# Estruturas lineares dinâmicas (Listas)

Prof. Paulo Henrique Pisani

http://professor.ufabc.edu.br/~paulo.pisani/

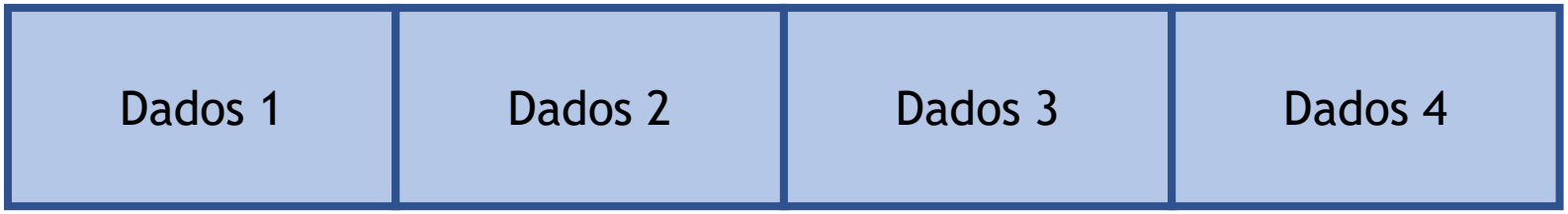
dezembro/2018

# Agenda

- Listas com arranjos;
- Listas ligadas/encadeadas:
  - Listas simplesmente ligadas;
  - Listas duplamente ligadas;
  - Outros tipos: Listas com nó cabeça e Listas circulares.
- Exercícios.

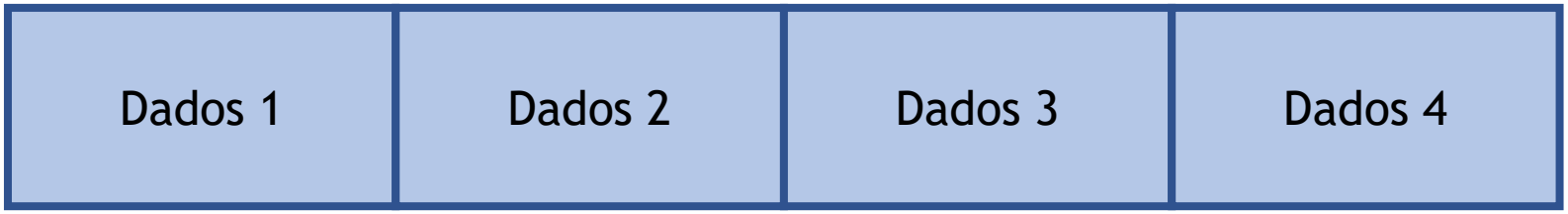
# Listas com arranjos (revisão)

- Itens dispostos em um arranjo sequencial;



# Listas com arranjos (revisão)

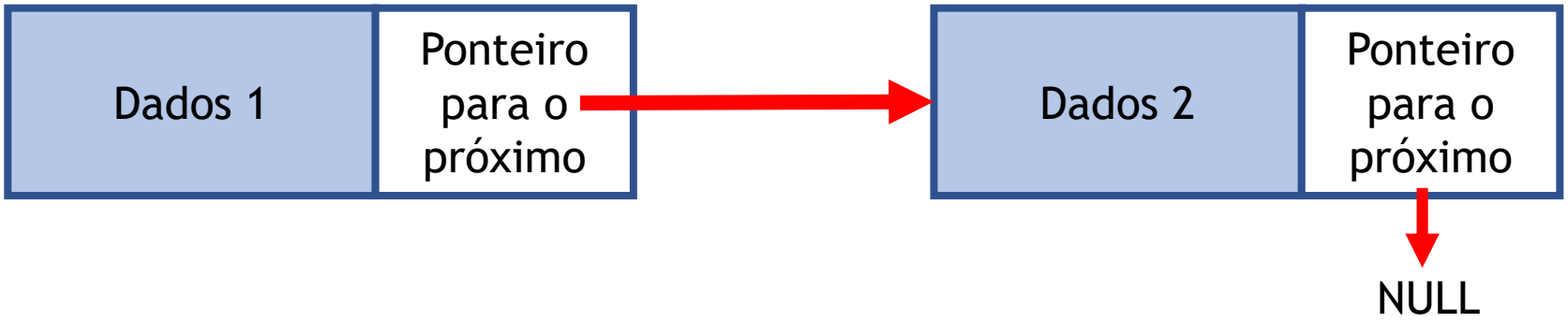
- Itens dispostos em um arranjo sequencial;




Problemas?

# Listas ligadas/encadeadas

- Estrutura de dados que armazena os itens de forma não consecutiva na memória:
  - Usa ponteiros para “ligar” um item no próximo.



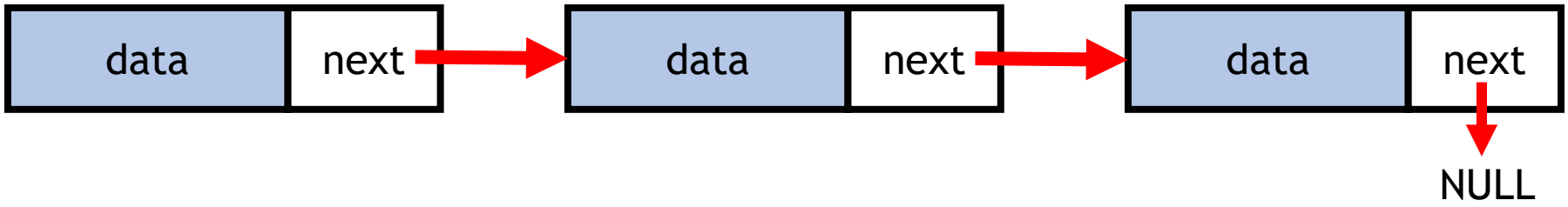
# Listas ligadas/encadeadas

- Vários tipos:
  - Listas simplesmente ligadas (com e sem nó cabeça);
  - Listas duplamente ligadas (com e sem nó cabeça);
  - Listas circulares.

# Listas simplesmente ligadas

# Listas simplesmente ligadas

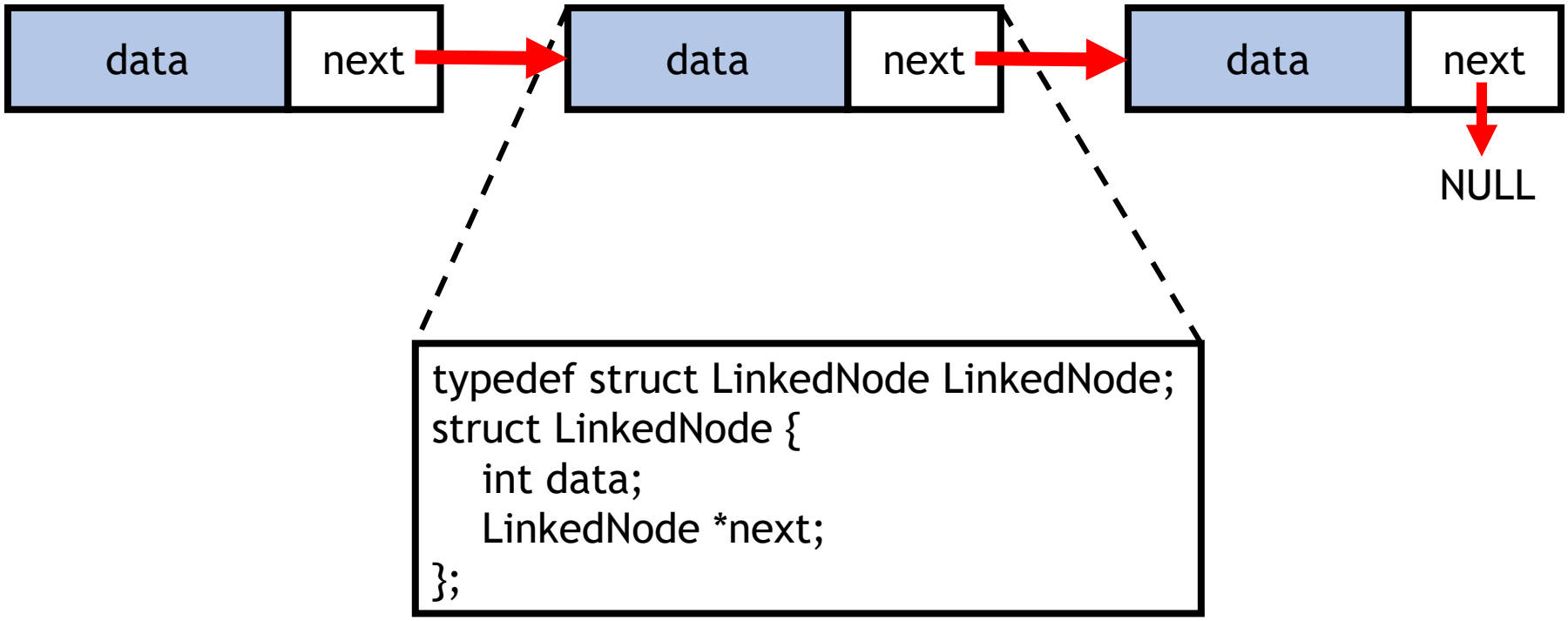
- Cada item é ligado somente ao próximo item;



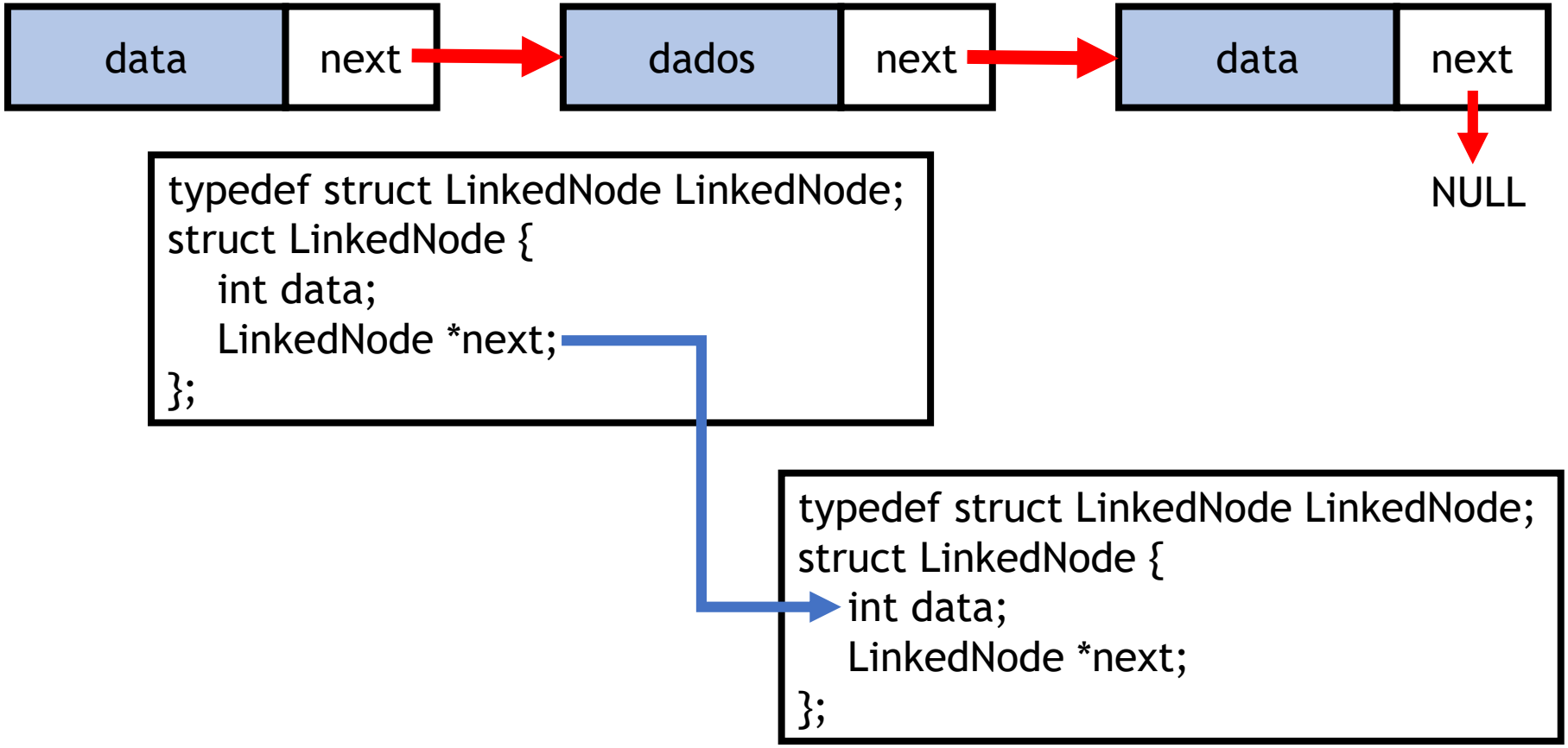

## Como implementar no C?

# Listas simplesmente ligadas

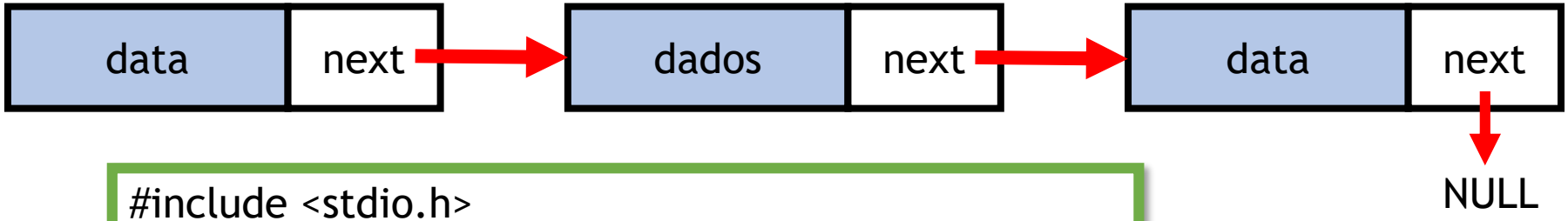
- Cada item é ligado somente ao próximo item;



# Listas simplesmente ligadas



# Listas simplesmente ligadas




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

typedef struct LinkedNode LinkedNode;
struct LinkedNode {
    int data;
    LinkedNode *next;
};

Int main() {
      LinkedNode *first = malloc(sizeof(LinkedNode));

      …
}
```

O ponteiro para o primeiro item deve ser salvo.

# Listas simplesmente ligadas

```
typedef struct LinkedNode LinkedNode;
struct LinkedNode {
    int data;
    LinkedNode *next;
};
```

```
linked_node *append_node(LinkedNode *inicio, int valor) {
        LinkedNode *novo = malloc(sizeof(LinkedNode));
        if (novo == NULL)
                return inicio;
        novo->data = valor;
        novo->next = NULL;

        LinkedNode *anterior = NULL;
        LinkedNode *atual = inicio;
        while (atual != NULL) {
                anterior = atual;
                atual = atual->next;
        }

        if (anterior != NULL)
                anterior->next = novo;
        else
                inicio = novo;
        return inicio;
}
```

# Listas simplesmente ligadas

```
typedef struct LinkedNode LinkedNode;
struct LinkedNode {
    int data;
    LinkedNode *next;
};
```

```
LinkedNode* appendNode(LinkedNode *last, int num) {
    LinkedNode *tmp = malloc(sizeof(LinkedNode));
    if (tmp == NULL)
        return NULL;

    tmp->data = num;
    tmp->next = NULL;

    if (last != NULL) {
        last->next = tmp;
    }
    return tmp;
}
```

# Listas simplesmente ligadas

```
typedef struct LinkedNode LinkedNode;
struct LinkedNode {
    int data;
    LinkedNode *next;
};
```

**E se usar essa função em um item no meio da lista?**

**Como evitar esse problema?**

```
LinkedNode* appendNode(LinkedNode *last, int num) {
    LinkedNode *tmp = malloc(sizeof(LinkedNode));
    if (tmp == NULL)
        return NULL;

    tmp->data = num;
    tmp->next = NULL;

    if (last != NULL) {
        last->next = tmp;
    }
    return tmp;
}
```

# Listas simplesmente ligadas

```
typedef struct LinkedNode LinkedNode;
struct LinkedNode {
    int data;
    LinkedNode *next
};
```

```
LinkedNode* insertNode(LinkedNode *current, int num) {
    LinkedNode *tmp = malloc(sizeof(LinkedNode));
    if (tmp == NULL)
        return NULL;

    tmp->data = num;

    if (current == NULL) {
        tmp->next = NULL;
    } else {
        tmp->next = current->next;
        current->next = tmp;
    }
    return tmp;
}
```

# Listas simplesmente ligadas

```
typedef struct LinkedNode LinkedNode;
struct LinkedNode {
    int data;
    LinkedNode *next;
};
```

```
void printList(LinkedNode *first) {
    LinkedNode *curr = first;
    while (curr != NULL) {
        printf("%d ", curr->data);
        curr = (*curr).next;
    }
    printf("\n");
}

void printListRecursive(LinkedNode *curr) {
    if (curr != NULL) {
        printf("%d ", curr->data);
        printListRecursive(curr->next);
    }
}
```

# Listas simplesmente ligadas

```
typedef struct LinkedNode LinkedNode;
struct LinkedNode {
    int data;
    LinkedNode *next;
};
```

**Lembrar de guardar o ponteiro do segundo item antes de remover!**

```
void removeFirstNode(LinkedNode *first) {
    free(first);
}
```

```
void removeNextNode(LinkedNode *current) {
    LinkedNode *nextnext = current->next->next;

    free(current->next);

    current->next = nextnext;
}
```

# Exercício 1 (a)

- Escrever um programa que fique lendo números em uma lista ligada até que o usuário digite o número -1;
- Depois imprima a lista de números.

# Exercício 1 (b)

- Adapte o exercício anterior de forma que os elementos sejam inseridos em ordem crescente na lista ligada.

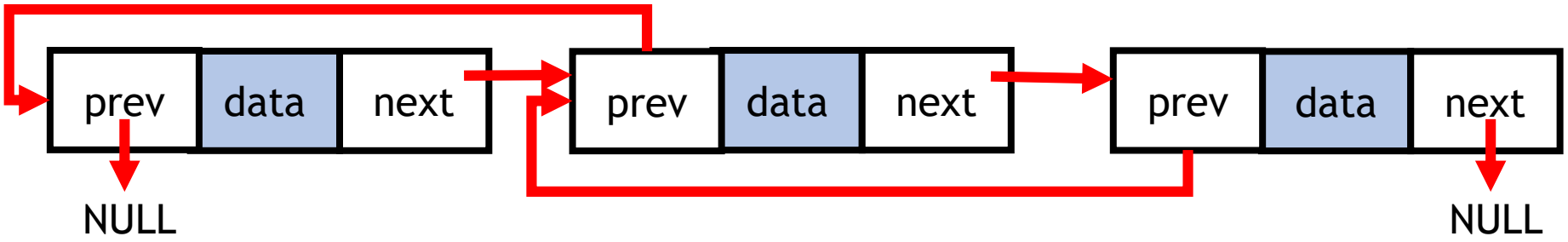
# Exercício 2

- Adaptar exercício AC6 (supermercado) para utiilzar lista ligada. Nesse caso, o programa não precisa ter um limite de itens pré-definido.

# Listas duplamente ligadas

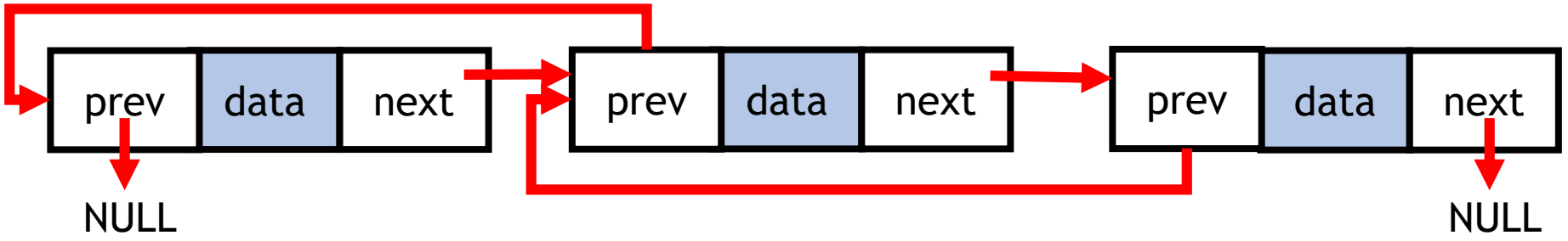
# Listas duplamente ligadas

- Cada item é ligado ao próximo item e também ao anterior;
- **Vantagem**: **a lista pode ser percorrida em ambas as direções.**



# Listas duplamente ligadas

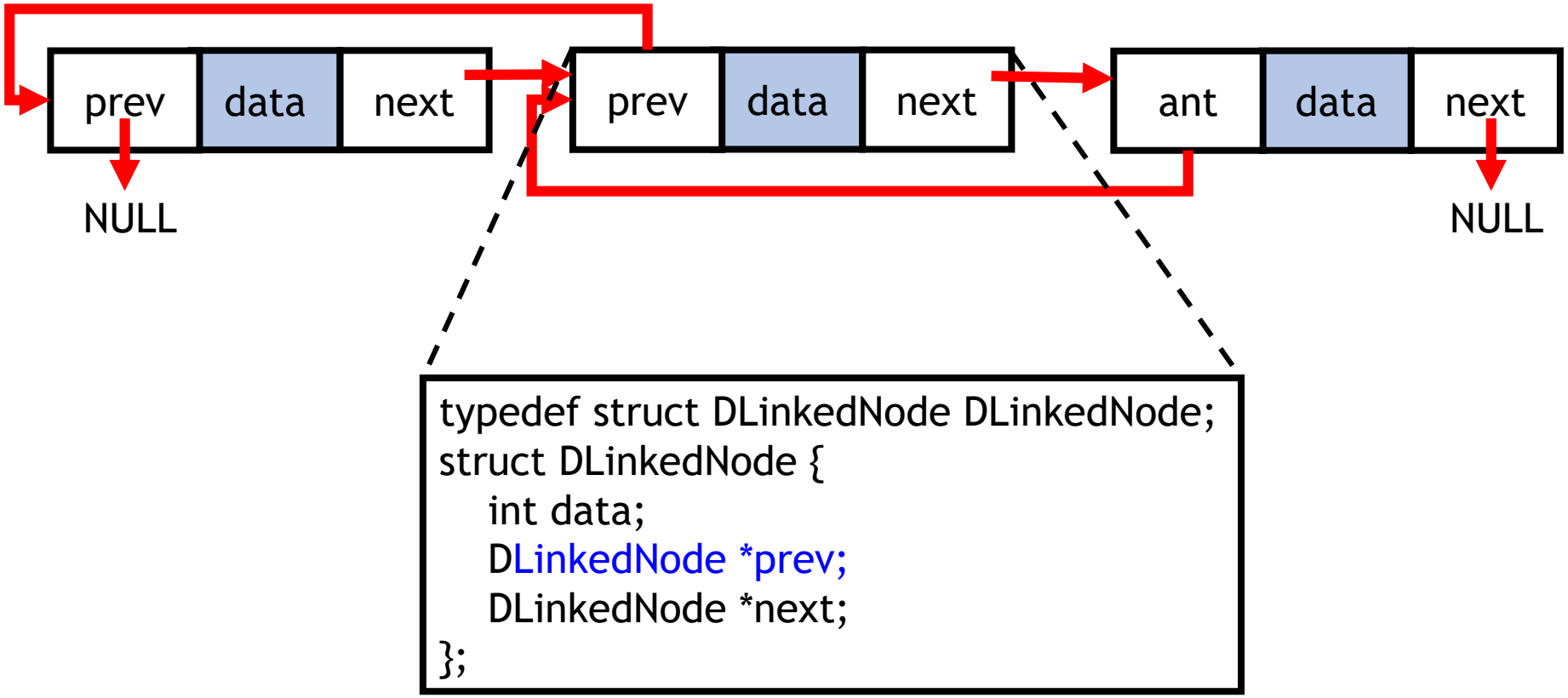
- Cada item é ligado ao próximo item e também ao anterior;



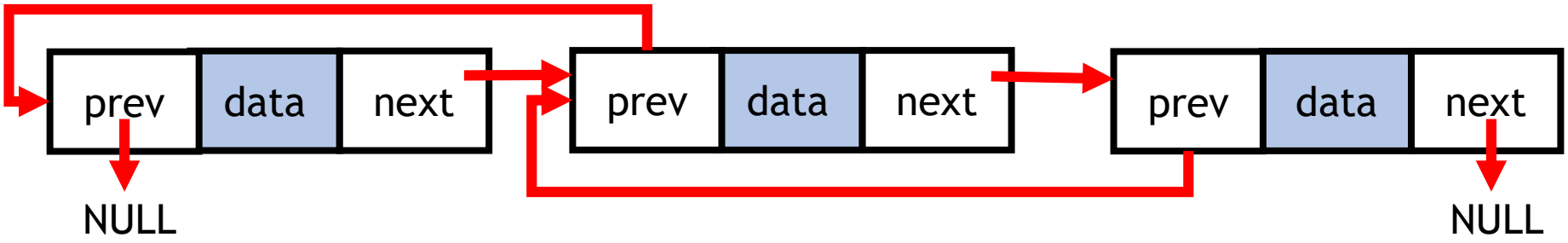

Como implementar no C?

# Listas duplamente ligadas

- Cada item é ligado ao próximo item e também ao anterior;



# Listas duplamente ligadas




```
typedef struct DLinkedNode DLinkedNode;
struct DLinkedNode {
    int data;
    DLinkedNode *prev;
    DLinkedNode *next;
};
```

```
typedef struct DLinkedNode DLinkedNode;
struct DLinkedNode {
    int data;
    DLinkedNode *prev;
    DLinkedNode *next;
};
```

# Listas duplamente ligadas

```
typedef struct DLinkedNode DLinkedNode;
struct DLinkedNode {
    int data;
    DLinkedNode *prev;
    DLinkedNode *next;
};
```

```
DLinkedNode* appendNodeD(DLinkedNode *last, int num) {
    DLinkedNode *tmp = malloc(sizeof(DLinkedNode));
    if (tmp == NULL)
        return NULL;

    tmp->data = num;
    tmp->prev = last;
    tmp->next = NULL;

    if (last != NULL) {
        last->next = tmp;
    }
    return tmp;
}
```

# Listas duplamente ligadas

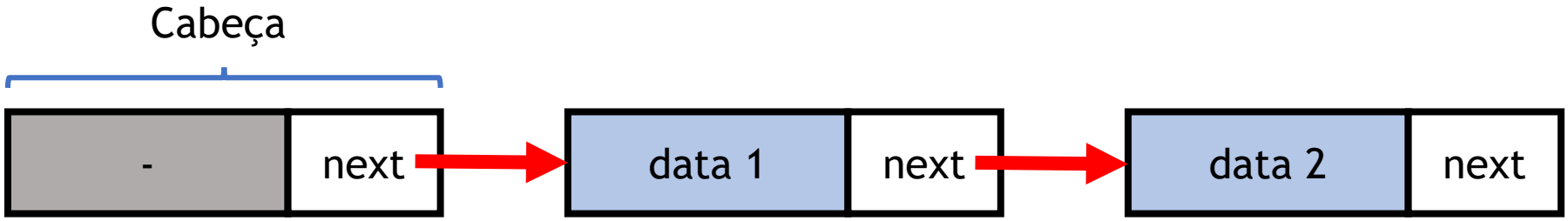
```
DLinkedNode* insertNodeD(DLinkedNode *current, int num) {
    DLinkedNode *tmp = malloc(sizeof(DLinkedNode));
    if (tmp == NULL)
        return NULL;

    tmp->data = num;
    tmp->prev = current;
    if (current == NULL) {
        tmp->next = NULL;
        return tmp;
    } else {
        tmp->next = current->next;
        if (current->next != NULL) {
            current->next->prev = tmp;
        }
        current->next = tmp;
    }
    return tmp;
}
```

# Outros tipos

Com nó cabeça e lista circular

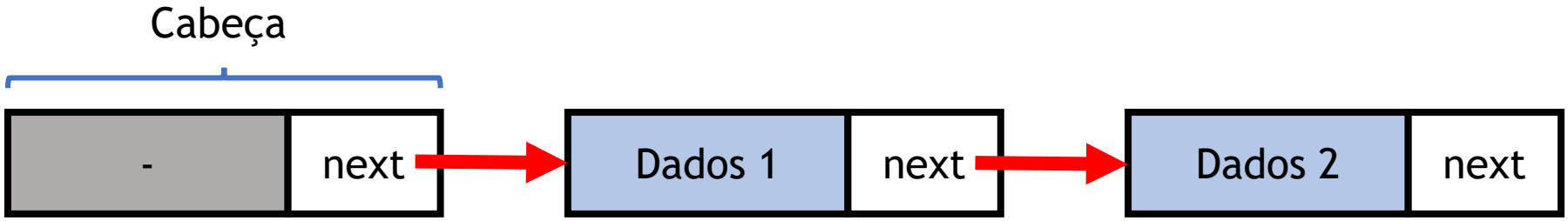
# Listas simplesmente ligadas **com nó cabeça**

- Cada item é ligado somente ao próximo item;
- O primeiro item não armazena dados da lista (e nunca é excluído);
- **Vantagem**: não é necessário verificar se a lista está vazia (o item cabeça nunca é removido).



# Listas simplesmente ligadas **com nó cabeça**

- Cada item é ligado somente ao próximo item;
- O primeiro item não armazena dados da lista (e nunca é excluído).




Como implementar no C?

# Listas simplesmente ligadas **com nó cabeça**

```
typedef struct LinkedNode LinkedNode
struct LinkedNode {
    int data;
    LinkedNode *next;
};
```

```
LinkedNode* appendNode(LinkedNode *last, int num) {
    LinkedNode *tmp = malloc(sizeof(LinkedNode));
    if (tmp == NULL)
        return NULL;

    tmp->data = num;
    tmp->next = NULL;

    last->next = tmp;

    return tmp;
}
```

# Listas simplesmente ligadas **com nó cabeça**

```
typedef struct LinkedNode LinkedNode
struct LinkedNode {
    int data;
    LinkedNode *next;
};
```

```
LinkedNode* insertNode(LinkedNode *current, int num) {
    LinkedNode *tmp = malloc(sizeof(LinkedNode));
    if (tmp == NULL)
        return NULL;

    tmp->data = num;

    tmp->next = current->next;
    current->next = tmp;

    return tmp;
}
```

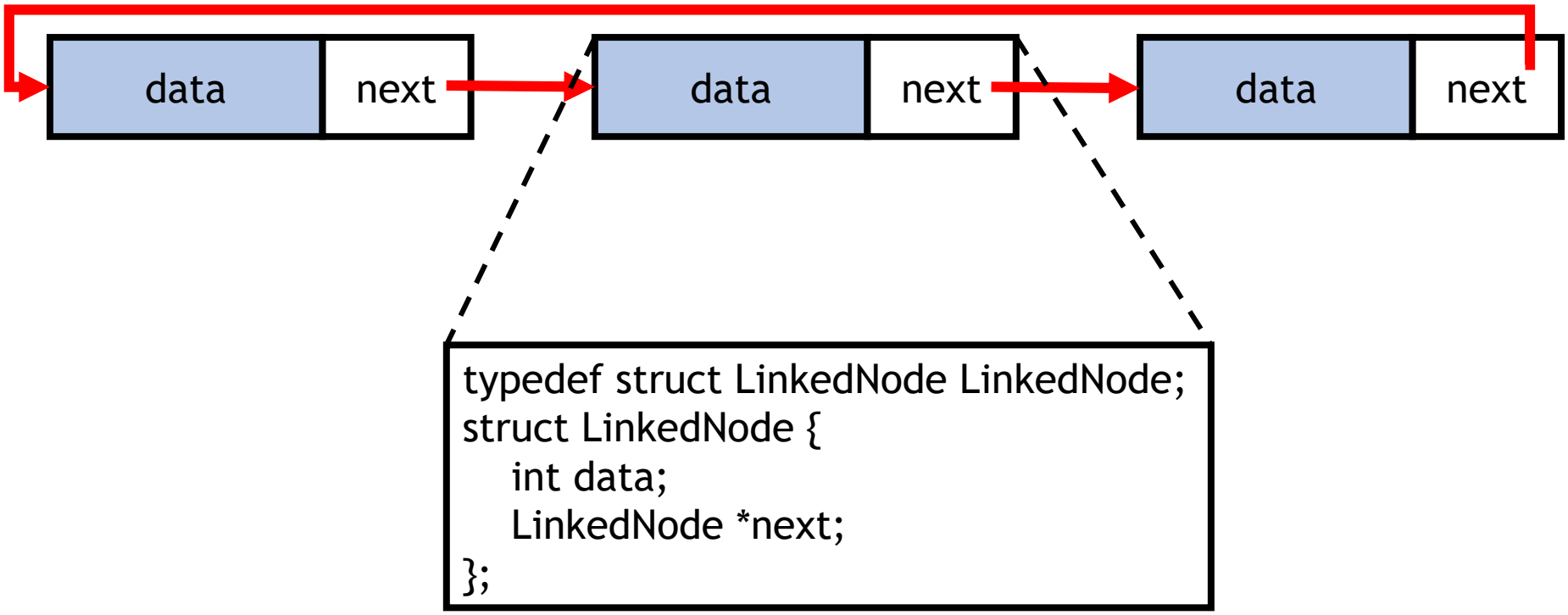
# Listas circulares

- Cada item é ligado somente ao próximo item e **o último item é ligado ao primeiro**.

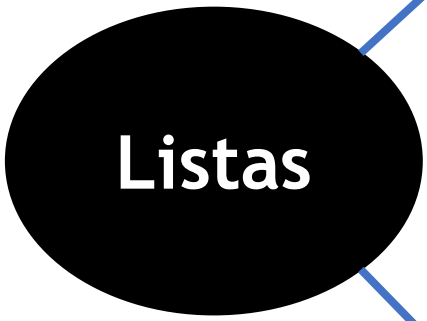

## Como implementar no C?

# Listas circulares

- Cada item é ligado somente ao próximo item e **o último item é ligado ao primeiro**.



# Resumo




**Listas com arranjos**

- Simples para usar
- Alocação em bloco contínuo
- Acesso a um item em tempo constante

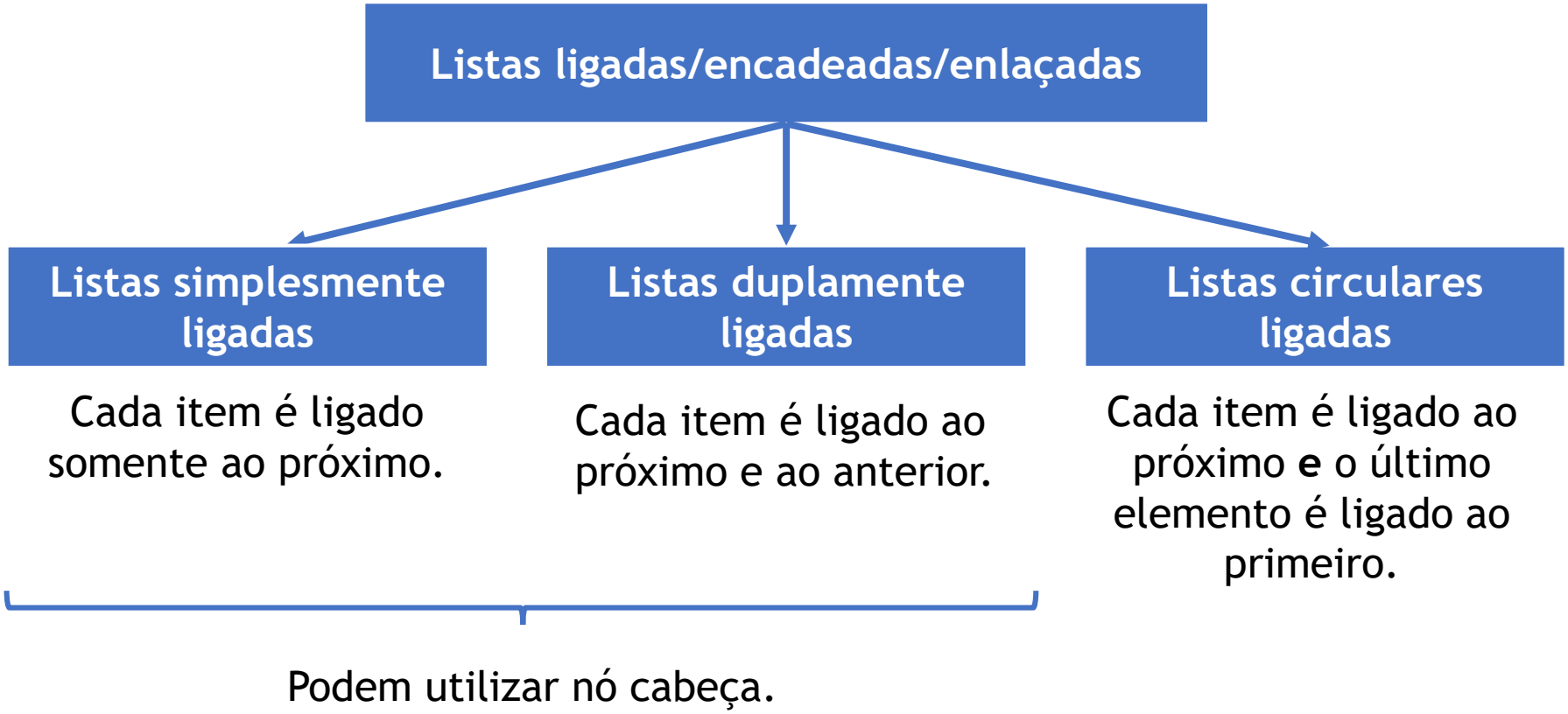
- Requer saber a quantidade de itens previamente (para alocaçao)
- Inserção/Remoção requer deslocamentos
- Expansão custosa (realocar e copiar)

**Listas ligadas/encadeadas/enlaçadas**

- Não requer conhecer a quantidade de itens previamente
- Inserção e remoção não requer deslocamentos

- Acesso a uma posição necessita percorrer a lista
- Memória extra para os ponteiros

# Resumo


Listas ligadas/encadeadas/enlaçadas
Listas simplesmente ligadas
Listas duplamente ligadas
Listas circulares ligadas
Cada item é ligado somente ao próximo.
Cada item é ligado ao próximo e ao anterior.
Cada item é ligado ao próximo **e** o último elemento é ligado ao primeiro.
Podem utilizar nó cabeça.

# Exercício extra

- Adaptar exercício AC6 (supermercado) para utiilzar lista ligada. Nesse caso, o programa não precisa ter um limite de itens pré-definido;
- Adapte o algoritmo de ordenação para trabalhar com listas ligadas também.

# Exercícios

- Escreva funções em C para realizar as seguintes operações com listas simplesmente ligadas:
  1. Concatenar duas listas;
  2. Inverter uma lista sobre ela mesma (sem criar uma nova);
  3. Dividir uma lista em duas metades. Se o tamanho da lista é ímpar, a segunda metade terá tamanho ímpar;
  4. Eliminar o primeiro item de uma lista;
  5. Eliminar o último item de uma lista;
  6. Inserir um item na posição *i* da lista;
  7. Remover o item da posição *i* da lista.

# Referências


- Slides de Algoritmos e Estruturas de Dados I (UFABC): Paulo H. Pisani, Mirtha L. Venero. 2018.
- Nivio Ziviani. Projeto de Algoritmos: com implementações em Pascal e C. Cengage Learning, 2015.
- Robert Sedgewick. Algorithms in C/C++/Java, Parts 1-4 (Fundamental Algorithms, Data Structures, Sorting, Searching). Addison-Wesley Professional.
- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. Algoritmos: Teoria e Prática. Elsevier, 2012.