



Estrutura de Dados

Prof. Marques Sousa
marques.sousa@ifsp.edu.br

Este material consiste de adaptações e extensões dos originais
gentilmente cedidos pelo Prof. Gilberto Viana - IFTM

Imagens retiradas do Google



Na Aula Passada...

- Tipos Abstratos de Dados (TAD);
- Estrutura de Dados:
 - Lista Linear

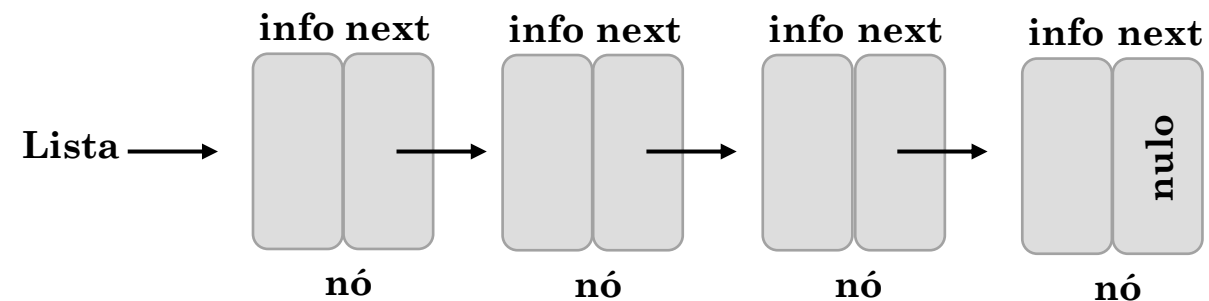
- Lista Linear
 - Implementação dinâmica.



- Estática:
 - Sequencial.
 - Quais os problemas de usarmos uma implementação estática?
 - É possível que uma implementação estática de uma lista seja mais eficiente do que uma implementação dinâmica?
- Dinâmica:
 - Blocos de memória são denominados **nós** ou **células**.
 - Itens ocupam células espalhadas por toda a memória.

Lista Encadeada

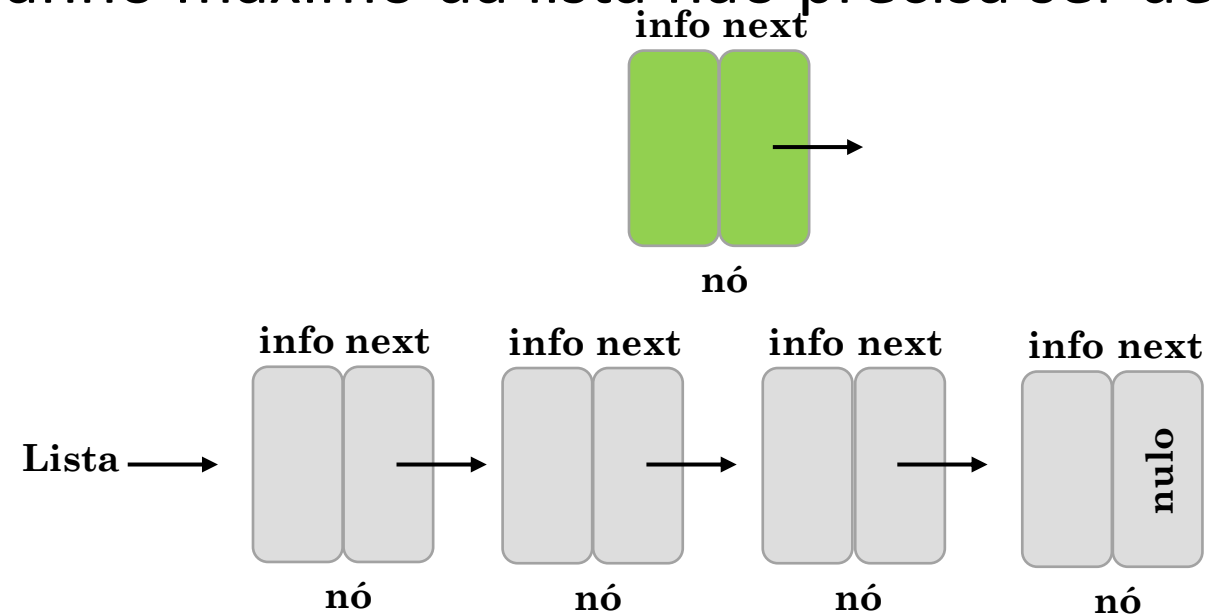
- Também conhecida como lista **ligada**.
- **Definição:** Estrutura de dados que mantém uma coleção de itens em ordem linear **sem exigir** que eles ocupem **posições consecutivas** de memória.
- **Itens** são armazenados em nós.
- Cada nó possui dois campos:
 - Informação;
 - Endereço para o próximo nó.



Lista Encadeada

- **Vantagens:**

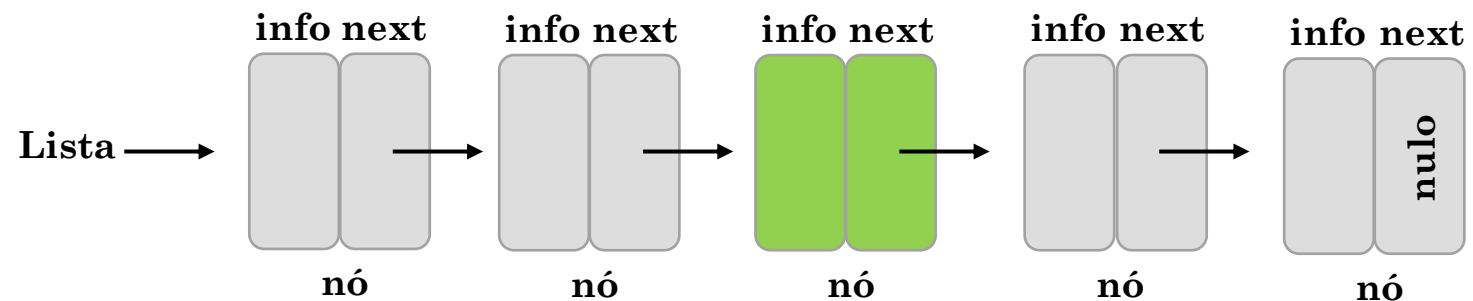
- Facilidade para inserção e remoção de itens em posições arbitrárias.
- Pouca movimentação de dados em memória.
- Útil em aplicações onde o tamanho máximo da lista não precisa ser definido com antecedência.



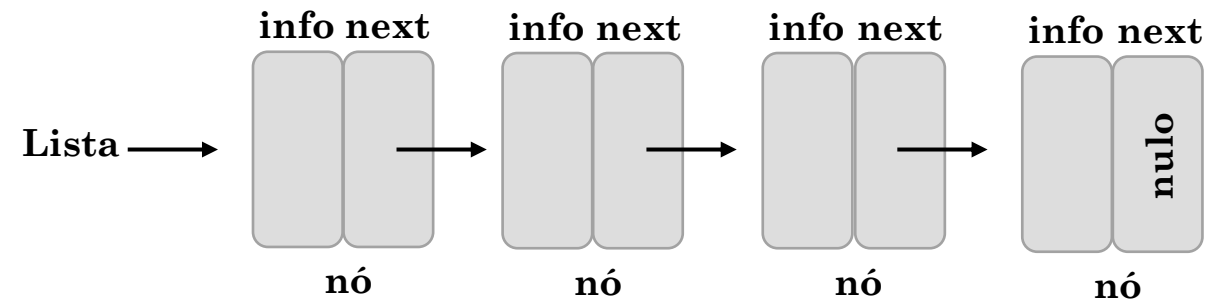
Lista Encadeada

- **Vantagens:**

- Facilidade para inserção e remoção de itens em posições arbitrárias.
- Pouca movimentação de dados em memória.
- Útil em aplicações onde o tamanho máximo da lista não precisa ser definido com antecedência.



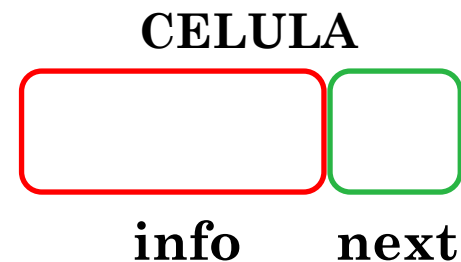
- **Lista Simplesmente Encadeada:**
 - A lista é percorrida em uma única direção;
 - Definir um nó para a lista linear dinamicamente.



Lista Encadeada (Implementação)

- A primeira estrutura a ser criada é a **célula** ou **nó**.
- Considerando que a informação armazenada para nossa lista seja do tipo **inteiro**, temos:

```
typedef struct sCell{  
    int info;  
    struct sCell *next;  
}CELULA;
```

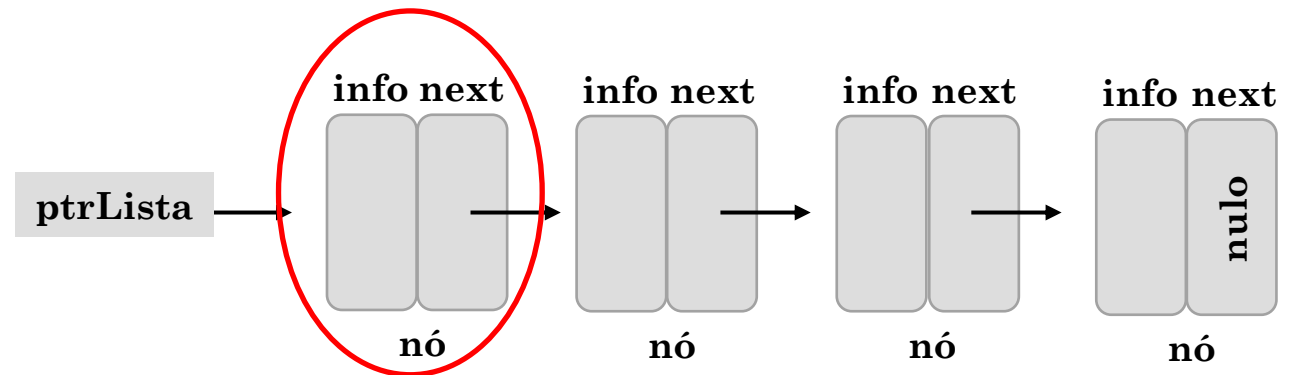


Lista Encadeada (Implementação)

- Podemos declarar uma célula da seguinte forma:

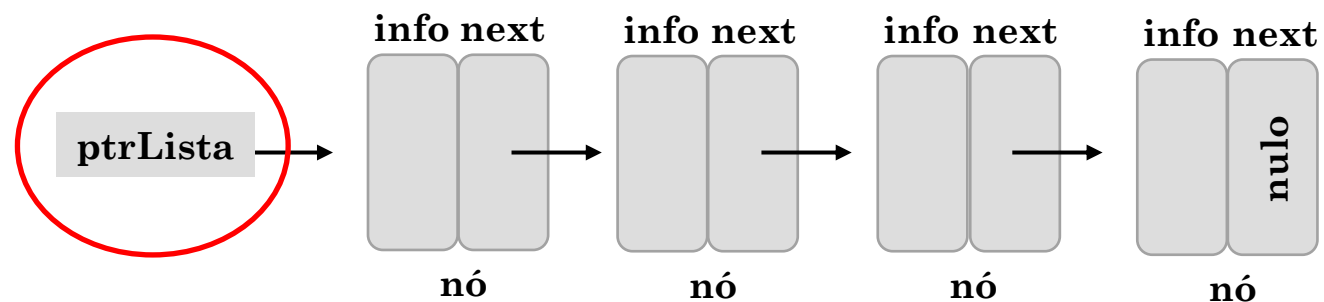
```
CELULA c;
```

- Assim temos:
 - c.info**: conteúdo de uma célula;
 - c.next**: endereço para a próxima célula.



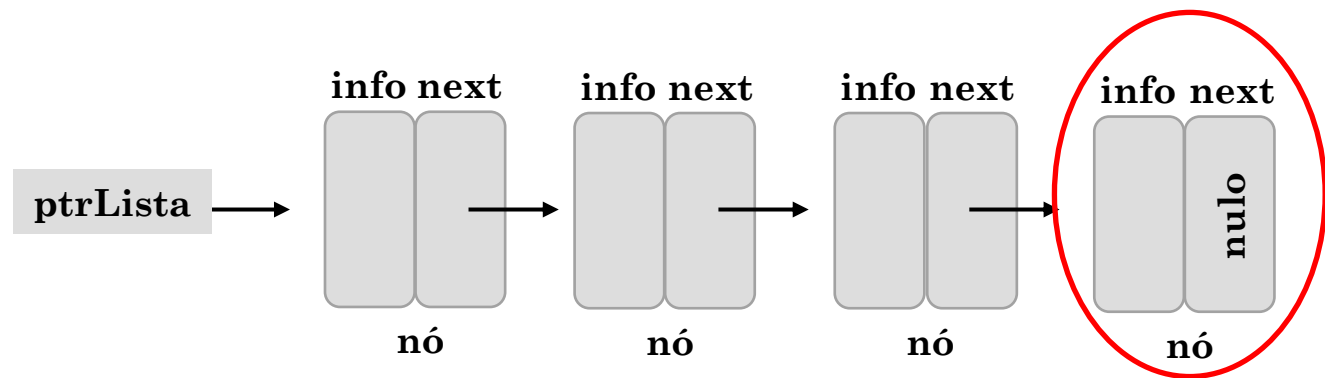
Lista Encadeada (Implementação)

- A lista encadeada é acessada a partir de um **ponteiro externo** *ptrLista* que aponta para o primeiro nó (célula) da lista. Logo:



Lista Encadeada (Implementação)

- O campo do próximo endereço (*next*) do último elemento da lista contém o valor do tipo NULL, indicando que não é um endereço válido.
- Esse ponteiro nulo é usado para identificarmos o final de uma lista.

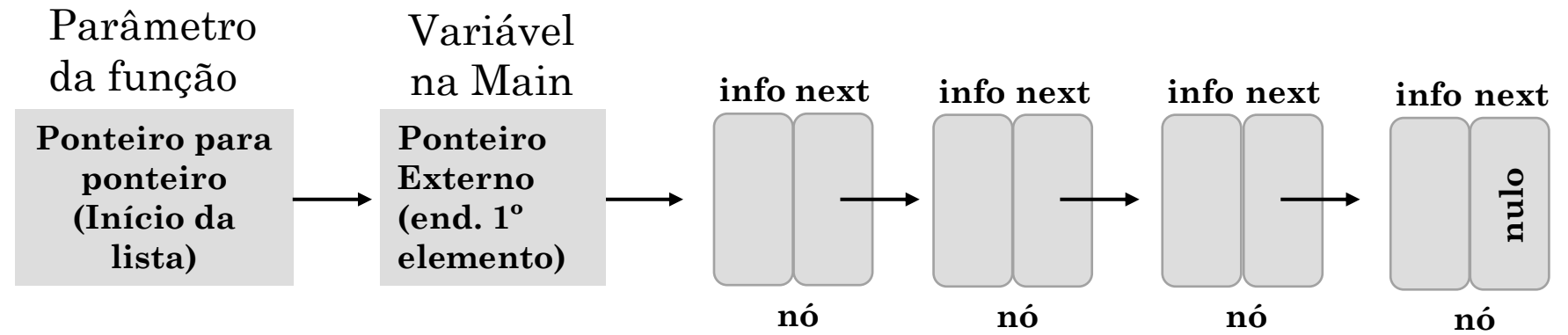


Lista Encadeada (Implementação)

- Ao implementarmos nossas funções, devemos conseguir acessar o 1º elemento da lista.
- Para facilitar a atualização da mesma, podemos receber o endereço desse 1º membro.
- Logo, podemos dizer que:
 - Cada elemento é tratado como um ponteiro alocado dinamicamente de acordo com a necessidade;
 - Para ter acesso à toda lista, passaremos o endereço da variável armazenada na main.

Lista Encadeada (Implementação)

- Temos então a estrutura final da nossa lista encadeada como sendo:



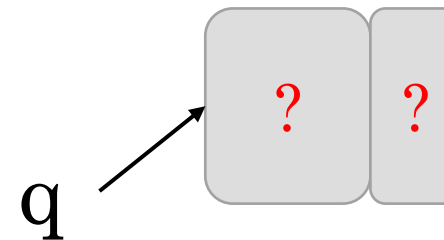
- **Início da lista:** ponteiro para ponteiro, pois armazena o endereço de um ponteiro.
- **Cada nó** da lista é um ponteiro (que será alocado dinamicamente).

Criar célula/nó

- Alocar dinamicamente um nó vazio para uma lista encadeada.

```
CELULA* criarCelula(){  
    CELULA *nova = (CELULA *) malloc(sizeof(CELULA));  
    return nova;  
}
```

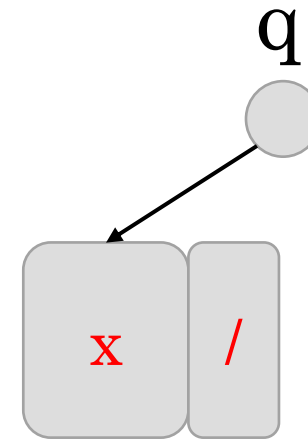
```
//Dentro de uma função  
CELULA *q = criarCelula();
```



Liberando Espaço de um nó

- **q** é o ponteiro que armazena o endereço do nó a ser liberado.

```
//Liberando o nó  
free(q);
```



Inicialização da Lista

- Inicializar o ponteiro externo à lista com o valor NULL.
- O endereço onde o nosso ponteiro para ponteiro aponta receberá o valor NULL.
- **lista** é um ponteiro para o ponteiro externo da lista, ou seja, **lista** armazena o endereço do ponteiro (Main) que armazena o endereço da primeira célula da lista (gerada dinamicamente).

```
void inicializarLista(CELULA **lista){  
    (*lista) = NULL;  
}
```


Lista Linear Dinâmica (Implementação)

- Verificar se a lista está vazia.



Lista Linear Dinâmica (Implementação)

- Verificar se a lista está vazia.



```
int listaVazia(CELULA **lista){  
    if((*lista) == NULL){  
        return 1;  
    }  
    return 0;  
}
```

Lista Linear Dinâmica (Implementação)

- Inserir no final da lista (Parte I).



Lista Linear Dinâmica (Implementação)

- Inserir no fim da lista (Parte I).

```
int inserirFim(CELULA **lista, int elemento){
    CELULA *novaCelula; //Armazena end. nova
    CELULA *auxiliar;    //Utilizado p/ percorrer

    //Aloca o espaço na mem. para novaCelula
    novaCelula = criarCelula();
    if(novaCelula == NULL){
        printf("\nERRO: Memoria Cheia");
        return 0;
    }
    //Colocando elemento dentro da célula
    novaCelula->info = elemento;
    novaCelula->next = NULL;
```

Lista Linear Dinâmica (Implementação)

- Inserir no fim da lista (Parte II).

```
//Se lista estiver vazia
```

```
if(listaVazia(lista)){  
    (*lista) = novaCelula;  
    return 1;  
}
```

```
//Auxiliar inicia busca pelo último elemento  
auxiliar = (*lista);
```

```
//Procurar última célula
```

```
while(auxiliar->next != NULL){  
    auxiliar = auxiliar->next;  
}
```

```
auxiliar->next = novaCelula;  
return 1;
```

```
}
```

Lista Linear Dinâmica (Implementação)

- Inserir no início da lista.



Lista Linear Dinâmica (Implementação)

- Inserir no início da lista.

```
int inserirInicio(CELULA **lista, int elemento){
    CELULA *novaCelula = criarCelula();
    if(novaCelula == NULL){
        printf("\nERRO: memoria cheia");
        return 0;
    }
    if(listaVazia(lista)){
        /*Ótimo, chama função inserir fim.*/
        return inserirFim(lista, elemento);
    }
    //Preenche a nova celula com o elemento.
    novaCelula->info = elemento;
    //next deve apontar para a primeira célula.
    novaCelula->next = (*lista);
    (*lista) = novaCelula;
    return 1;
}
```

Lista Linear Dinâmica (Implementação)

- Imprimir uma lista.



Lista Linear Dinâmica (Implementação)

- Imprimir uma lista.

```
void imprimirElemento(int elemento){  
    printf("%d ", elemento);  
}
```

```
void imprimirLista(CELULA **lista){  
    CELULA *auxiliar = (*lista);  
    if(listaVazia(lista)){  
        printf("\nLista Vazia.");  
    }else{  
        printf("\nLista: ");  
        while(auxiliar != NULL){  
            imprimirElemento(auxiliar->info);  
            auxiliar = auxiliar->next;  
        }  
        printf("\n");  
    }  
}
```

Testando Funções

- Algum problema com esse teste?

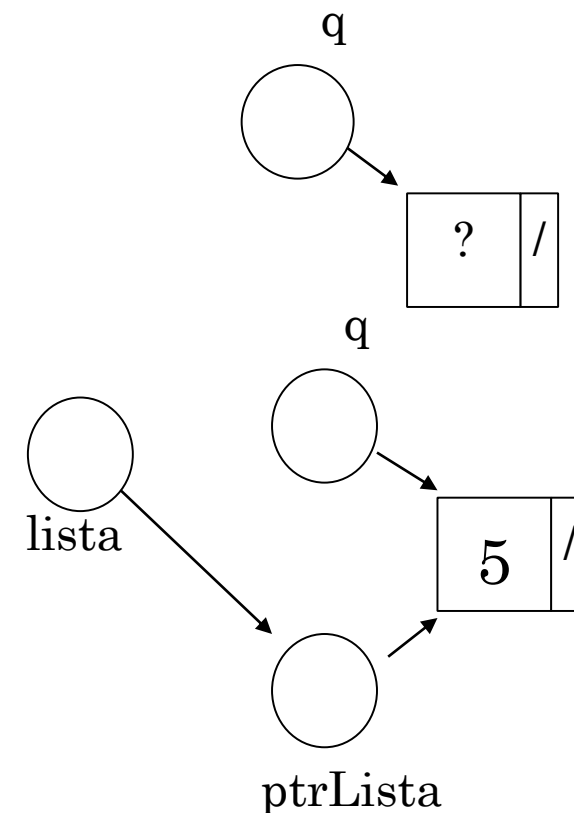
```
int main(){  
  
    CELULA *lista;  
    inicializarLista(&lista);  
    printf("\n%d", listaVazia(&lista));  
    inserirFim(&lista, 1);  
    imprimirLista(&lista);  
    inserirFim(&lista, 2);  
    imprimirLista(&lista);  
    inserirInicio(&lista, 0);  
    imprimirLista(&lista);  
    inserirInicio(&lista, -1);  
    imprimirLista(&lista);  
    inserirFim(&lista, 3);  
    imprimirLista(&lista);  
    return 0;  
}
```

Exemplo de Operações

- O que acontece quando criamos o primeiro nó?

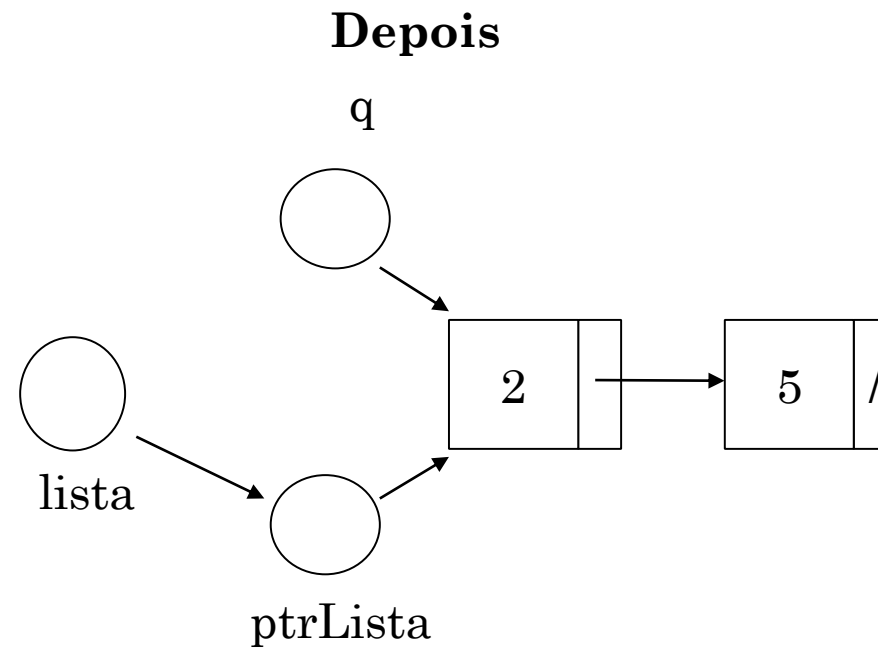
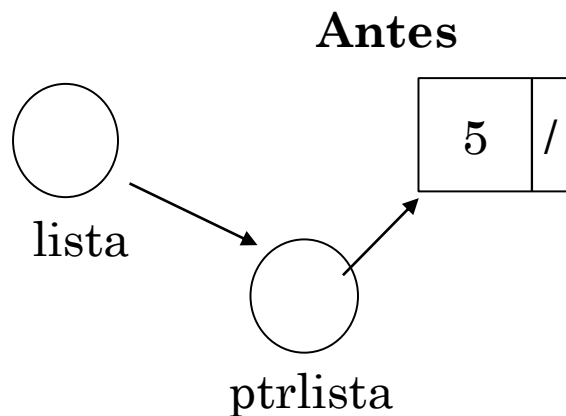
```
CELULA *q;  
q = criarCelula();  
if (q != NULL) {  
    q->info = 5; /* Elemento */  
}  
(*lista) = q;
```

→ Lembrando que **q** armazena o endereço do novo nó e **lista** armazena o endereço do ponteiro externo que aponta para o primeiro nó da lista encadeada.



Exemplo: Inserindo Início

```
q = criarCelula();  
if (q != NULL)  
{  
    q->info = 2; /* Elemento */  
    q->prox= (*lista);  
    (*lista) = q;  
}
```



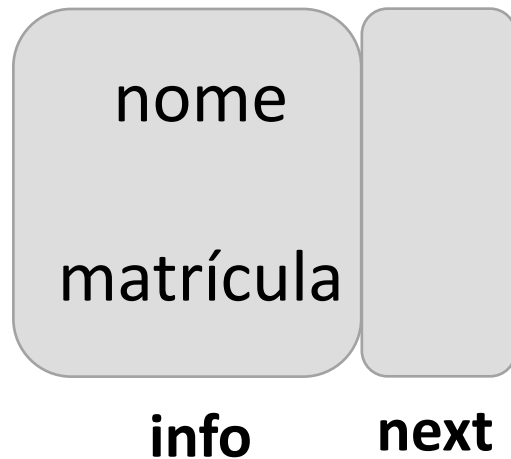
Lista Encadeada

- Assumiremos a partir de agora que o nosso elemento não será mais apenas um **int**.
- Logo, criaremos uma estrutura que será armazenada no nosso campo **info** dentro da célula.



Lista Encadeada

- Definiremos uma estrutura PESSOA que contém os campos:
 - Nome;
 - Matrícula.



```
typedef struct sPessoa{  
    char nome[50];  
    int matricula;  
} PESSOA;  
  
typedef struct sCell{  
    PESSOA info;  
    struct sCell *next;  
} CELULA;
```




Lista Encadeada

- Exemplo:
 - Antes nossa função **imprimirElemento**, imprimia apenas um valor do tipo inteiro.

```
void imprimirElemento(int elemento){  
    printf("%d ", elemento);  
}
```

- Agora ele deve imprimir todos os valores de uma elemento.

```
void imprimirElemento(PESSOA elemento){  
    printf("\nMatricula: %d \tNome: %s", elemento.matricula, elemento.nome);  
}
```

Da mesma forma, temos que nos atentar a todos os acessos ao campo **info** durante a implementação de cada uma das funções.

Remover Nó do Início (Parte 1)

```
PESSOA removerInicio(CELULA **lista){
    /*Ponteiro para armazenar o end. da
    célula a ser excluída.*/
    CELULA *removida;

    //Elemento vazio
    PESSOA removido;
    strcpy(removido.nome, " ");
    removido.matricula = -1;

    //Se lista vazia, então não remove.
    if(listaVazia(lista)){
        printf("\nERRO: Lista vazia");
        return removido;
    }
}
```

Remover Nó do Início (Parte 2)

```
//Guarda end. 1º elemento
removida = (*lista);
removido = removida->info;

//Faz a lista apontar para 2º elemento.
(*lista) = (*lista)->next;

//Remove o antigo 1º elemento
free(removida);
return removido;
}
```

Remover Nó do Final (Parte 1)

```
PESSOA removerFim(CELULA **lista) {  
    //Armazenaar o enedereço do nó a ser removido  
    CELULA *removida;  
  
    //Guarda o endereço do ní que passará a ser o último.  
    CELULA *anterior;  
  
    PESSOA removido = criarPessoa("", -1);  
  
    if(listaVazia(lista)) {  
        printf("Erro: lista vazia!\n");  
        return removido;  
    }  
    //Verifica se há apenas 1 elemento.  
    if((*lista)->next == NULL)  
        return removerInicio(lista);  
}
```

Remover Nó do Final (Parte 2)

*//O ponteiro removida percorrerá a lista para encontrar
//a posição onde deverá remover o item.*

```
removida = (*lista);  
while(removida->next != NULL){  
    anterior = removida;  
    removida = removida->next;  
}
```

```
removido = removida->info;  
anterior->next = NULL;  
free(removida);  
return removido;
```

```
}
```

- E se quisermos pesquisar uma pessoa pela sua matrícula, como fazemos?



Pesquisar pela Matrícula

```
CELULA *pesquisarMatricula(CELULA **lista, int mat) {
    CELULA *auxiliar;

    if(listaVazia(lista)) {
        printf("ERRO: lista vazia.\n");
        return NULL;
    }

    auxiliar = (*lista); //Recebe o primeiro elemento da lista.
    while(auxiliar != NULL) {
        if(auxiliar->info.matricula == mat)
            return auxiliar;
        //Se não encontrou, continuar procurando.
        auxiliar = auxiliar->next;
    }
    return NULL; //Caso em que não há o elemento procurado.
}
```


Remover pela Matrícula (Parte 1)

```
PESSOA removeMatr(CELULA **lista, int mat){
    CELULA *removida;
    CELULA *anterior;
    PESSOA pRemover = criarPessoa(" ", -1);

    if(listaVazia(lista)){
        printf("\nERRO: Lista Vazia");
        return pRemover;
    }
    //Pesquisa pela matrícula
    removida = pesquisarMatr(lista, mat);
    if(removida == NULL){
        printf("\nERRO: Matricula nao encontrada");
        return pRemover;
    }
    //Caso seja o primeiro elemento
    if(removida == (*lista)){
        return removerInicio(lista);
    }
}
```

Remover pela Matrícula (Parte 2)

```
//Copia o conteudo da célula a ser removida  
//COMO SEI ESSE CONTEÚDO? Pela função pesquisaMatr  
pRemover = removida->info;  
  
anterior = (*lista);  
//Encontra célula/nó anterior ao que será removido  
while(anterior->next != removida){  
    anterior = anterior->next;  
}  
  
//Movimenta o ponteiro. O anterior deve apontar  
//para onde o removida aponta.  
anterior->next = removida->next;  
free(removida);  
  
return pRemover;  
}
```

1. Tendo em mente todas as funções estudadas até o momento, estude o seguinte trecho de código e escreva no caderno qual seriam todas as informações mostradas na tela. Não esqueça de nenhuma impressão (*printf*)!



Exercícios (Parte 1)

```
int main() {
    PESSOA temp;
    CELULA *ptrlista, *tempCel;
    inicializarLista(&ptrlista);

    strcpy(temp.nome, "Jose");
    temp.matricula = 1;

    tempCel = pesquisarMatr(&ptrlista, 2);
    (tempCel != NULL) ? printf("\nEncontrada") : printf("\nNao encontrada");

    inserirFim(&ptrlista, temp);
    imprimirLista(&ptrlista);

    tempCel = pesquisarMatr(&ptrlista, 2);
    (tempCel != NULL) ? printf("\nMat. Encontrada") : printf("\nMat. Nao encontrada");
}
```

Exercícios (Parte 2)

```
strcpy(temp.nome, "Maria");  
temp.matricula = 2;  
  
inserirFim(&ptrlista, temp);  
imprimirLista(&ptrlista);  
  
strcpy(temp.nome, "Joao");  
temp.matricula = 4;  
  
tempCel = pesquisarMatr(&ptrlista, 2);  
(tempCel != NULL) ? printf("Encontrada") : printf("N encontrada");  
  
inserirInicio(&ptrlista, temp);  
imprimirLista(&ptrlista);  
  
temp = removerMatr(&ptrlista, 4);  
imprimirLista(&ptrlista);  
  
liberaLista(&ptrlista);  
imprimirLista(&ptrlista);  
return 0;  
}
```

Exercícios (Respostas)

Acertaram tudo???

```
Lista Vazia.  
Nao encontrada  
Lista:  
Matricula: 1    Nome: Jose  
  
Mat. Nao encontrada  
Lista:  
Matricula: 1    Nome: Jose  
Matricula: 2    Nome: Maria  
Encontrada  
Lista:  
Matricula: 4    Nome: Joao  
Matricula: 1    Nome: Jose  
Matricula: 2    Nome: Maria  
  
Lista:  
Matricula: 1    Nome: Jose  
Matricula: 2    Nome: Maria  
  
Lista Vazia.  
Process returned 0 (0x0)   execution time : 0.273 s  
Press any key to continue.  
_
```

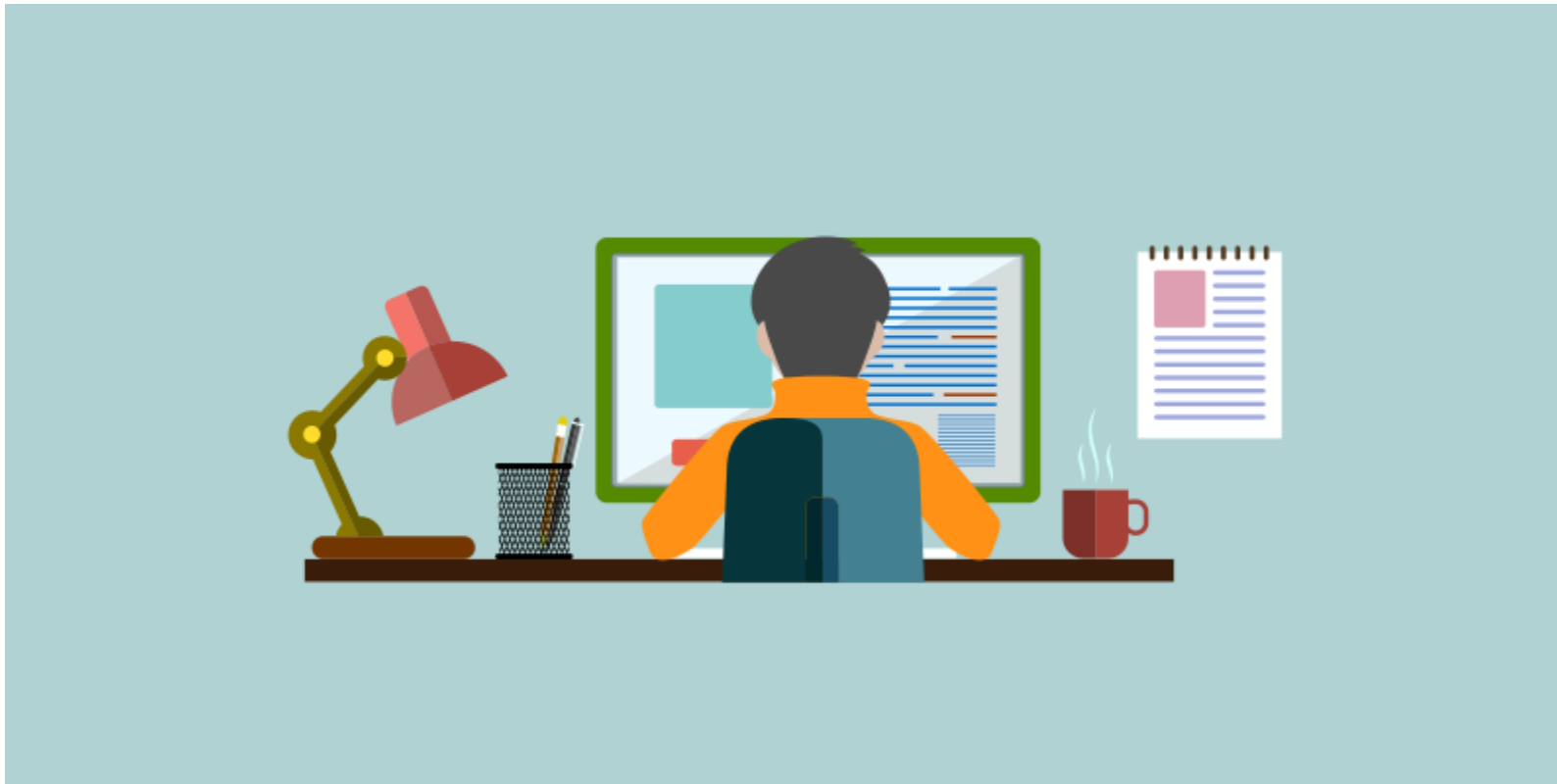
Exercícios (Respostas)

Crie um esquema de MENU que possibilite ao usuário a utilização de todas as funções apresentadas no slide. Assuma que utilizaremos a struct PESSOA.

A entrega deverá ser realizada por meio da tarefa no Teams.

```
int menu(){  
    printf("=====MENU=====\\n");  
    printf("0 - SAIR\\n");  
    printf("1 - VERIFICAR SE VAZIA\\n");  
    printf("2 - INSERIR PESSOA NO FIM\\n");  
    printf("3 - INSERIR PESSOA NO INICIO\\n");  
    printf("4 - IMPRIMIR LISTA DE PESSOAS\\n");  
    printf("5 - REMOVER PESSOA NO INICIO\\n");  
    printf("6 - REMOVER PESSOA NO FIM\\n");  
    printf("7 - PESQUISAR POR MATRICULA\\n");  
    printf("8 - REMOVER POR MATRICULA\\n");  
}
```

Lista Encadeada Ordenada





Estrutura de Dados

Prof. Marques Sousa
marques.sousa@ifsp.edu.br