

# Listas encadeadas

(IED-001)

---

Prof. Dr. Silvio do Lago Pereira

Departamento de Tecnologia da Informação

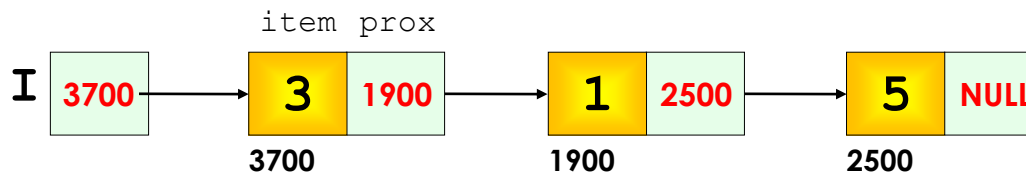
Faculdade de Tecnologia de São Paulo



# Listas

## Lista (encadeada)

é uma sequência de nós compostos por um item e um ponteiro para o próximo nó.



### Observações:

- Cada nó ocupa uma posição arbitrária de memória (a ordem é lógica, não física).
- Operações de inserção, remoção e acesso podem ser feitas em qualquer ponto.
- O endereço do primeiro nó é mantido num ponteiro especial (por exemplo, **I**).
- O último nó da sequência não tem sucessor (seu campo **prox** vale **NULL**).
- Um ponteiro inicial com valor igual a **NULL** representa uma lista vazia.



Lista é útil para guardar uma coleção de itens, cujo tamanho pode variar durante a execução!

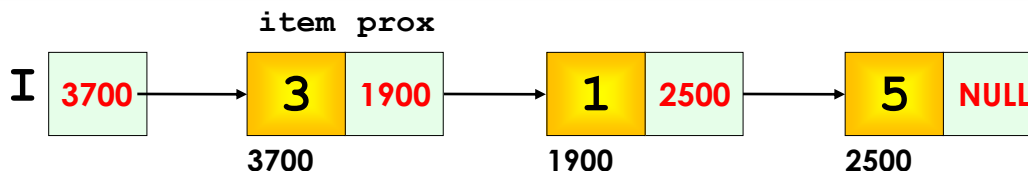


# Listas

## Exemplo 1. O tipo Lista

```
typedef int Item;  
typedef struct no {  
    Item item;  
    struct no *prox;  
} *Lista;
```

**Lista  $\equiv$  struct no \***



### Observações:

- O tipo **Item** indica o tipo dos itens armazenados na lista.
- O tipo **Lista** é usado para declarar um ponteiro de lista (que aponta o primeiro nó).
- Se **I** é um ponteiro de lista, então **I->item** é o primeiro item da lista.
- Se **I** é um ponteiro de lista, então **I->prox** é o endereço do segundo nó da lista (resto).

Note que o tipo **Item** pode ser redefinido, em função da aplicação que usa o tipo **Lista**!

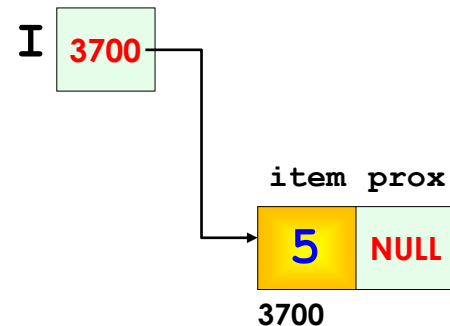


# Listas

## Exemplo 2. Criação de nó

```
➔ Lista no(Item x, Lista p) {  
➔     Lista n = malloc(sizeof(struct no));  
➔     n->item = x;  
➔     n->prox = p;  
➔     return n;  
➔ }
```

➔ Lista I = no(5, NULL);



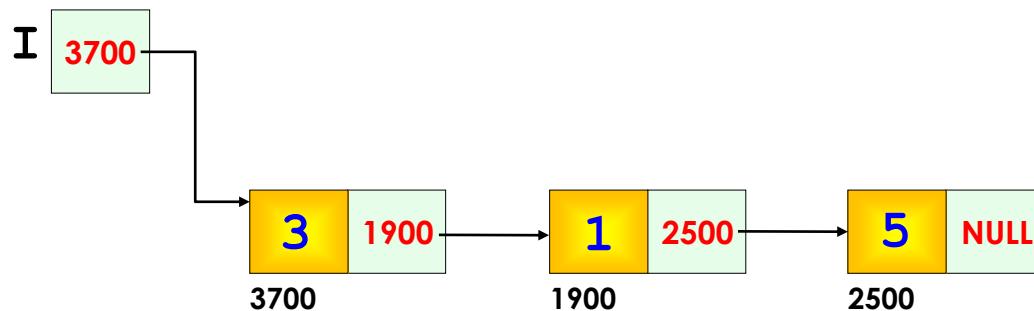
Usando a função **no()** é possível criar qualquer lista desejada!



# Listas

## Exemplo 3. Criação de lista

```
Lista I = no(3, no(1, no(5, NULL))) ;
```



Note que as chamadas de `no()` são executadas da mais interna para a mais externa!

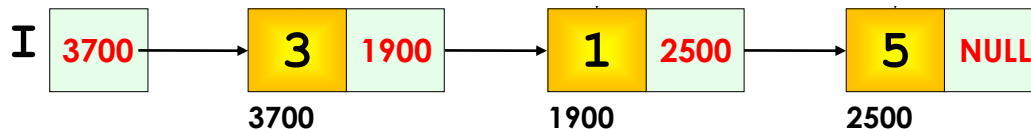


# Listas

## Exemplo 4. Exibição de lista

```
➡ void exibe(Lista L) {  
➡     while( L != NULL ) {  
➡         printf("%d\n", L->item) ;  
➡         L = L->prox;  
➡     }  
➡ }
```

➡ **exibe(I)**



vídeo

3  
1  
5

Note que, no final do percurso, o ponteiro inicial continua apontando o primeiro nó da lista!



# Listas

## Exercício 1. Programa para criação e exibição

Complete e execute o programa a seguir.

```
#include <stdio.h>
#include <stdlib.h>

typedef int Item;
typedef struct no {
    Item item;
    struct no *prox;
} *Lista;
...

int main(void) {
    Lista I = no(3,no(1,no(5,NULL)));
    exibe(I);
    return 0;
}
```



# Listas

## Exercício 2. Outra forma de exibição

Altere a função **exibe()**, de modo que os itens da lista sejam exibidos entre colchetes e separados por vírgulas. A lista vazia deve ser exibida como **[]**.

Por exemplo, após a alteração da função **exibe()**, a execução do código abaixo deve produzir a saída indicada a seguir:

...

```
int main(void) {  
    Lista I = no(3,no(1,no(5,NULL)));  
    exibe(I);  
    return 0;  
}
```

Saída:

**[3,1,5]**



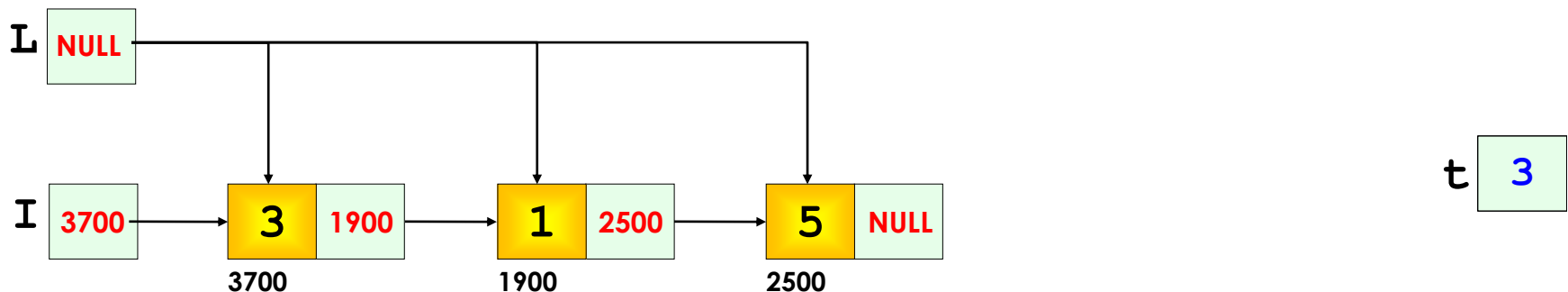


# Listas

## Exemplo 5. Tamanho de lista

```
➡ int tamanho(Lista L) {  
➡     int t = 0;  
➡     while( L ) {  
➡         t++;  
➡         L = L->prox;  
➡     }  
➡     return t;  
➡ }
```

➡ tamanho(I)



Note que, quando **L** tem o valor **NULL**, a condição do comando **while** é falsa!



# Listas

## Exercício 3. Programa para tamanho

Complete e execute o programa a seguir.

```
#include <stdio.h>
#include <stdlib.h>
...
int main(void) {
    Lista I = no(3,no(1,no(5,NULL)));
    exibe(I);
    printf("Tamanho = %d\n", tamanho(I));
    return 0;
}
```

## Exercício 4. Soma dos itens da lista

Adicione no programa do exercício anterior uma função para calcular a soma dos itens da lista. Por exemplo, considerando que **I** aponta a lista de inteiros **[3,1,5]**, a chamada **soma(I)** deve devolver a resposta **9**.

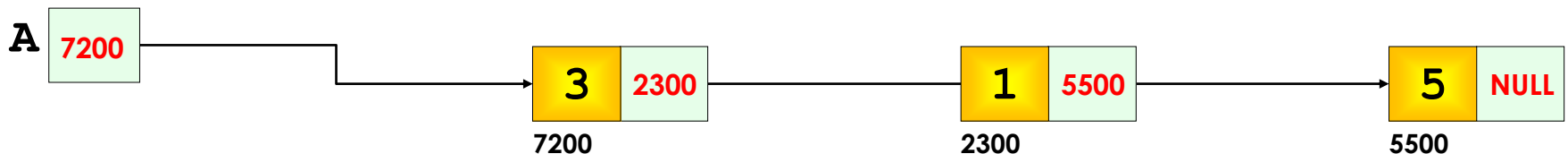


# Listas

**Exemplo 6.** Criação de lista com  $n$  itens, escolhidos aleatoriamente em  $[0, m-1]$

```
➔ Lista aleatoria(int n, int m) {  
➔     Lista L = NULL;  
➔     while( n>0 ) {  
➔         L = no(rand()%m, L);  
➔         n--;  
➔     }  
➔     return L;  
➔ }
```

➔ Lista A = aleatoria(3,10);



No Pelles C, a função **rand()** devolve um número aleatório no intervalo  $[0, 1073741823]$ !



# Listas

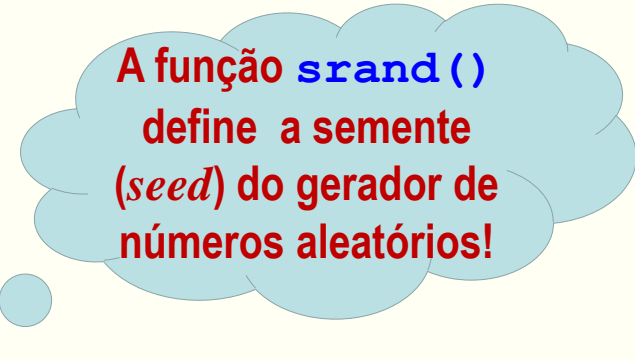
## Exercício 5. Programa para lista aleatória

Complete e execute o programa a seguir.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

...

int main(void) {
    srand(time(NULL));
    Lista A = aleatoria(10,100);
    exibe(A);
    return 0;
}
```



A função **srand()**  
define a semente  
(*seed*) do gerador de  
números aleatórios!

## Exercício 6. Criação de lista contendo um intervalo

Crie a função **intervalo(n)**, que devolve uma lista com os inteiros consecutivos de 1 até n.

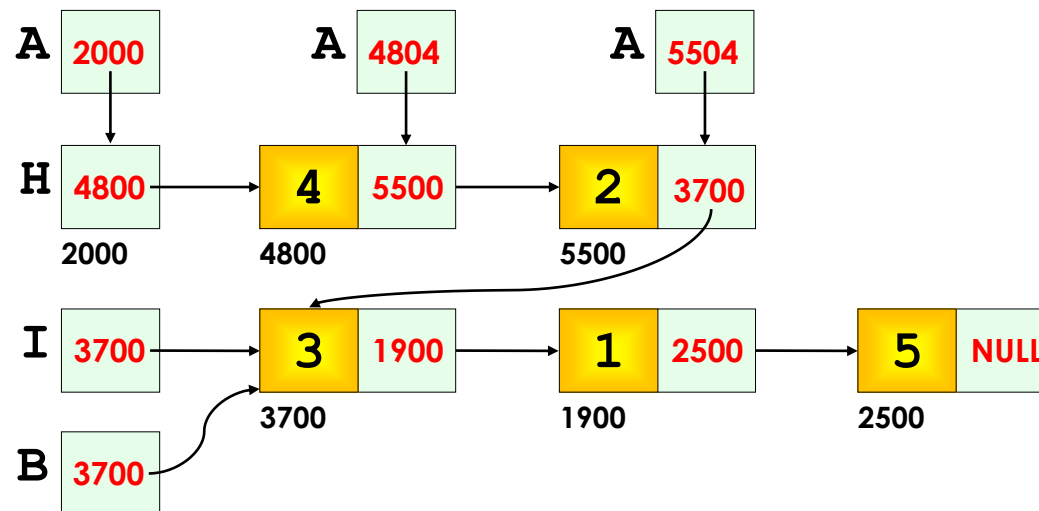


# Listas

## Exemplo 7. Anexação de listas

```
void anexa(Lista *A, Lista B) {  
    if( !B ) return;  
    while( *A )  
        A = &(*A)->prox;  
    *A = B;  
}
```

➡ `anexa(&H, I)`



Note que a lista apontada por `I` não é modificada!



# Listas

## Exercício 7. Programa para anexação

Complete e execute o programa a seguir.

```
#include <stdio.h>
#include <stdlib.h>
...
int main(void) {
    Lista H = no(4,no(2,NULL));
    Lista I = no(3,no(1,no(5,NULL)));
    printf("H = "); exibe(H);
    printf("I = "); exibe(I);
    printf("Pressione enter");
    getchar();
    anexa(&H,I);
    printf("H = "); exibe(H);
    printf("I = "); exibe(I);
    return 0;
}
```

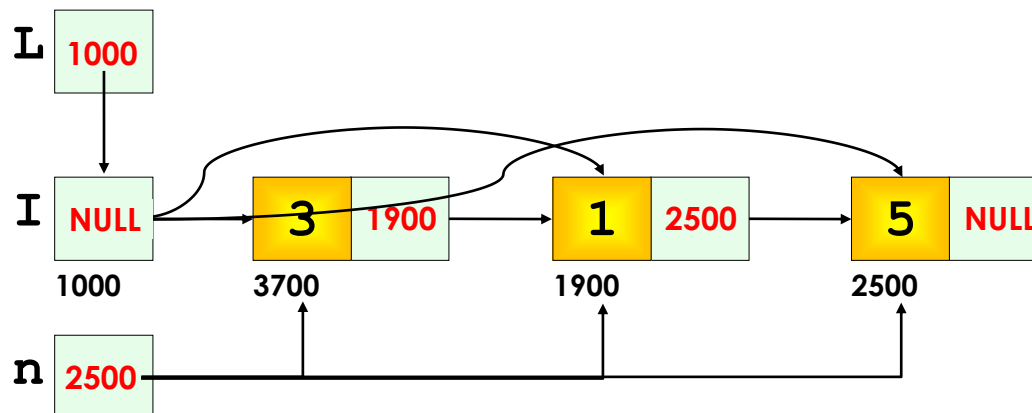


# Listas

## Exemplo 8. Destruição de lista

```
void destroi(Lista *L) {  
    while( *L ) {  
        Lista n = *L;  
        *L = n->prox;  
        free(n);  
    }  
}
```

➡ `destroi(&I)`



Por que a operação de destruição de lista é necessária?



# Listas

## Exercício 8. Último item

Crie a função **ultimo** (**L**) , que devolve o último item da lista **L** (se **L** estiver vazia, *erro fatal*).

## Exercício 9. Item máximo

Crie a função **maximo** (**L**) , que devolve o maior item da lista **L** (se **L** estiver vazia, *erro fatal*).

## Exercício 10. Pertinência

Crie a função **pertence** (**x**, **L**) , que verifica se o item **x** pertence à lista **L**.

## Exercício 11. Inversão

Crie a função **inversa** (**L**) , que cria e devolve uma cópia invertida da lista **L**. Por exemplo, se **L** for a lista **[1, 2, 3]**, a chamada **inversa** (**L**) deve devolver a lista **[3, 2, 1]**.

## Exercício 12. Intervalo

Crie a função **intervalo** (**p**, **u**) , que cria e devolve uma lista contendo inteiros consecutivos de **p** até **u** (para  $p \leq u$ ). Por exemplo, a chamada **intervalo** (**-2, 3**) deve devolver a lista **[-2, -1, 0, 1, 2, 3]**. Se  $p > u$ , a função deve devolver uma lista vazia.



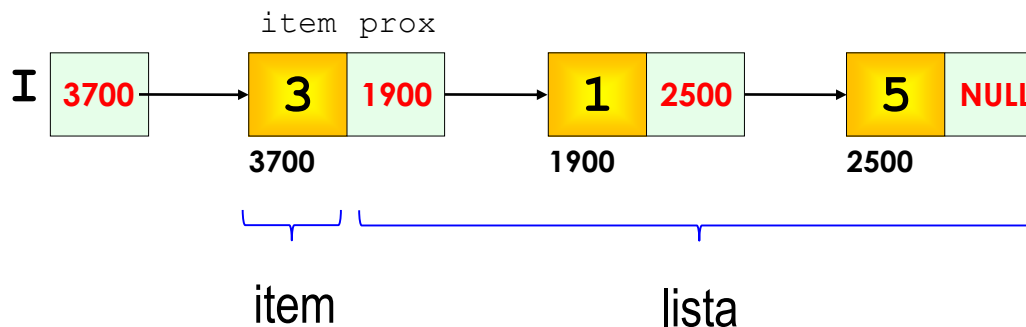


# Listas

## Manipulação recursiva de lista

Listas são naturalmente definidas de forma recursiva. Podemos dizer que uma lista é:

- **vazia**, ou
- um **item** seguido de uma **lista**.



Explorando esse fato, podemos criar funções bem simples e concisas para manipular listas!



# Listas

## Exercício 13. Tamanho

Crie a função recursiva **len**(**L**) , que devolve o tamanho da lista **L**.

## Exercício 14. Soma

Crie a função recursiva **sum**(**L**) , que devolve a soma dos itens da lista **L**.

## Exercício 15. Clone

Crie a função recursiva **clone**(**L**) , que devolve uma cópia da lista **L**.

## Exercício 16. Aleatória

Crie a função recursiva **rnd**(**n**,**m**) , que devolve uma lista com **n** itens aleatórios em  $[0, m-1]$ .

## Exercício 17. Último

Crie a função recursiva **last**(**L**) , que devolve o último item da lista **L**.

## Exercício 18. Pertinência

Crie a função recursiva **in**(**x**,**L**) , que verifica se o item **x** está na lista **L**.



# Listas

## Exercício 19. Enésimo

Crie a função recursiva **nth** ( $n, L$ ) , que devolve o enésimo ( $n \geq 1$ ) item da lista  $L$ .

## Exercício 20. Mínimo

Crie a função recursiva **minimum** ( $L$ ) , que devolve o menor item da lista  $L$ .

## Exercício 21. Ordenada

Crie a função recursiva **sorted** ( $L$ ) , que verifica se a lista  $L$  está ordenada.

## Exercício 22. Igualdade

Crie a função recursiva **equal** ( $A, B$ ) , que verifica se as listas  $A$  e  $B$  são iguais.

## Exercício 23. Contagem

Crie a função recursiva **count** ( $x, L$ ) , que informa quantas vezes o item  $x$  ocorre na lista  $L$ .

## Exercício 24. Substituição

Crie a função recursiva **replace** ( $x, y, L$ ) , que substitui ocorrências de  $x$  por  $y$  na lista  $L$ .

Fim

