



# INF 1007 – Programação II

## Aula 11 – Listas Encadeadas

Edirlei Soares de Lima  
<elima@inf.puc-rio.br>

# Introdução

- **Vetores:**

- Ocupa um espaço contíguo de memória;
- Permite acesso randômico;
- Requer pré-dimensionamento de espaço de memória;

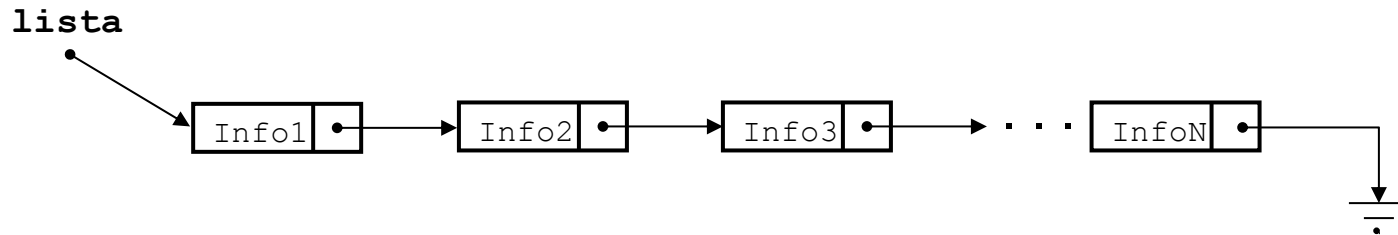


- **Estruturas de Dados Dinâmicas:**

- Crescem (ou decrescem) à medida que elementos são inseridos (ou removidos);
- Exemplo: Listas Encadeadas;
- Outras estruturas: pilhas, filas...

# Listas Encadeadas

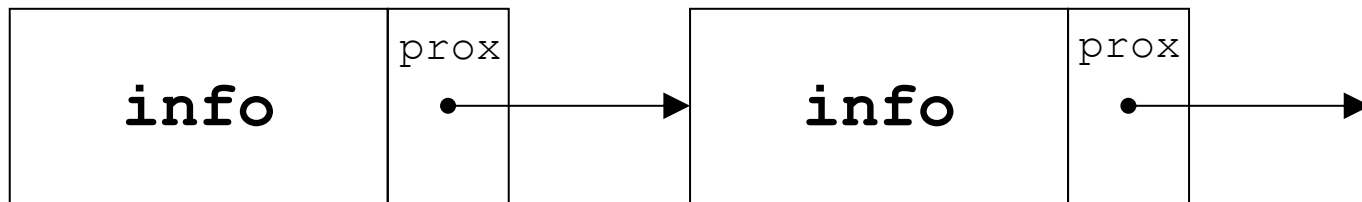
- Uma **Lista Encadeada** é uma sequência de elementos, onde cada elemento tem uma informação armazenada e um ponteiro para o próximo elemento da sequência:



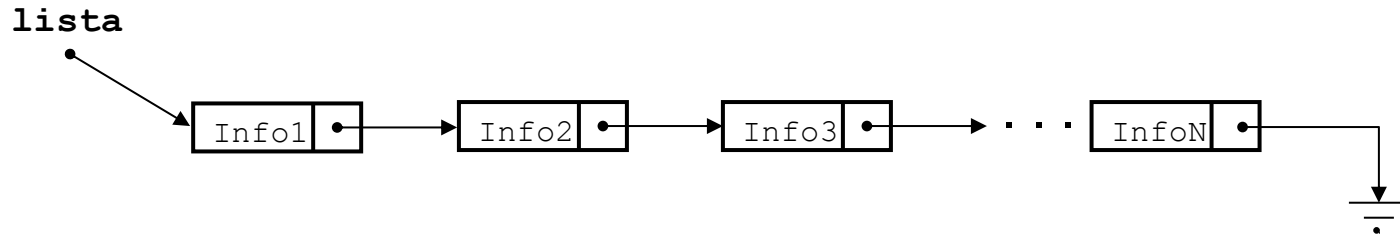
- sequência encadeada de elementos, chamados de nós da lista;
- nó da lista é representado por dois campos:
  - informação armazenada;
  - ponteiro para o próximo elemento da lista;
- a lista é representada por um ponteiro para o primeiro nó;
- o ponteiro do último elemento é NULL;

# Listas Encadeadas

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



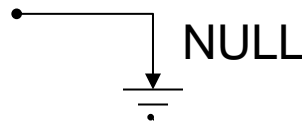
# Listas Encadeadas



- Operações em listas encadeadas:
  - Criação;
  - Inserção;
  - Impressão;
  - Teste de vazia;
  - Busca;
  - Remover um elemento;
  - Libera a lista;
  - Manter lista ordenada;
  - Igualdade;

# Listas Encadeadas – Criação

```
/* função de criação: retorna uma lista vazia */  
Elemento* lista_cria()  
{  
    return NULL;  
}
```

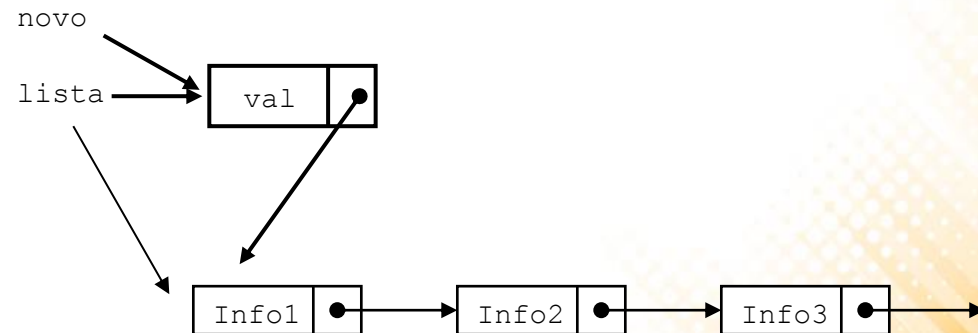


Cria uma lista vazia, representada pelo ponteiro NULL

# Listas Encadeadas – Inserção

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

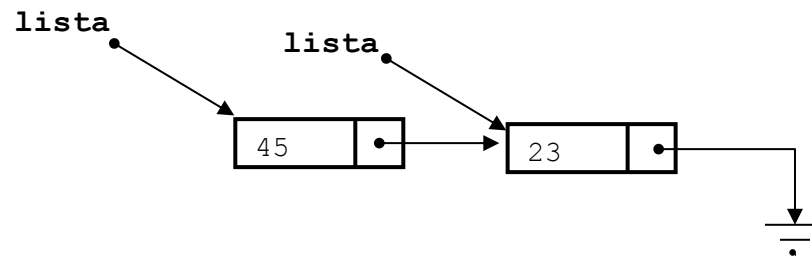
1. Aloca memória para armazenar o elemento;
2. Encadeia o elemento na lista existente;



# Listas Encadeadas – Exemplo

```
int main(void)
{
    Elemento* lista;      /* declara uma lista não inicializada */
    → lista = lista_cria(); /* cria e inicializa lista como vazia */

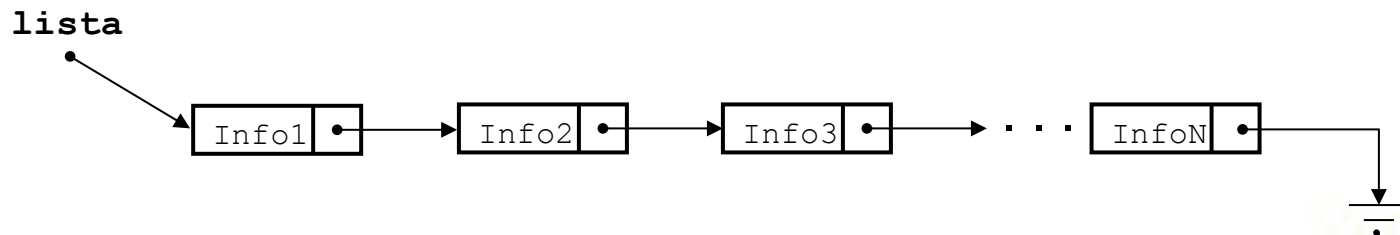
    → lista = lista_insere(lista, 23); /* insere o elemento 23 */
    → lista = lista_insere(lista, 45); /* insere o elemento 45 */
    ...
    return 0;
}
```





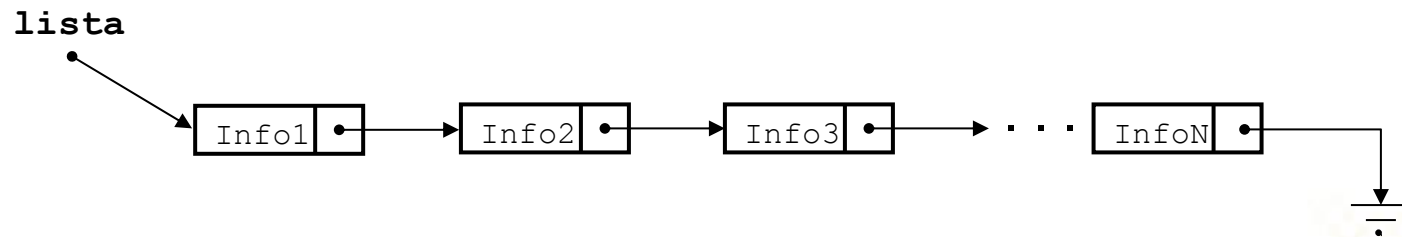
# Listas Encadeadas – Impressão

```
/* função imprime: imprime valores dos elementos */  
void lista_imprime(Elemento* lista)  
{  
    Elemento* p;  
    for (p = lista; p != NULL; p = p->prox)  
        printf("info = %d\n", p->info);  
}
```



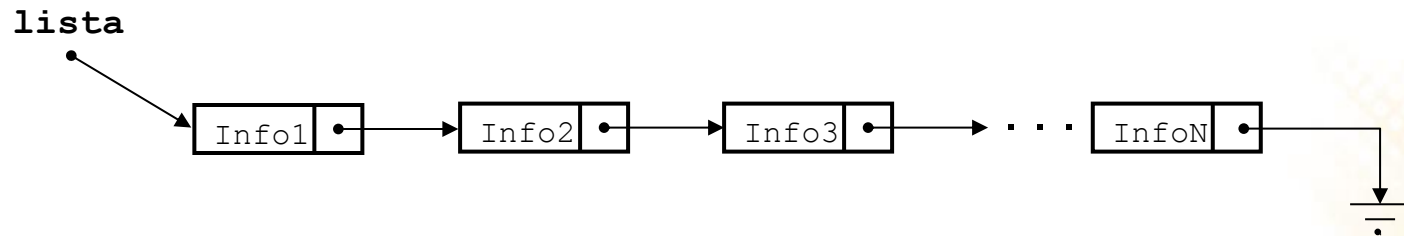
# Listas Encadeadas – Teste Vazia

```
/* função vazia: retorna 1 se vazia ou 0 se não vazia */  
int lista_vazia(Elemento* lista)  
{  
    if (lista == NULL)  
        return 1;  
    else  
        return 0;  
}
```



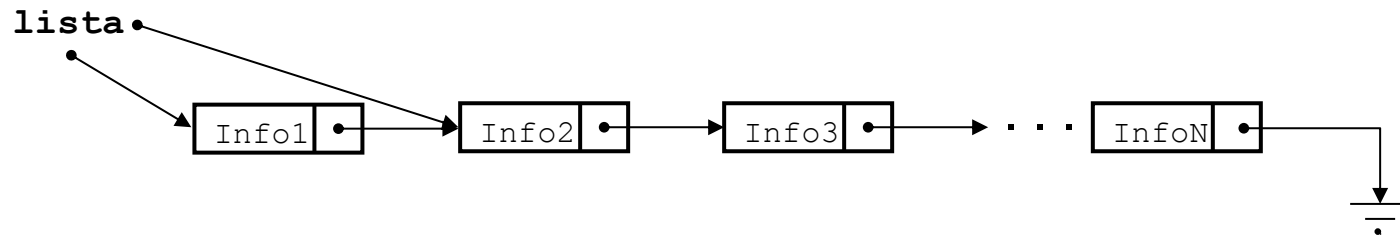
# Listas Encadeadas – Busca

```
/* função busca: busca um elemento na lista */
Elemento* busca (Elemento* lista, int v)
{
    Elemento* p;
    for (p = lista; p != NULL; p = p->prox)
    {
        if (p->info == v)
            return p;      /* achou o elemento */
    }
    return NULL;          /* não achou o elemento */
}
```

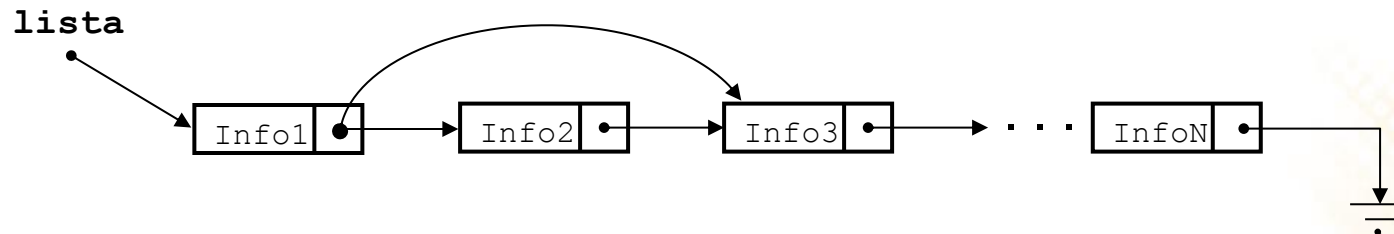


# Listas Encadeadas – Remove

- Recebe como entrada a lista e o valor do elemento a retirar
- Se o elemento a ser removido for o primeiro, atualiza o ponteiro da lista:



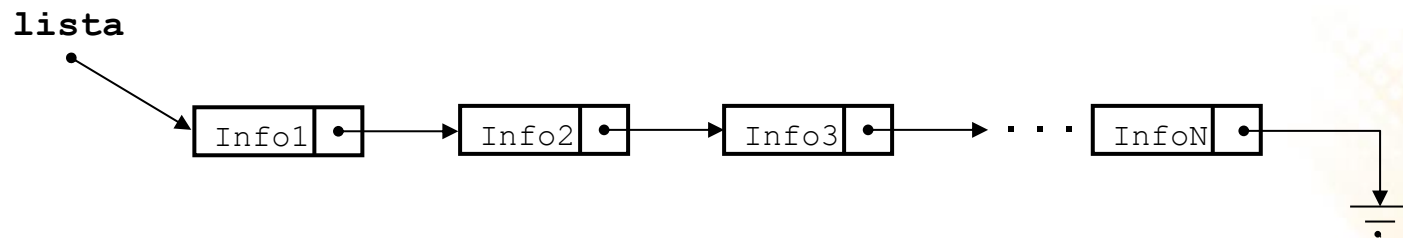
- Caso contrário, apenas remove o elemento da lista:



```
/* função retira: retira elemento da lista */
Elemento* lista_retira(Elemento* lista, int val)
{
    Elemento* a = NULL;    /* ponteiro para elemento anterior */
    Elemento* p = lista;  /* ponteiro para percorrer a lista */
    while ((p != NULL) && (p->info != val))
    {
        a = p;
        p = p->prox;
    }
    if (p == NULL)        /* não achou: retorna lista original */
        return lista;
    if (a == NULL)       /* retira elemento do inicio */
        lista = p->prox;
    else
        a->prox = p->prox; /* retira elemento do meio da lista */
    free(p);
    return lista;
}
```

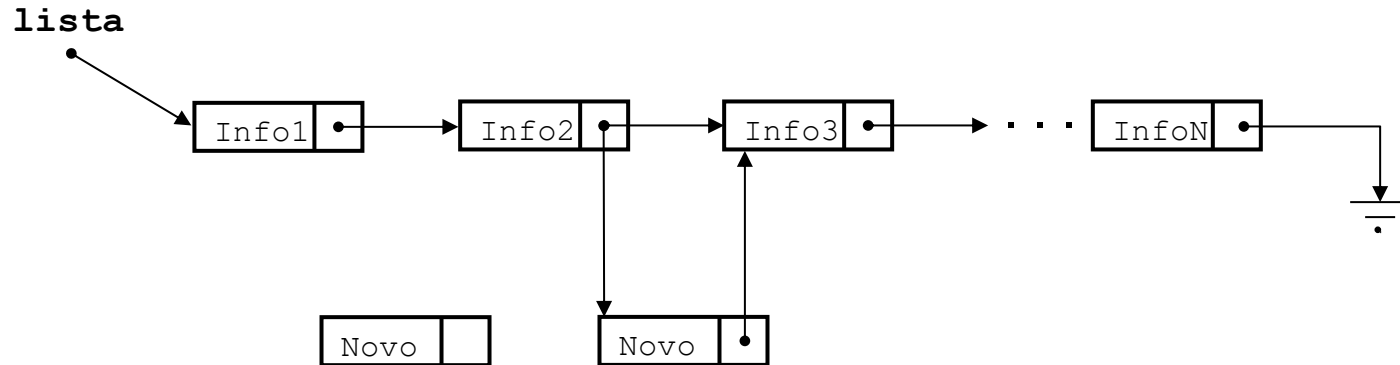
# Listas Encadeadas – Libera Lista

```
/* funcao libera: libera todos os elementos alocados da lista */  
void lista_libera(Elemento* lista)  
{  
    Elemento* p = lista;  
    Elemento* t;  
    while (p != NULL)  
    {  
        t = p->prox;  
        free(p);  
        p = t;  
    }  
}
```



# Listas Encadeadas – Manter Ordenação

- A função de inserção percorre os elementos da lista até encontrar a posição correta para a inserção do novo:



```

/* função insere_ordenado: insere elemento em ordem */
Elemento* lista_insere_ordenado(Elemento* lista, int val){
    Elemento* novo;
    Elemento* a = NULL;      /* ponteiro para elemento anterior */
    Elemento* p = lista;    /* ponteiro para percorrer a lista */
    while ((p != NULL) && (p->info < val)){
        a = p;              /* procura posição de inserção */
        p = p->prox;
    }
    novo = (Elemento*) malloc(sizeof(Elemento));
    novo->info = val;
    if (a == NULL){
        novo->prox = lista;    /* insere no início da lista */
        lista = novo;
    }else {
        novo->prox = a->prox;  /* insere no meio da lista */
        a->prox = novo;
    }
    return lista;
}

```

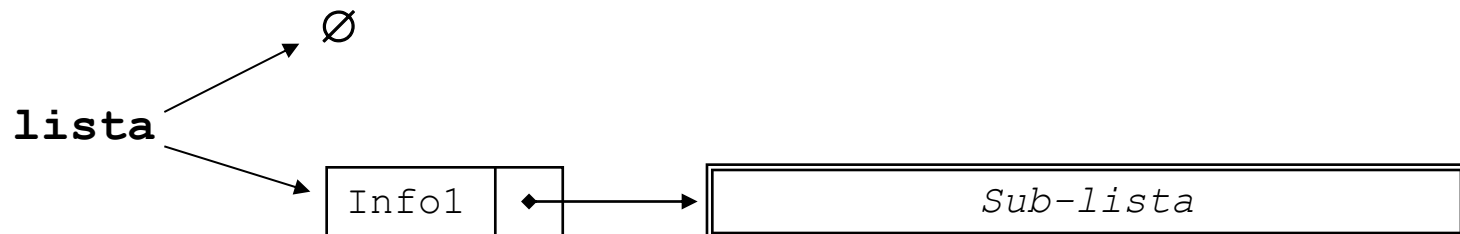


# Listas Encadeadas – Igualdade

```
/* funcao igualdade: compara se duas listas são iguais */
int lista_igual(Elemento* lista1, Elemento* lista2)
{
    Elemento* p1;          /* ponteiro para percorrer lista1 */
    Elemento* p2;          /* ponteiro para percorrer lista2 */
    for (p1=lista1, p2=lista2; p1 != NULL && p2 != NULL;
         p1 = p1->prox, p2 = p2->prox)
    {
        if (p1->info != p2->info)
            return 0;
    }
    if (p1 == p2) /* se ambos forem NULL as listas são iguais */
        return 1;
    else
        return 0;
}
```

# Listas Encadeadas: Definição Recursiva

- Uma lista encadeada é:
  - Uma lista vazia; ou
  - Um elemento seguido de uma (sub-)lista.



# Listas Encadeadas – Impressão Recursiva

- Se a lista for vazia:
  - não imprima nada
- Caso contrário:
  - imprima a informação associada ao primeiro nó, dada por lista->info;
  - imprima a sub-lista, dada por lista->prox, chamando recursivamente a função;

```
/* Função imprime recursiva */  
void lista_imprime_rec(Elemento* lista)  
{  
    if (lista_vazia(lista) == 0) {  
        printf("info: %d\n", lista->info);  
        lista_imprime_rec(lista->prox);  
    }  
}
```

# Listas Encadeadas – Impressão Invertida Recursiva

```
/* Função imprime invertido recursiva */  
void lista_imprime_inv_rec(Elemento* lista)  
{  
    if (lista_vazia(lista) == 0) {  
        lista_imprime_inv_rec(lista->prox);  
        printf("info: %d\n", lista->info);  
    }  
}
```

# Listas Encadeadas – Retira Recursiva

- retire o elemento, se ele for o primeiro da lista (ou da sub-lista);
- caso contrário, chame a função recursivamente para retirar da sub-lista.

```
Elemento* lista_retira_rec(Elemento* lista, int val){
    Elemento* temp;
    if (lista_vazia(lista) == 0){
        if (lista->info == val) {    /* se for o primeiro */
            temp = lista;
            lista = lista->prox;
            free(temp);
        }
        else    /* retira de sub-lista */
            lista->prox = lista_retira_rec(lista->prox, val);
    }
    return lista;
}
```

# Listas Encadeadas – Igualdade Recursiva

- Se as duas listas dadas são vazias, são iguais;
- Se não forem ambas vazias, mas uma delas é vazia, são diferentes;
- Se ambas não forem vazias, teste:
  - Se informações associadas aos primeiros nós são iguais; e
  - Se as sub-listas são iguais .

```
int lista_igual_rec(Elemento* lista1, Elemento* lista2)
{
    if (lista_vazia(lista1)==1 && lista_vazia(lista2)==1)
        return 1;
    else if (lista_vazia(lista1)==1 || lista_vazia(lista2)==1)
        return 0;
    else
        return (lista1->info == lista2->info) &&
            lista_igual_rec(lista1->prox, lista2->prox);
}
```

# Listas de Tipos Estruturados

- A informação associada a cada nó de uma lista encadeada pode ser mais complexa, sem alterar o encadeamento dos elementos;
- As funções apresentadas para manipular listas de inteiros podem ser adaptadas para tratar listas de outros tipos;
- O campo da informação pode ser representado por um ponteiro para uma estrutura, em lugar da estrutura em si independente da informação armazenada na lista, a estrutura do nó é sempre composta por:
  - um ponteiro para a informação; e
  - um ponteiro para o próximo nó da lista.

# Listas de Tipos Estruturados

- Exemplo: Lista Encadeada de Pontos

```
struct ponto{
    float x;
    float y;
};
typedef struct ponto Ponto;

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



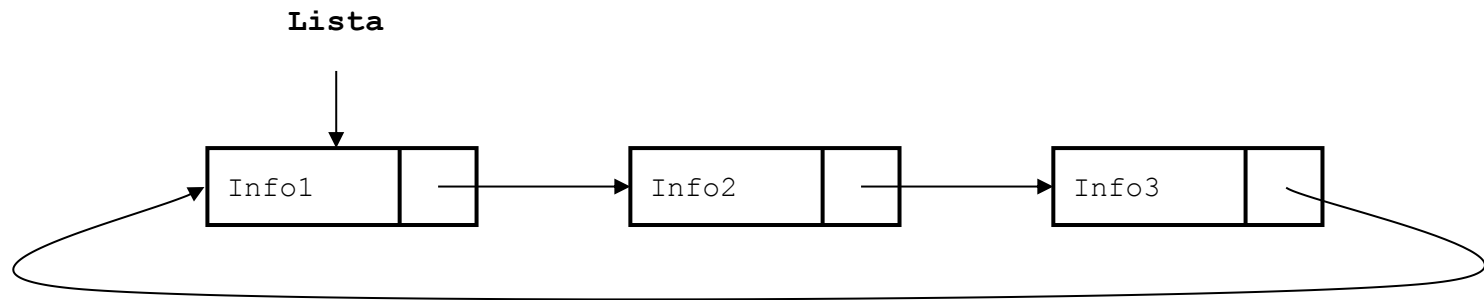
# Listas de Tipos Estruturados

- Exemplo: Inserção em uma lista de pontos

```
/* inserção no início: retorna a lista atualizada */
Elemento* lista_insere(Elemento* lista, float px, float py)
{
    Elemento* p = (Elemento*) malloc(sizeof(Elemento));
    p->info = (Ponto*) malloc(sizeof(Ponto));
    p->info->x = px;
    p->info->y = py;
    p->prox = lista;
    return p;
}
```

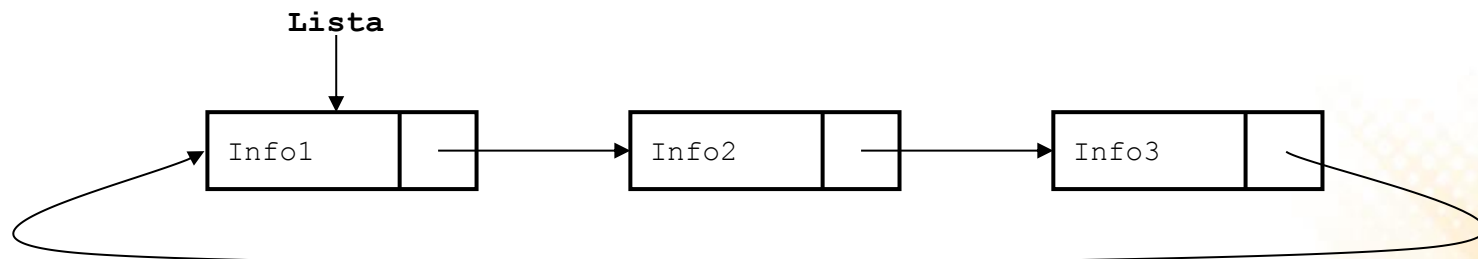
# Listas Circulares

- O último elemento da lista tem como próximo o primeiro elemento da lista, formando um ciclo;
- A lista pode ser representada por um ponteiro para um elemento inicial qualquer da lista;



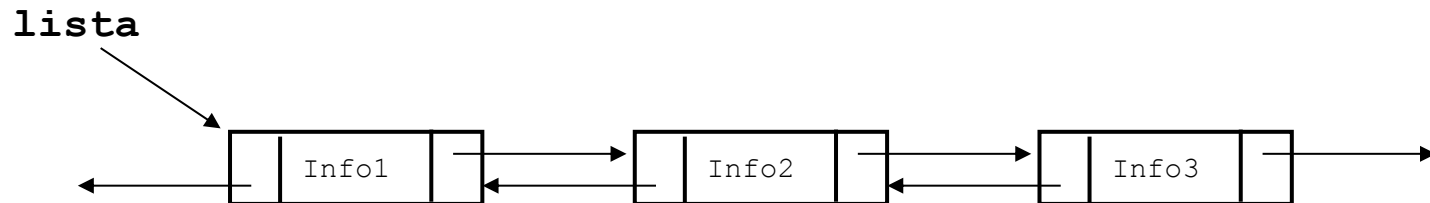
# Listas Circulares – Impressão

```
/* função imprime: imprime valores dos elementos */  
void listacircular_imprime(Elemento* lista)  
{  
    Elemento* p = lista;  
    if (p != NULL){  
        do {  
            printf("%d\n", p->info);  
            p = p->prox;  
        } while (p != lista);  
    }  
}
```



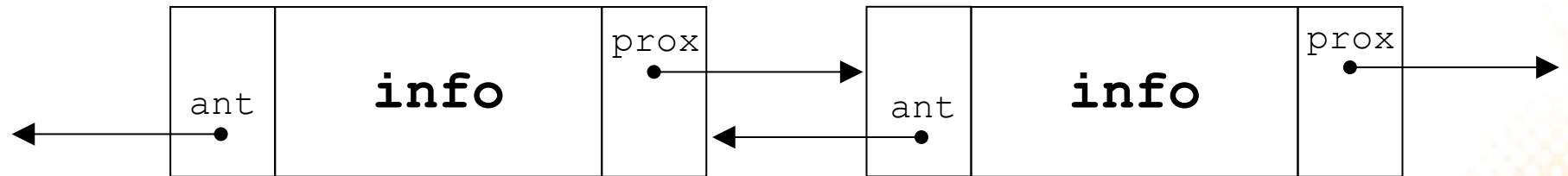
# Listas Duplamente Encadeadas

- Cada elemento tem um ponteiro para o próximo elemento e um ponteiro para o elemento anterior;
- Dado um elemento, é possível acessar o próximo e o anterior;
- Dado um ponteiro para o último elemento da lista, é possível percorrer a lista em ordem inversa ;



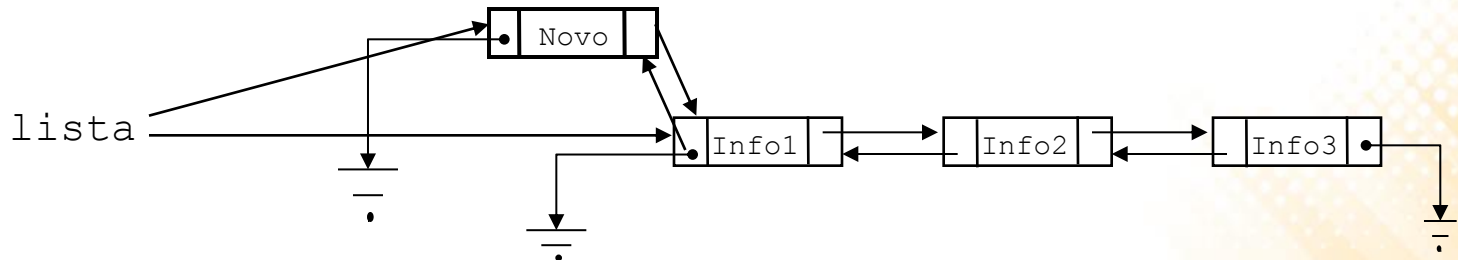
# Listas Duplamente Encadeadas

```
struct lista2 {  
    int info;  
    struct lista2* ant;  
    struct lista2* prox;  
};  
typedef struct lista2 Lista2;
```



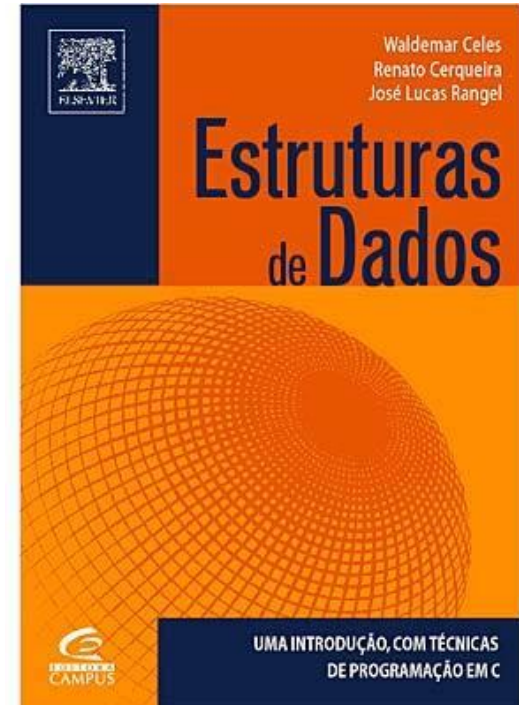
# Listas Duplamente Encadeadas – Inserção

```
/* inserção no início: retorna a lista atualizada */  
Lista2* lista2_inserere(Lista2* lista, int val)  
{  
    Lista2* novo = (Lista2*) malloc(sizeof(Lista2));  
    novo->info = val;  
    novo->prox = lista;  
    novo->ant = NULL;  
    if (lista != NULL)  
        lista->ant = novo;  
    return novo;  
}
```



# Leitura Complementar

- Waldemar Celes, Renato Cerqueira, José Lucas Rangel, **Introdução a Estruturas de Dados**, Editora Campus (2004).
- **Capítulo 10 – Listas Encadeadas**



# Exercícios

## Lista de Exercícios 09 – Listas Encadeadas

<http://www.inf.puc-rio.br/~elima/prog2/>