Simplesmente Encadeada

- RemoveNoInicio ( )

```
Aluno* removeInicio(LSE *lista){
    Aluno *aux = lista->primeiro;
    if(aux != NULL){
        //lista com um ou mais elementos//
        lista->primeiro = aux->proximo;
        lista->n_elementos--;
    }
    return aux;
}
```

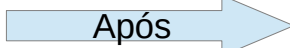
Mostra Lista Simplesmente Encadeada;

 Antes

Aluno Nome: Maria Idade 21 Matricula 334343  
Aluno Nome: Paulo Roberto Idade 34 Matricula 232323  
Aluno Nome: Joao Pedro Idade 23 Matricula 12345  
Aluno Nome: Juca Silva Idade 25 Matricula 3232323  
Total de alunos na Lista = 4  
Fim da Lista!

Elemento removido foi:  
Aluno Nome: Maria Idade 21 Matricula 334343

Mostra Lista Simplesmente Encadeada;

 Após

Aluno Nome: Paulo Roberto Idade 34 Matricula 232323  
Aluno Nome: Joao Pedro Idade 23 Matricula 12345  
Aluno Nome: Juca Silva Idade 25 Matricula 3232323  
Total de alunos na Lista = 3  
Fim da Lista!

# Tarefa

- Implementar as Funções da Lista LSE:
  - RemoveNoFim():
    - Recebe um ponteiro de lista e remove o elemento na última posição;
    - Retorna um ponteiro \*Aluno
  - RemoveNaPosição():
    - Recebe um ponteiro de lista, uma posição e remove o elemento na posição;
    - Retorna um ponteiro \*Aluno
  - InsereNaPosição(int posicao):
    - Recebe um ponteiro de lista, uma posição(pos) e um ponteiro de Aluno;
    - Função sem retorno.

# Tarefa

- LSE.H

Protótipo das funções  
de LSE;

```
void insereNoInicio(LSE *ls, Aluno *novo);  
void insereNoFim(LSE *ls, Aluno *novo);  
void insereNaPosicao(LSE *ls, Aluno *novo, int pos);
```

```
Aluno* removeNoInicio(LSE *ls);  
Aluno* removeNoFim(LSE *ls);  
Aluno* removeNaPosicao(LSE *ls, int pos);
```

```
int retornaQuantidade(LSE *ls);
```

```
void mostraLista(LSE *ls);  
void mostraAluno(Aluno *novo);  
void mostraPosicao(LSE *ls, int pos);
```

```
void apagaLista(LSE *ls);  
void apagaAluno(Aluno *novo);
```

# Tarefa

- Função Menu
  - Controla a execução da Lista

```
int menu(LSE *ls){
    printf("1 - Inserir No Início:\n");
    printf("2 - Inserir No Fim:\n");
    printf("3 - Inserir No Posição:\n");
    printf("4 - Remove No Início:\n");
    printf("5 - Remove No Fim:\n");
    printf("6 - Remove No Posição:\n");
    printf("7 - Mostra Lista:\n");
    printf("8 - InsereOrdenado \n");
    printf("0 - Sair Programa:\n");
    int op,pos;
    scanf("%d",&op);
    DadosAluno *novo;
    if(op==1){
        novo = lerDadosAluno();
        insereNoInicio(ls,novo);
    }else if(op == 2){
        novo = lerDadosAluno();
        insereNoFim(ls,novo);
    }else if(op == 3){
        printf("\nInforme a Posição:");
        scanf("%d",&pos);
        novo = lerDadosAluno();
        insereNaPosicao(ls,novo,pos);
    }else if(op == 4){
        novo = removeNoInicio(ls);
    }else if(op == 5){
        novo = removeNoFim(ls);
    }else if(op == 6){
        printf("\nInforme a Posição:");
        scanf("%d",&pos);
        removeNaPosicao(ls,pos);
    }else if(op == 7){
        mostraLista(ls);
    }else{
        printf("\nFim da Execução!\n");
    }
    return op;
}
```

# Tarefa

- Execução da Lista

```
luciano@luciano-pc:~/Documentos/IFRS-2021/ED/Code-C/Aula4/Referencia$ ./roda
```

Nome:Pedro	Idade:34	Matricula:1123301
Nome:Paulo	Idade:44	Matricula:1123301
Nome:Maria	Idade:54	Matricula:1123301
Nome:Paula	Idade:24	Matricula:1123301

Fim da Lista

Remove no Início:

Nome:Paulo	Idade:44	Matricula:1123301
Nome:Maria	Idade:54	Matricula:1123301
Nome:Paula	Idade:24	Matricula:1123301

Fim da Lista

- 1 - Inserir No Início:
- 2 - Inserir No Fim:
- 3 - Inserir No Posição:
- 4 - Remove No Início:
- 5 - Remove No Fim:
- 6 - Remove No Posição:
- 7 - Mostra Lista:
- 0 - Mostra Lista:



# Material de Apoio

- IFRS – Pergamum
  - SZWARCFITER, Jayme Luiz. Estruturas de dados e seus algoritmos. 3. Rio de Janeiro LTC 2010 1 recurso online ISBN 978-85-216-2995-5.
  - DEITEL, Paul; Deitel, Harvey. C: como programar - 6ª edição. Editora Pearson 850 ISBN 9788576059349.