

Aula Anterior

Lista Linear (não-recursiva)

```
void imprime(TipoPtNo* ptLista)
{
    TipoPtNo* ptaux;
    if (ptLista == NULL)
        puts("lista vazia");
    else
        for (ptaux=ptLista; ptaux!=NULL; ptaux=ptaux->prox)
            printf("Titulo = %s Distribuidor = %s Diretor = %s\n",
                ptaux->info.titulo,
                ptaux->info.distr,
                ptaux->info.diretor);
}
```

**O que mudar para imprimir
de forma reversa???**

```
void imprimeInv(TipoPtNo* l)
```

```
{
```

```
    TipoPtNo* ptaux;
```

```
    int numelem, i;
```

```
    numelem=0;
```

```
    ptaux = l;
```

```
    while (ptaux!=NULL) //descobrindo o numero de elementos da lista
```

```
    {
```

```
        ptaux = ptaux->prox;
```

```
        numelem++;
```

```
    }
```

```
    while (numelem>=1) //executa uma vez para cada elemento
```

```
    {
```

```
        ptaux = l;
```

```
        i=1;
```

```
        while(i<numelem) //vai até o i-ésimo elemento
```

```
        {
```

```
            i++;
```

```
            ptaux = ptaux->prox;
```

```
        }
```

```
        printf("Titulo = %s Distribuidor = %s Diretor = %s\n", ptaux->info.titulo, ptaux->info.distr, ptaux->info.diretor);
```

```
        numelem--;
```

```
    }
```

```
}
```

É eficiente?
Qual seria a solução???

Aula Anterior

Lista Linear (recursiva)

```
void imprime(TipoPtNo* ptLista)
```

```
{
```

```
    if (ptLista != NULL)
```

```
    {
```

```
        printf("Titulo = %s Distribuidor = %s Diretor = %s\n",
```

```
                ptLista->info.titulo,
```

```
                ptLista->info.distr,
```

```
                ptLista->info.diretor);
```

```
        imprime(ptLista->prox);
```

```
    }
```

```
}
```

É indicado usar recursão???

O que mudar para imprimir de forma reversa???

Aula Anterior

Lista Linear (recursiva)

```
void imprime(TipoPtNo* ptLista)
```

```
{
```

```
    if (ptLista != NULL)
```

```
    {
```

```
        imprime(ptLista->prox);
```

```
        printf("Titulo = %s Distribuidor = %s Diretor = %s\n",
```

```
               ptLista->info.titulo,
```

```
               ptLista->info.distr,
```

```
               ptLista->info.diretor);
```

```
        imprime(ptLista->prox);
```

```
    }
```

```
}
```

É indicado usar recursão???

O que mudar para imprimir de forma reversa???

Listas Lineares

Duplamente Encadeadas

Listas duplamente encadeadas

PtLista



Declaração- C

```
struct s_TipoInfoNo{
    int codigo;
    char nome[20];
    float preco;
};
typedef struct s_TipoInfoNo TipoInfoNo;
struct s_TipoPtNo{
    TipoInfoNo info;
    struct s_TipoPtNo* ant;
    struct s_TipoPtNo* prox;
};
typedef struct s_TipoPtNo TipoPtNo;
```

Anterior Info Próximo



nodo

Operações sobre listas duplamente encadeadas

- criar lista, lendo dados de arquivo / teclado
- listar todos os nodos da lista
- listar de trás para diante
- inserir um nodo antes / depois de determinado nodo
- remover o primeiro / último / k-ésimo nodo
- trocar a ordem de 2 nodos
- ...

Inicializa

```
TipoPtNo* inicializa(void)
{
    return NULL;
}
```


Imprimir todos os elementos da lista

```
void imprime(TipoPtNo* PtLista)
{
    TipoPtNo* ptaux=PtLista;
    if (PtLista == NULL)
        puts("lista vazia");
    else
        do {
            printf("Codigo = %d Nome = %s Preco = %f\n",
                ptaux->info.codigo,
                ptaux->info.nome,
                ptaux->info.preco);
            ptaux = ???;
        } while (???);
}
```

Imprimir todos os elementos da lista

```
void imprime(TipoPtNo* PtLista)
{
    TipoPtNo* ptaux=PtLista;
    if (PtLista == NULL)
        puts("lista vazia");
    else
        do {
            printf("Codigo = %d Nome = %s Preço = %f\n",
                ptaux->info.codigo,
                ptaux->info.nome,
                ptaux->info.preco);
            ptaux = ptaux->prox;
        } while (ptaux != NULL);
}
```

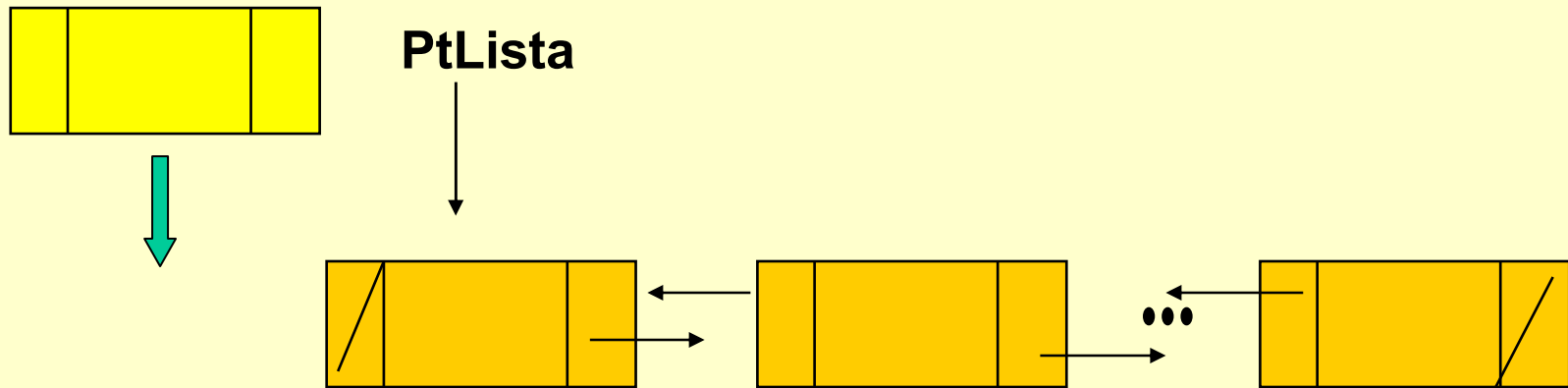
Imprimir de trás para diante as Info de uma lista duplamente encadeada

```
void ImprimirLLDEInvertida(TipoPtNo *PtLista){
    TipoPtNo *PtAux;
    if (PtLista==NULL){
        printf("Lista vazia ! ");
    }else
    {
        PtAux=PtLista;
        while (PtAux->Prox!=NULL) PtAux=PtAux->Prox;
        while (????)
        {
            printf("Elemento: %d %s\n",PtAux->info.cod,PtAux->info.nome);
            PtAux=????;
        }
        printf("Elemento: %d %s\n",PtAux->info.cod,PtAux->info.nome);
    }
}
```

Imprimir de trás para diante as Info de uma lista duplamente encadeada

```
void ImprimirLLDEInvertida(TipoPtNo *PtLista){
    TipoPtNo *PtAux;
    if (PtLista==NULL){
        printf("Lista vazia ! ");
    }else
    {
        PtAux=PtLista;
        while (PtAux->Prox!=NULL) PtAux=PtAux->Prox;
        while (PtAux!=PtLista)
        {
            printf("Elemento: %d %s\n",PtAux->info.cod,PtAux->info.nome);
            PtAux=PtAux->Ant;
        }
        printf("Elemento: %d %s\n",PtAux->info.cod,PtAux->info.nome);
    }
}
```

Inserir um novo nodo no início



Inserir um novo nodo no início

```
TipoPtNo* InserirNodoInicio(TipoPtNo *PtLista, TipoInfoNo Dado)
{
    TipoPtNo *Pt;
    Pt = (TipoPtNo*) malloc(sizeof(TipoPtNo));

    Pt->info = Dado;

    Pt->Ant = ???;
    Pt->Prox = ???;
    if (PtLista != NULL)
        PtLista->Ant = ???;
    PtLista = ???;
    return PtLista;
}
```

Solução com função

Inserir um novo nodo no início

```
TipoPtNo* InserirNodoInicio(TipoPtNo *PtLista, TipoInfoNo Dado)
{
    TipoPtNo *Pt;
    Pt = (TipoPtNo*) malloc(sizeof(TipoPtNo));


    Pt->info = Dado;

    Pt->Ant = NULL;
    Pt->Prox = PtLista;
    if (PtLista != NULL)
        PtLista->Ant = Pt;
    PtLista = Pt;
    return PtLista;
}
```

Solução com função

Inserir um novo nodo no início

Ponteiro de ponteiro



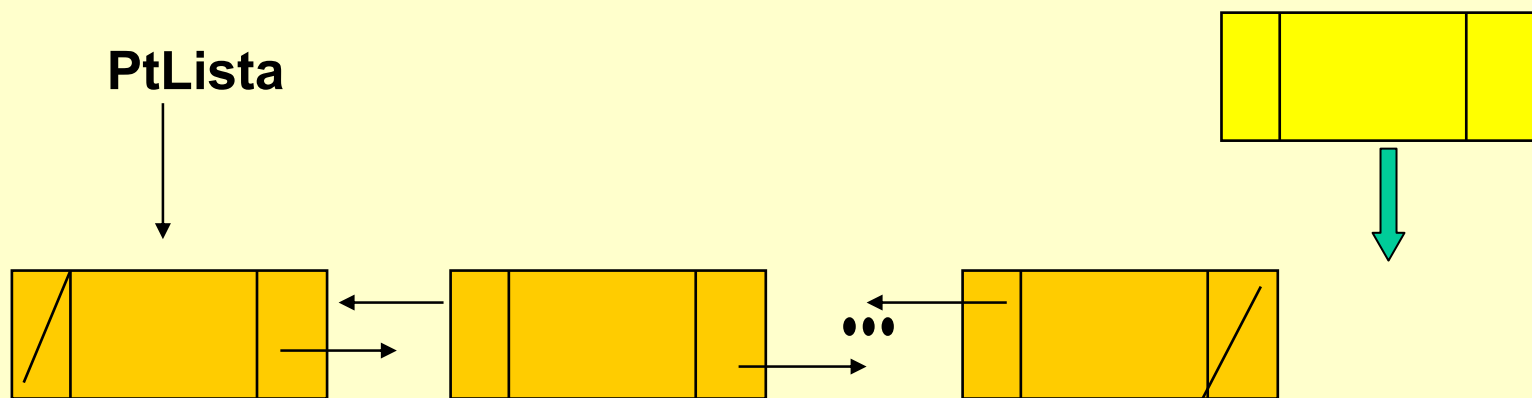
```
void InserirInicio(TipoPtNo **PtLista, TipoInfoNo Dado)
{
    TipoPtNo *Pt;
    Pt = (TipoPtNo*) malloc(sizeof(TipoPtNo));

    Pt->info = Dado;

    Pt->Ant = NULL;
    Pt->Prox = *PtLista;
    if (*PtLista != NULL)
        (*PtLista)->Ant = Pt;
    *PtLista = Pt;
}
```

Solução com procedimento

Inserir um novo nodo no fim



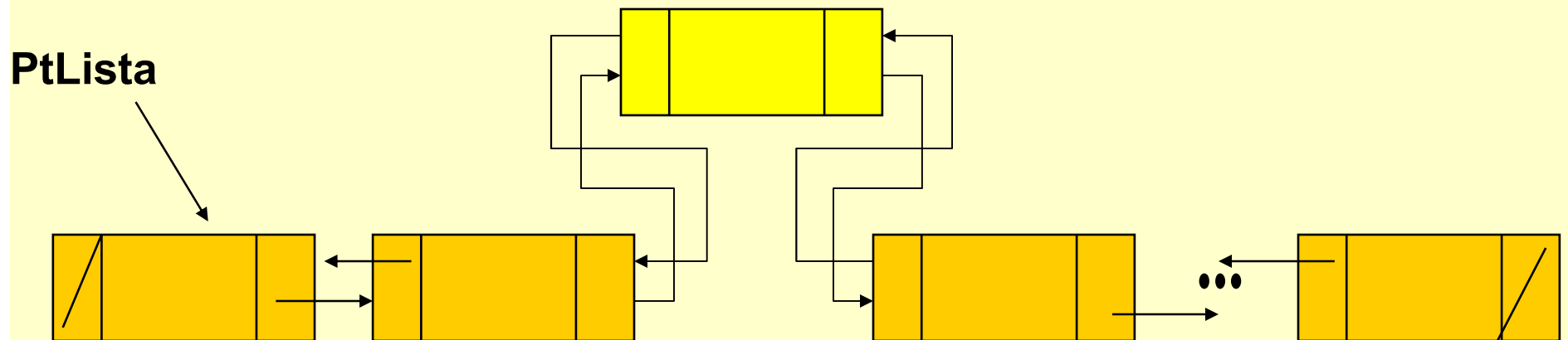
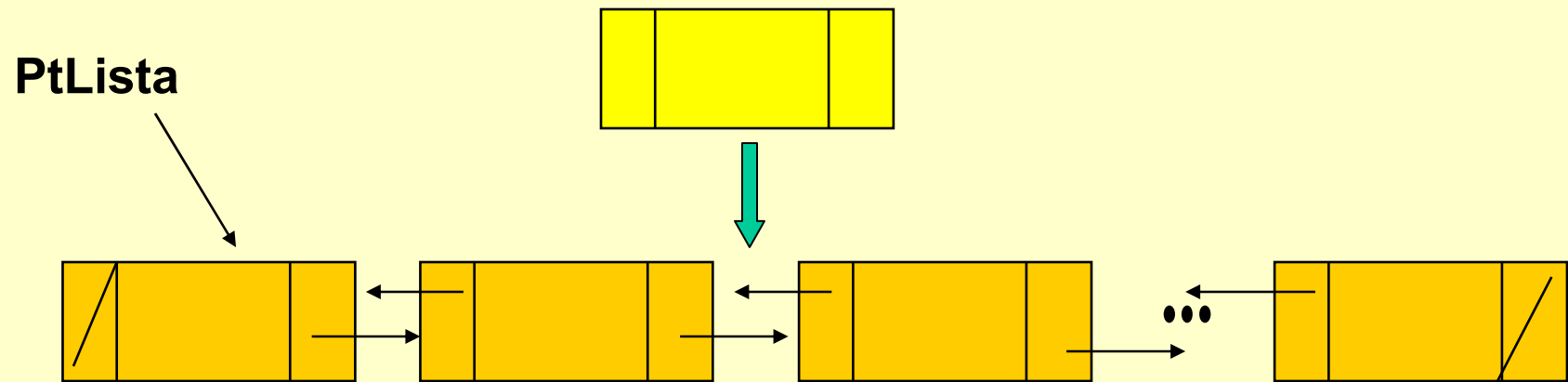
Inserir um novo nodo no final em C

```
TipoPtNo* InserirnodoFinal(TipoPtNo *PtLista, TipoInfoNo Dado)
{
    TipoPtNo *Pt, *PtAux;
    Pt = (TipoPtNo*) malloc(sizeof(TipoPtNo));
    Pt->info = Dado;
    if ((PtLista) == NULL)
    {
        PtLista = ???;
        Pt->Ant = ???;
        Pt->Prox = ???;
    }
    else {
        PtAux = PtLista;
        while (PtAux->Prox != NULL)
            PtAux=PtAux->Prox;
        PtAux->Prox = ???;
        Pt->Ant = ???;
        Pt->Prox = ???;
    }
    return PtLista;
}
```

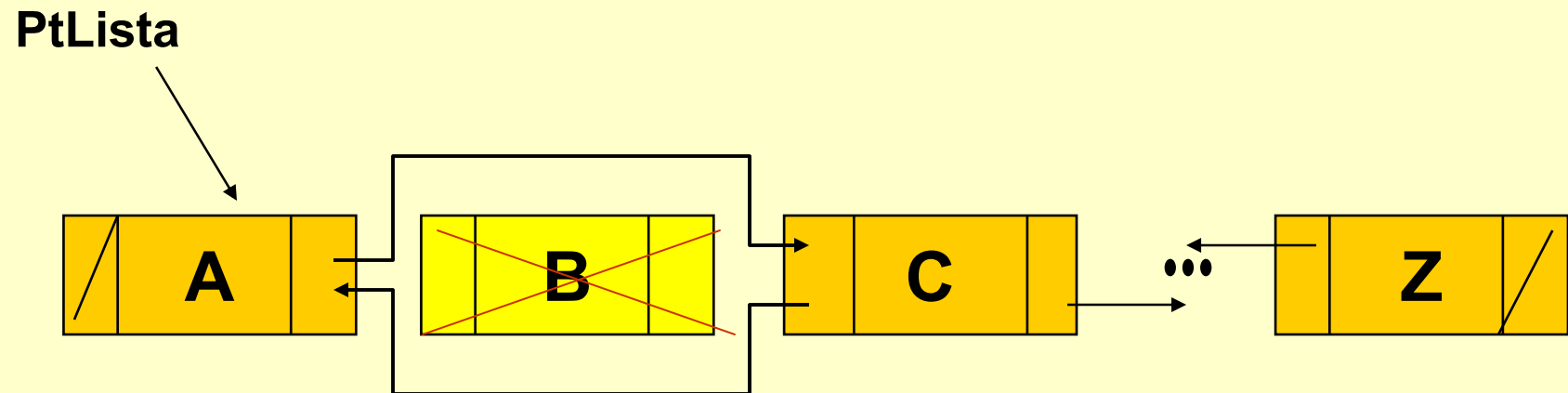
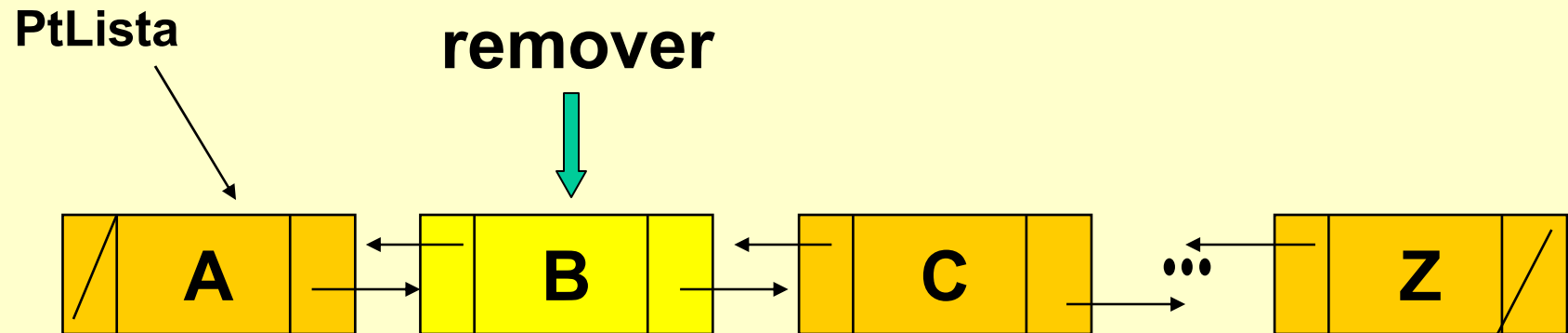
Inserir um novo nodo no final em C

```
TipoPtNo* InserirnodoFinal(TipoPtNo *PtLista, TipoInfoNo Dado)
{
    TipoPtNo *Pt, *PtAux;
    Pt = (TipoPtNo*) malloc(sizeof(TipoPtNo));    //aloca novo nodo
    Pt->info = Dado;                               //coloca dados no novo nodo
    if ((PtLista) == NULL)                         //primeiro nodo
    {
        PtLista = Pt;
        Pt->Ant = NULL;
        Pt->Prox = NULL;
    }
    else {
        PtAux = PtLista;                          //auxiliar no início da lista
        while (PtAux->Prox != NULL)                //PtAux avança até o final da lista
            PtAux=PtAux->Prox;
        PtAux->Prox = Pt;
        Pt->Ant = PtAux;                          //Encadeia Pt com PtAux
        Pt->Prox = NULL;
    }
    return PtLista;
}
```

Inserir um novo nodo no meio



Remover um novo nodo do meio



Destrói Lista

```
TipoPtNo* destroi(TipoPtNo* ptLista)
{
    TipoPtNo *ptaux; //ponteiro auxiliar para percorrer a lista
    while (ptLista != NULL)
    {
        ptaux = ptLista;
        ptLista = ptLista->prox;
        free(ptaux);
    }
    free(ptLista);
    return NULL;
}
```