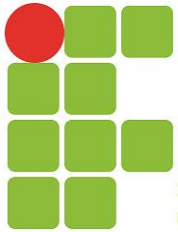


INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Campus Presidente Epitácio

# Estruturas de Dados – ED1C3

## Lista Circular e Lista Duplamente Encadeada

PROF. MARCELO ROBERTO ZORZAN

DISCIPLINA: ESTRUTURAS DE DADOS I

AULA 10

# Aula de Hoje

Lista Circular

Lista Duplamente Encadeada

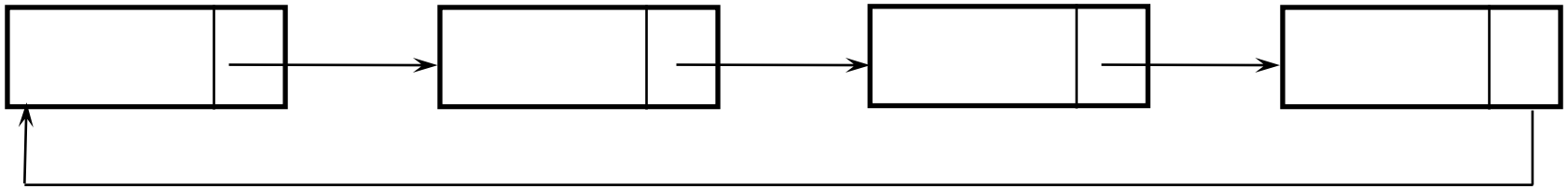
# Situação-Problema

- Suponha que você precise desenvolver uma aplicação em que diversos processos de um sistema operacional utilizam concorrentemente um recurso (exemplo: processador). Além disso, é preciso garantir que nenhum processo acesse o recurso antes de todos os outros o utilizarem.

→ Descreva uma solução para o cenário descrito anteriormente usando o conceito de lista simplesmente encadeada.

# Lista Encadeada Circular

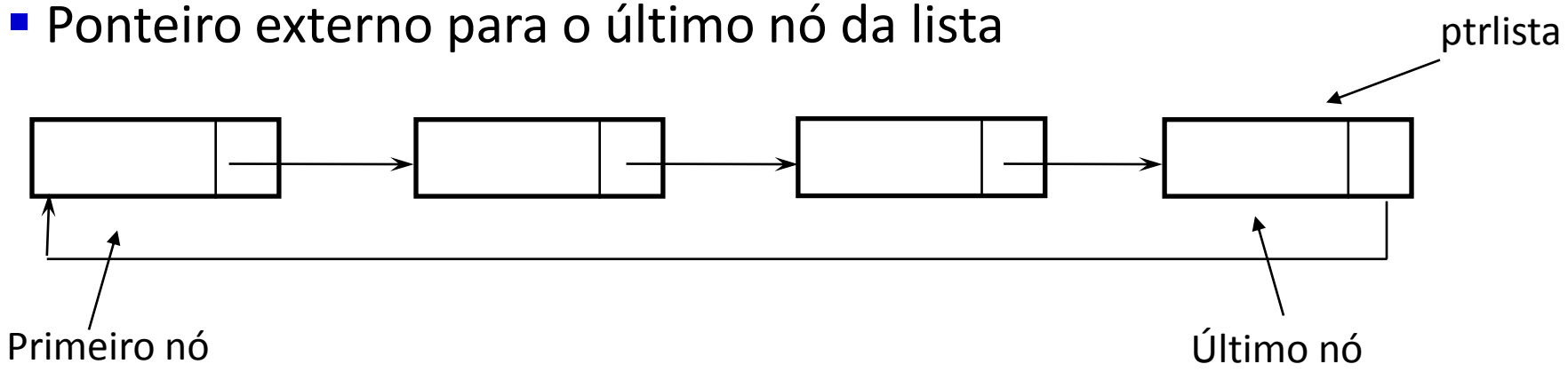
- Em uma lista circular os nós formam um anel.



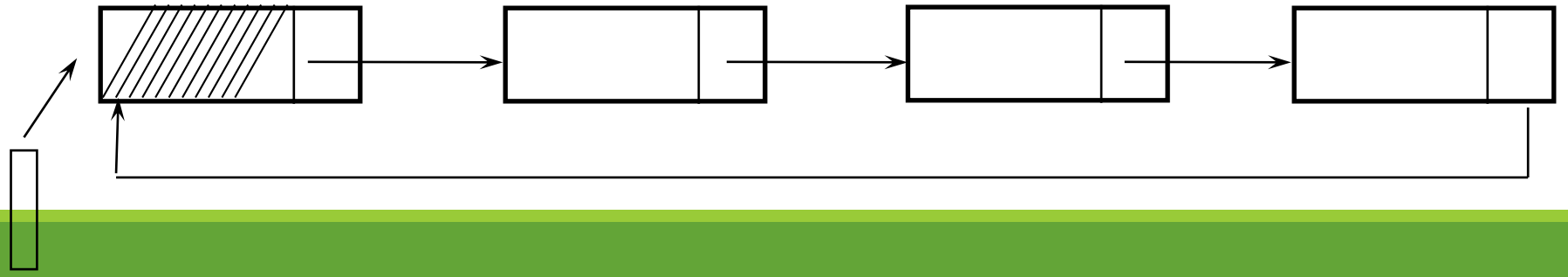
- O último elemento da lista tem como próximo elemento o primeiro elemento desta lista, formando um ciclo.

# Lista Encadeada Circular Possíveis implementações

- Ponteiro externo para o último nó da lista

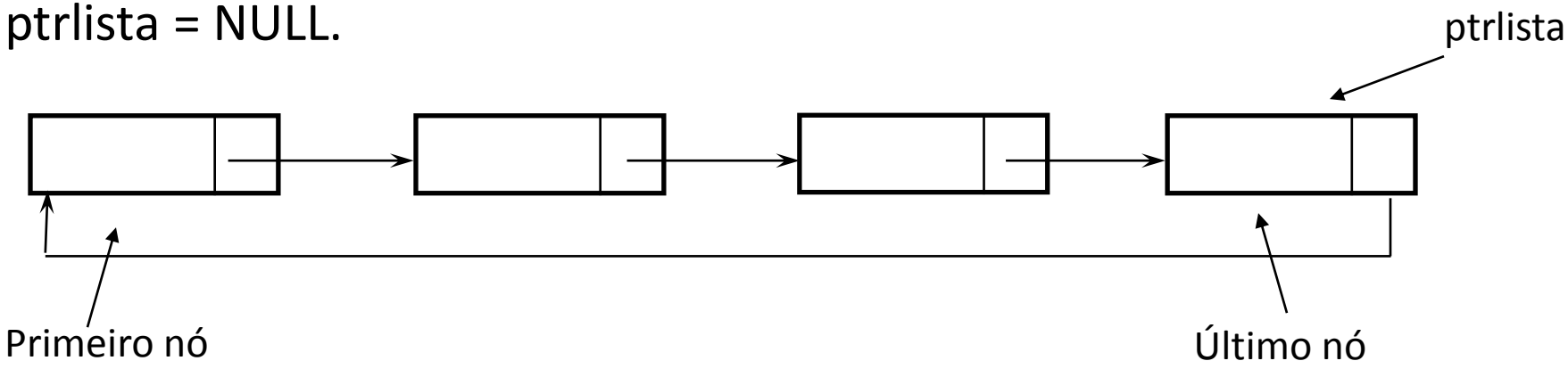



- Nó cabeçalho



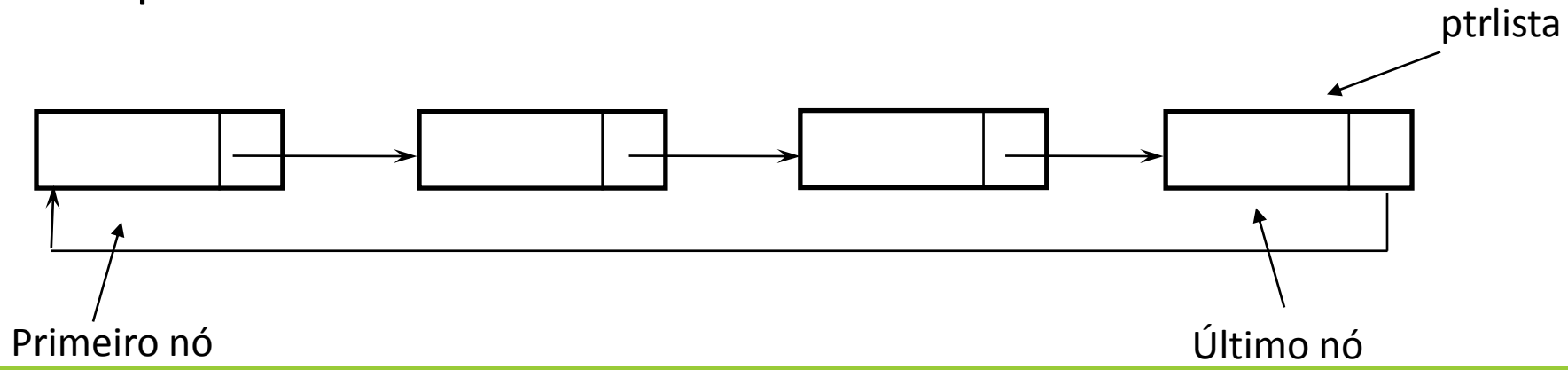
# Lista Encadeada Circular

- Observe que a lista circular não tem o “primeiro” ou o “ultimo” nó natural. Sendo assim, é preciso estabelecê-los por convenção.
- A convenção mais útil é permitir que o ponteiro externo para a lista circular aponte para o último nó, e que o nó seguinte se torne o primeiro nó:
- A lista vazia é representada pelo ponteiro externo nulo, ou seja, ptrlista = NULL.



# Lista Encadeada Circular

- Se ptrlista é um ponteiro externo para uma lista circular, ptrlista referencia o último nó e ptrlista->next referencia o primeiro nó da lista.
- **Vantagem desta convenção:**
  - Poder incluir ou remover um elemento convenientemente a partir do início ou final de uma lista.



# Lista Encadeada Circular - Implementação

- A estrutura de dados lista encadeada circular é definida da mesma forma que uma lista encadeada simples.

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

# Lista Encadeada Circular - Implementação

- Inserir elemento na lista circular

```
void insere_fim(CELULA **lista, int x) {
    CELULA *q;

    q = getnode ();
    if (q != NULL) {
        q->info = x;

        if (empty(*lista)){
            q->next = q;
        }
```

# Lista Encadeada Circular - Implementação

- Inserir elemento na lista circular (cont.)

```
        else{  //insere no fim da lista
           q->next = (*lista)->next;
           (*lista)->next = q;
      }
      *lista = q;
   }// Fim do if(q != NULL)
   else {
      printf ("\nERRO na alocação do nó.\n");
      exit(1);
    }
}
```

# Lista Encadeada Circular - Implementação

```
void listar (CELULA *lista) {
    CELULA *aux;

    aux = lista->next;

    if(aux != NULL) {
        do{
          printf ("%d\t", aux->info);
          aux = aux->next;
        }while(aux != lista->next);
    }
    else
    {
        printf("\nNao ha elemento na lista.");
    }
    printf("\n");
}
```

Exibir elementos da lista circular

# Lista Encadeada Circular - Implementação

- Remover elementos da lista circular

```
void remove_inicio (CELULA **lista) {
    CELULA *aux;

    if (!empty(*lista)) {  //há itens na lista
        if ((*lista) == (*lista)->next) {
                freenode(*lista);
                *lista = NULL;
        }
```

# Lista Encadeada Circular - Implementação

- Remover elementos da lista circular (cont.)

```
        else {
                aux = (*lista)->next;
                (*lista)->next = aux->next;
                freenode(aux);
        }
    }
    else {
        printf ("\nERRO: lista vazia.\n");
        exit(1);
    }
}
```

# Lista Encadeada Circular - Implementação

- Função main

```
int main()
{
    CELULA *ptrlista;
    init(&ptrlista);

    insere_fim(&ptrlista, 9);
    insere_fim(&ptrlista, 1);
    insere_fim(&ptrlista, 3);

    listar(ptrlista);
    remove_inicio (&ptrlista);
    listar(ptrlista);
    remove_inicio (&ptrlista);
    listar(ptrlista);
}
```

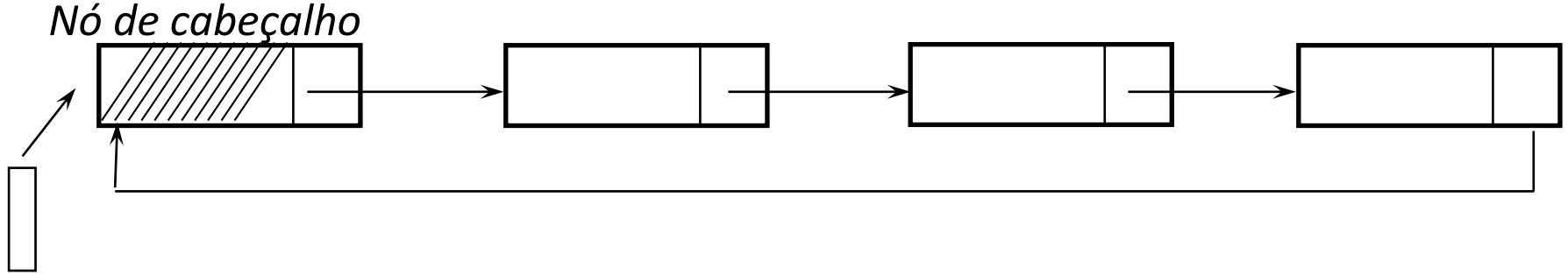
# Nó de cabeçalho

- Suponha que precisemos percorrer uma lista circular

- Executar *aux = aux → next* várias vezes

- Como a lista é circular, não saberemos quando a lista inteira foi percorrida, a não ser que façamos o teste: if(aux == lista->next).

- Solução:
  - Posicionar o nó de cabeçalho como primeiro elemento da lista circular
  - Campo *info* do nó de cabeçalho deverá ser inválido para o contexto do problema, ou poderá conter um sinal que o marque como nó cabeçalho



# Exercício

1) Crie um arquivo ListaSimplesmentEncadeadaCircular.c e implementa as seguintes informações de uma lista circular:

- ✓ Definição
- ✓ Operações
  - init
  - getnode
  - freenode
  - insere_inicio
  - listar
  - remove_inicio

→ Teste seu programa criando um menu com as opções de inserir, remover e exibir.

# Exercício

2) Considerando que cada elemento de uma lista circular é formado por um processo do sistema operacional *Windows* (nome e número do processo) defina um lista simplesmente encadeada circular assumindo que a mesma será implementada usando a representação com nó cabeçalho. O nó cabeçalho deverá conter o número total de processos que estão sendo executados. Em seguida desenvolva uma aplicação que permita inserir e exibir os dados de cada processo.

# Listas encadeadas

Classificação:

- Listas simplesmente encadeadas
- Listas duplamente encadeadas

# Situação-Problema

- É comum no processo inicial de alfabetização de crianças canhotas elas escreverem na folha do caderno começando pelo lado direito em direção ao esquerdo. Esse processo, quando acompanhado pelos pais e professores, é corrigido logo no início desse processo. Considerando que Joãozinho é canhoto e se encontra nessa fase, apresente uma solução usando o conceito de lista para imprimir o nome completo de Joãozinho escrito de forma invertida no caderno.

# Revisando listas simplesmente encadeadas

Um nó em uma lista simplesmente encadeada possui basicamente dois itens:

- informação
- ponteiro para o próximo nó

Limitações:

- Não permite percorrer a lista na ordem inversa
- O processo de remoção exige a presença de um ponteiro auxiliar para acessar o nó anterior ao que desejamos remover.

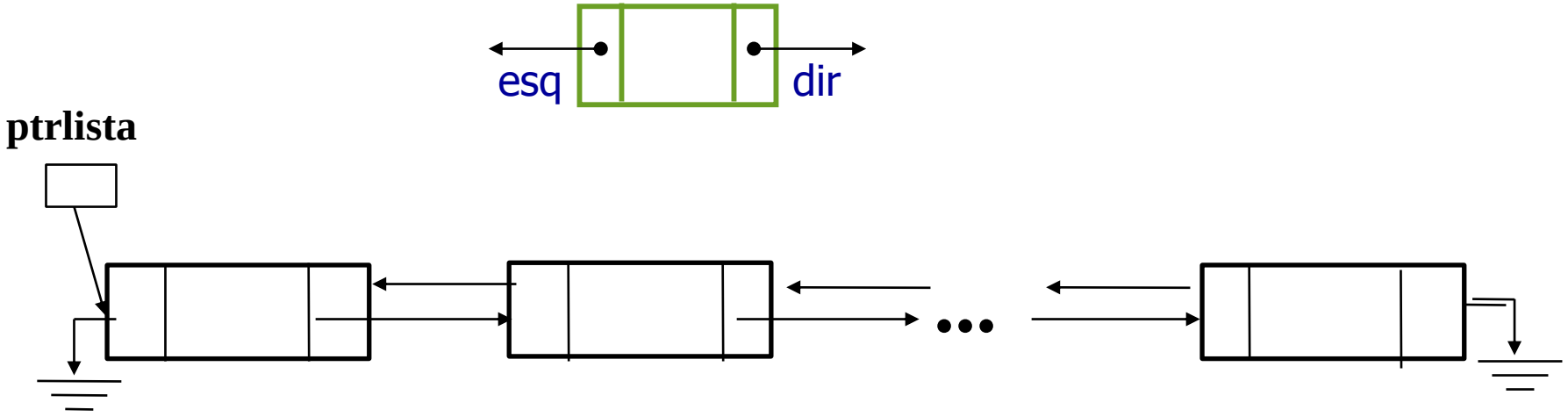
# Revisando listas simplesmente encadeadas

- Definição:

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


CELULA * ptrlista;
```

# Lista Duplamente Encadeada

Cada nó em uma lista duplamente encadeada possui dois ponteiros, um para seu predecessor (ou nó à esquerda) e outro para seu sucessor (ou nó à direita):



# Lista Duplamente Encadeada

## Vantagens

- A partir de um nó é possível acessar os nós adjacentes: direito e esquerdo
- É possível remover um elemento da lista conhecendo apenas o endereço do nó
- Facilita o processo de inserção à direita e à esquerda de um nó
- Permite que a lista seja percorrida em ambas as direções.

## Desvantagem

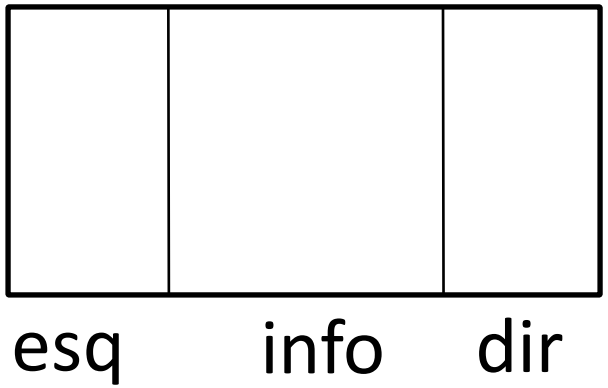
- Maior gasto de memória

# Lista Duplamente Encadeada

Podemos considerar os nós de uma lista duplamente encadeada consistindo de 3 campos:

- o campo **info** contém as informações armazenadas no nó;
- os campos **esq** e **dir**, que contém ponteiros para os nós de ambos os lados.

```
typedef struct cell
{
   int info;
   struct cell *esq;
   struct cell *dir;
}CELULA;

CELULA* ptrlista;
```



# Lista Duplamente Encadeada

Operações

- → inserção
- → remoção

# Lista Duplamente Encadeada (implementação)

Insere elemento no início da lista

```
void insere_inicio (CELULA **lista, int x) {
    CELULA *q;

    q = getnode ();
    if (q != NULL) {
        q->info = x;
        q->esq = NULL;
        q->dir = *lista;
        (*lista)-> esq = q;
        *lista = q;
    }
    else {
        printf ("\nERRO: falha na alocacao do noh.\n");
        exit(1);
    }
}
```

# Lista Duplamente Encadeada (implementação)

Inserir no final da lista

```
void insere_fim (CELULA **lista, int x) {
    CELULA *q;
    CELULA *aux;

    q = getnode ();
    if (q != NULL) {
        q->info = x;
        q->esq = NULL;
        q->dir = NULL;

        if (empty(*lista))
            *lista = q;
```

# Lista Duplamente Encadeada (implementação)

Inserir no final da lista (cont.)

```
        else{  //percorre lista até chegar ao ultimo nó
          aux = *lista;
          while (aux->dir != NULL)
               aux = aux->dir;

          aux->dir = q;
          q -> esq = aux;
      }
   } // Fim do if(q != NULL)
   else {
      printf ("\nERRO na alocação do nó.\n");
      exit(1);
    }
}
```

# Lista Duplamente Encadeada (implementação)

Remoção no início da lista

```
void remove_inicio (CELULA **lista) {
    CELULA *q;

    q = *lista;
    if (!empty(*lista)) {  //há itens na lista
        *lista = q->dir;
        (*lista)->esq = NULL;
        freenode (q);
    }
    else {
        printf ("\nERRO: lista vazia.\n");
        exit(1);
    }
}
```

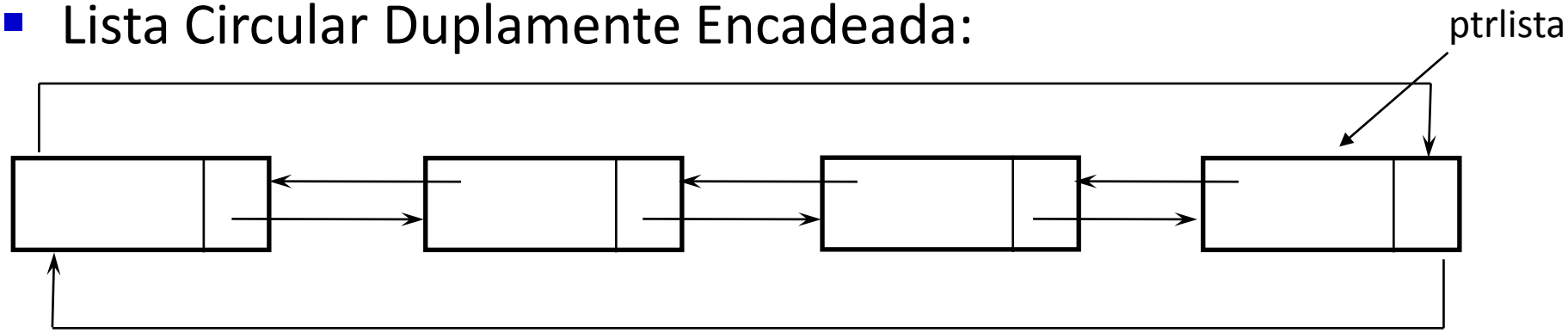
# Lista Duplamente Encadeada (implementação)

Remoção de um elemento na lista

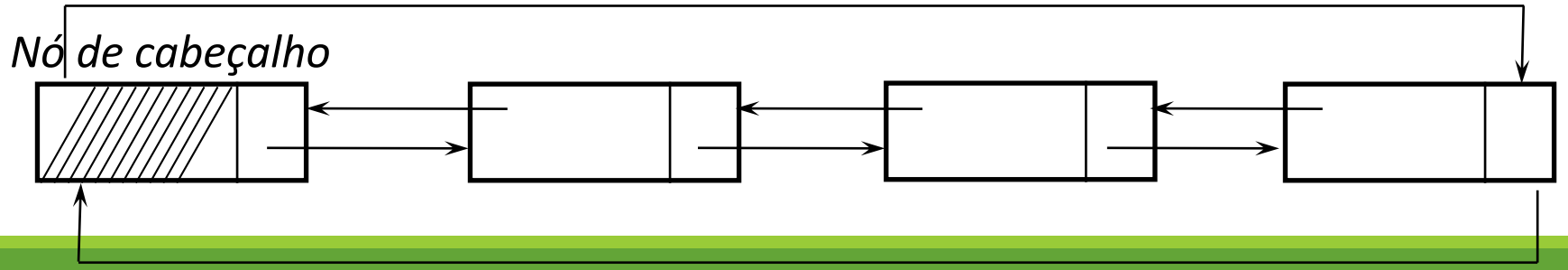
```
int remove_valor (CELULA **lista, int x) {
    CELULA *q;

    if ((q = pesquisa (*lista, x)) != NULL)
    {
        if(*lista == q) // nó está no início da lista
            remove_inicio (lista);
        else {
            (q->esq)->dir = q->dir;
            if(q->dir!=NULL)
                (q->dir)->esq = q->esq;
            freenode (q);
        }
        return 1; // removeu
    }
    return 0; //não removeu
}
```

# Lista Duplamente Encadeada

- Lista Circular Duplamente Encadeada:




- Lista Circular Duplamente Encadeada com cabeçalho



# Exercício

1) Crie um arquivo ListaDuplamenteEncadeada.c e implementa as seguintes informações de uma lista duplamente encadeada:
   - ✓ Definição
   - ✓ Operações
     - init
     - getnode
     - freenode
     - empty
     - exibe_lista
     - insere_inicio, insere_fim
     - remove_inicio, remove_valor
     - pesquisa

→ Teste seu programa criando um menu de opções para as principais operação.

# Leituras Recomendadas

- DROZDEK, Adam. Estrutura de Dados e Algoritmos em C++. Editora Pioneira Thomson Learning, 2005.

→Pág 85, seção 3.3 (Listas Circulares) - até pág. 96

→ Pág 80, seção 3.2 (Lista Duplamente Ligada) – até pág. 84

- TENENBAUM A., LANGSAM Y. e AUGENSTEIN M. J. Estrutura de Dados usando C. Editora Makron, 1995.

→Pág 279, seção 4.5 (Lista Circular) – até pág 280

→Pág 294 (Lista Duplamente Ligada) – até pág. 300

→ Pág 287 (Nós de Cabeçalho) – até pág. 291