

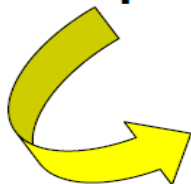
Estrutura de Dados - I

Tipos de dados

Prof. MSc. Rafael Staiger Bressan
rafael.bressan@unicesumar.edu.br

Tipos de dados

Tipo de dado



definição do conjunto de valores (domínio)
que uma variável pode assumir

Ex: inteiro

< ... -2, -1, 0, +1, +2, ... >

lógico

< verdadeiro, falso >



Estrutura de Dados

- **Tipos básicos (primitivos)**
 - inteiro, real, e caractere
- **Tipos de estruturados (construídos)**
 - arranjos (vetores e matrizes)
 - Estruturas sequências (conjuntos)
 - referências (ponteiros)
- **Tipos definidos pelo usuário**



Estrutura de Dados

- **Tipos de dados básicos**
 - Fornecidos pela Linguagem de Programação
- **Estruturas de Dados**
 - Estruturação *conceitual* dos dados
 - Reflete um **relacionamento lógico** entre dados, de acordo com o problema considerado

Estrutura de Dados

Lista Linear

- Relação de ordem entre os dados
- Linear - seqüencial



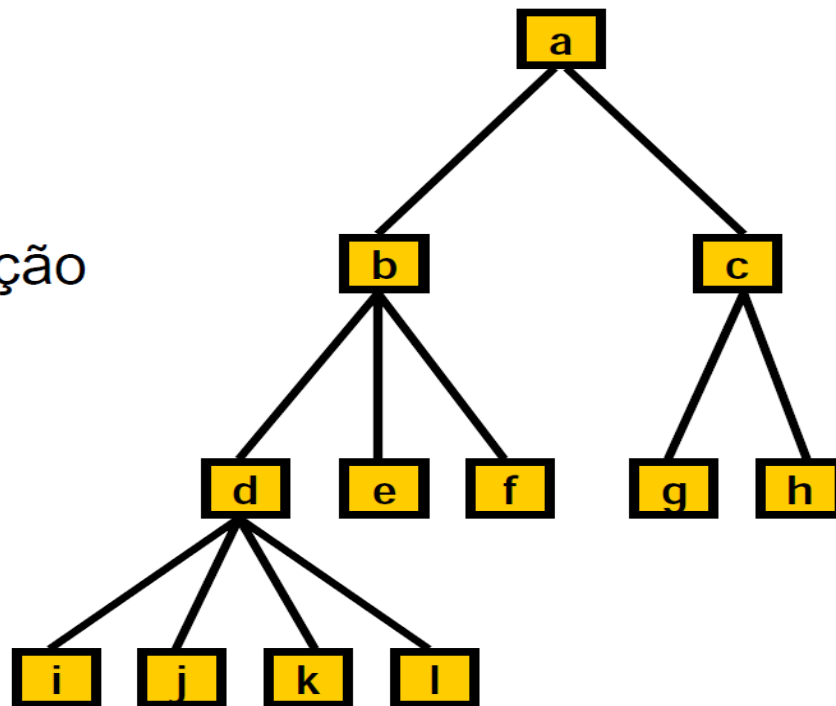
Ex:

aplicação: empresa

problema: dados dos funcionários – cada nó um funcionário

Estrutura de Dados Árvore

- Relação de subordinação entre os dados



Ex:

aplicação: empresa

problema: organograma de funções



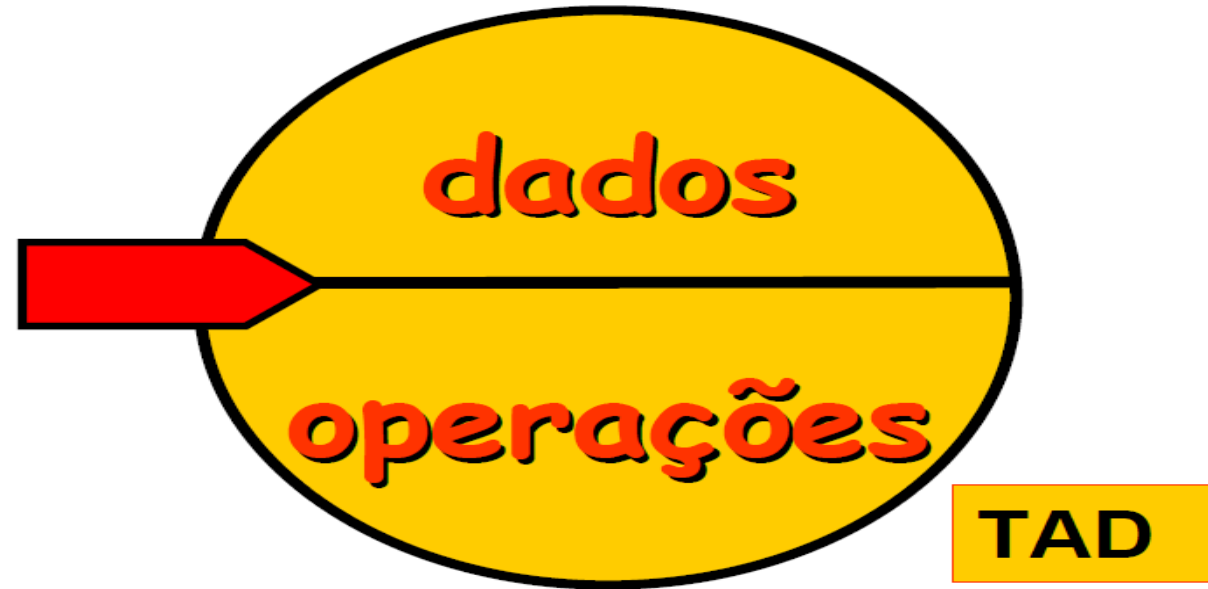
Estrutura de Dados

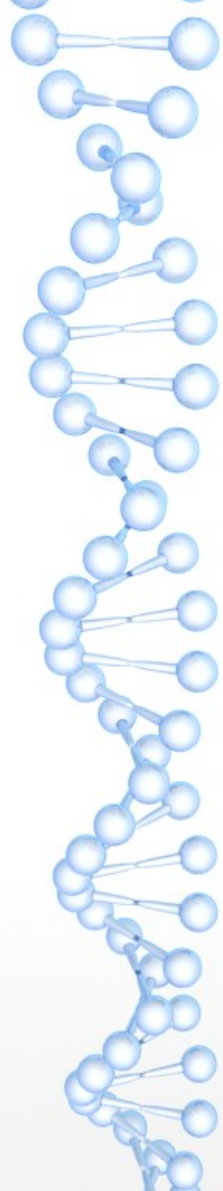
Operações

- Estruturas de Dados incluem as operações para a manipulação de seus dados
- Operações básicas:
 - **Criação** da estrutura de dados
 - **Inclusão** de um novo elemento
 - **Remoção** de um elemento
 - **Acesso** a um elemento
 - **Destruição** da estrutura de dados

Estrutura de Dados

TADs - Tipos Abstratos de Dados

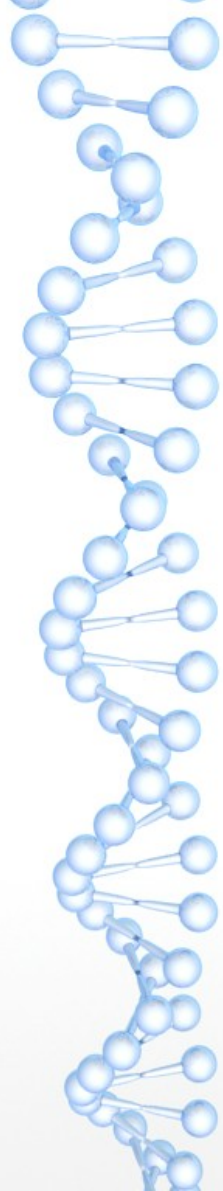




Estrutura de Dados

TADs - Tipos Abstratos de Dados

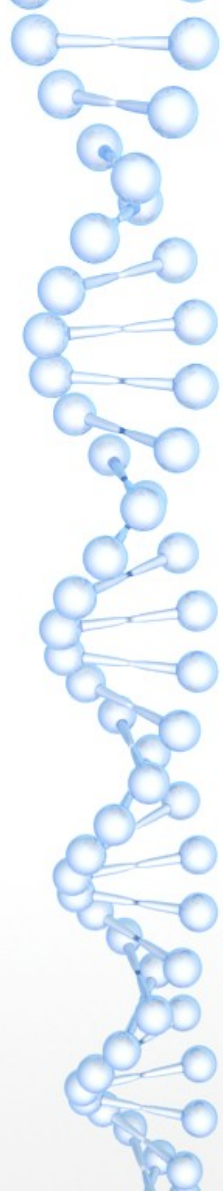
- TAD é uma forma de definir um novo tipo de dado juntamente com as operações que manipulam esse novo tipo de dado



Estrutura de Dados

TADs - Tipos Abstratos de Dados

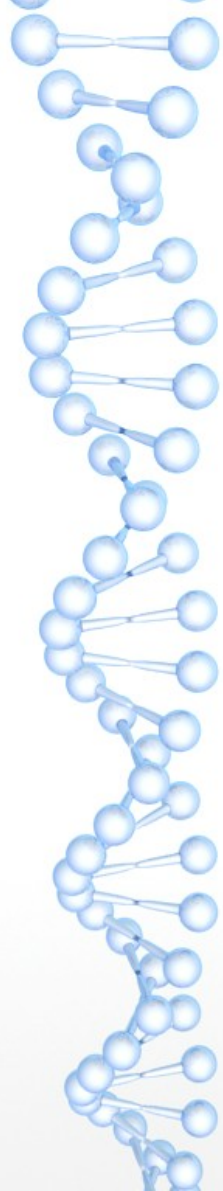
- Separação entre conceito (definição do tipo) e implementação das operações
- Visibilidade da estrutura internado tipo fica limitada às operações
- Aplicações que usam o TAD são denominadas *clientes* do tipo de dado
- Cliente tem acesso somente à forma abstrata do TAD



Estrutura de Dados

TADs - Tipos Abstratos de Dados

- Um TAD (em LP) é um tipo de dado que satisfaz as condições:
- A representação ou a definição do tipo e as operações sobre variáveis desse tipo estão contidas numa única unidade sintática
 - MÓDULO



Estrutura de Dados

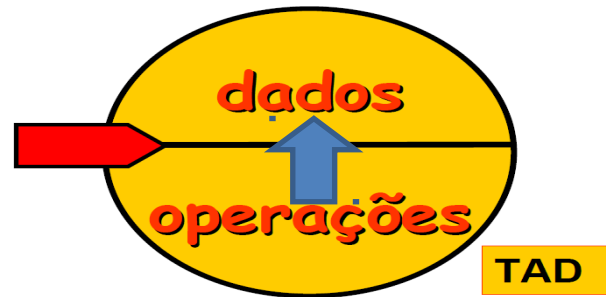
TADs - Tipos Abstratos de Dados

- A representação interna do tipo (a implementação) não é visível de outras unidades sintáticas, de modo que só as operações oferecidas na definição do tipo podem ser usadas com as variáveis desse dados.

Estrutura de Dados

Propriedades - TADs

- Satisfazem as propriedades de
 - encapsulamento: definição isolada de outras unidades do programa
 - Invisibilidade e proteção: representação do tipo deve ser acessada somente no ambiente encapsulado





Estrutura de Dados

Propriedades - TADs

- A LP deve possibilitar
 - ambiente encapsulado
 - proteção de dados
 - interface para acesso
 - operações básicas



Estrutura de Dados

Vantagens - TADs

- Possibilidade de utilização do mesmo TAD em diversas aplicações diferentes
- Possibilidade de alterar o TAD sem alterar as aplicações que o utilizam.
 - **Reutilização**



Projeto de um TAD

- Envolve a escolha de operações adequadas para uma determinada estrutura de dados, definindo seu comportamento.

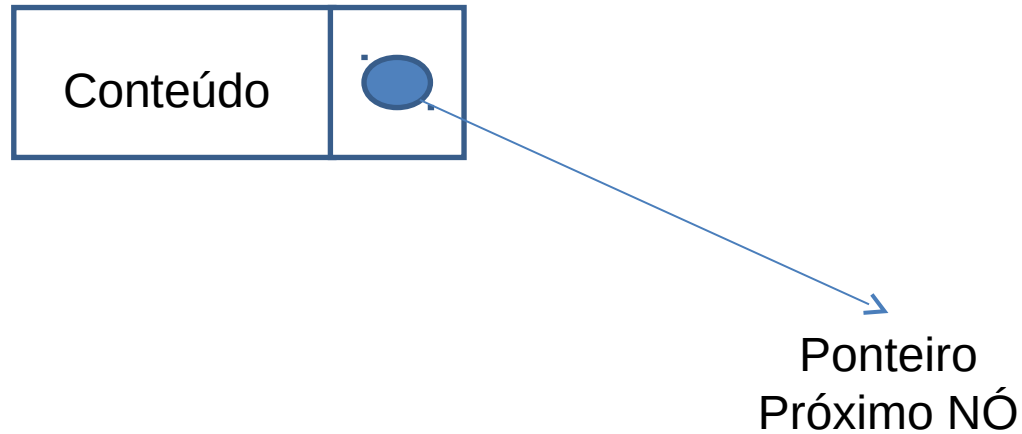


Projeto de um TAD

- **Dicas para definir um TAD:**
 - definir pequeno número de operações
 - conjunto de operações deve ser suficiente para realizar as computações necessárias às aplicações que utilizarem o TAD
 - cada operação deve ter um propósito bem definido, com comportamento constante e coerente

Listas Linear Simplesmente Encadeada

Estrutura de um Nó



Listas Linear Simplesmente Encadeada

Lista

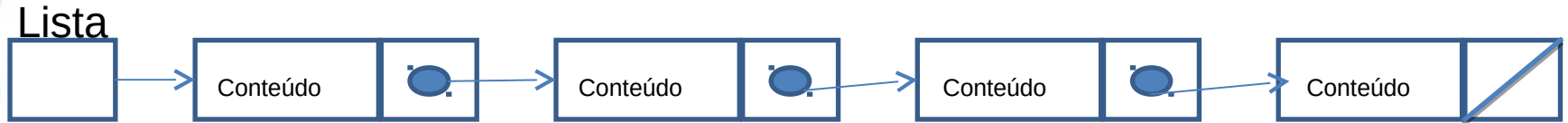


Lista Vazia.
A lista é um ponteiro
Aponta para NULL

Listas Linear Simplesmente Encadeada



Listas Linear Simplesmente Encadeada





Listas lineares

- Uma lista é uma estrutura que armazena elementos de forma alinhada, ou seja, com elementos dispostos um após o outro.



Listas lineares

- Estudo de listas lineares e das operações básicas sobre elas, considerando as diferentes formas de implementação física



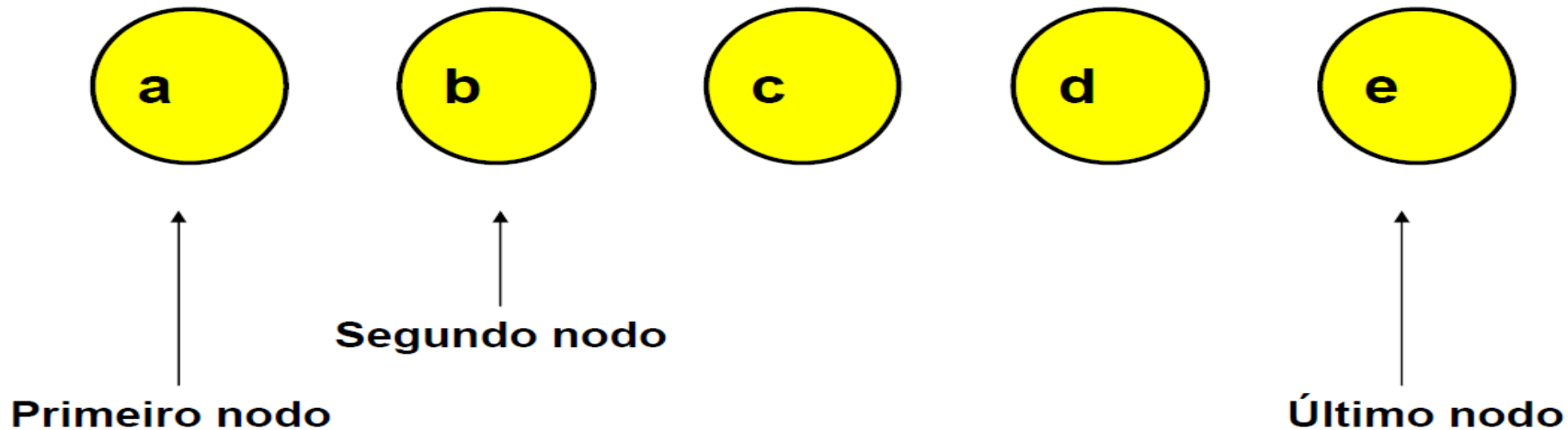
Listas lineares

Uma Lista Linear (**LL**) é uma **sequência** de nodos



- **Nodos - elementos do mesmo tipo**
- **Relação de ordem linear (ou total)**

Listas lineares



Listas lineares

- Estrutura interna é abstraída
- Pode ter uma complexidade arbitrária
- Enfatizado o conjunto de relações existente



INFORMAÇÕES				
Número	RG	Nome	Nasc.	Cargo



Listas lineares

Uma lista linear é uma coleção de $n \geq 0$ nodos x_1, x_2, \dots, x_n , todos do mesmo tipo, cujas propriedades estruturais relevantes envolvem apenas as posições relativas lineares entre nodos:

$n = 0$: lista vazia, apresenta zero nodos

$n > 0$: x_1 é o primeiro nodo

x_n é o último nodo

$1 < k < n$: x_k é precedido por x_{k-1} e sucedido por x_{k+1}



Listas lineares

- Aplicações
 - Notas de alunos
 - Cadastro de funcionários de uma empresa
 - Itens em estoque em uma empresa
 - Dias da semana
 - Letras de uma palavra
 - Pessoas esperando ônibus
 - Cartas de baralho
 - Lista telefônica



Listas lineares

- Operações básicas:
 - Criação de uma lista
 - Inserção de um nodo
 - Exclusão de um nodo
 - Acesso a um nodo
 - Destruição de uma lista



Listas lineares

- Disciplina de acesso refere-se à forma como os elementos de uma lista linear são acessados, inseridos e removidos.



Listas lineares

- Se os elementos de uma lista linear só podem ser inseridos, acessados ou removidos da última posição, chamamos esta lista **linear de pilha** (***LIFO - Last In First Out***);



Listas lineares

- Se os elementos de uma lista linear sópo dem ser inseridos na última posição e acessados ou removidos da primeira posição, chamamos esta **lista linear de fila (FIFO – *First In First Out*)**;



Alocação de Memória

- Como armazenar os elementos de uma lista?
- A alocação de memória para implementar uma lista pode ser **estática** ou **dinâmica**.



Alocação de Memória

- ## Alocação estática

- Área de memória é alocada no momento da compilação
- Uma lista com alocação estática de memória exige uma definição do número máximo de elementos **super** ou **sub** dimensionamento do tamanho da lista.



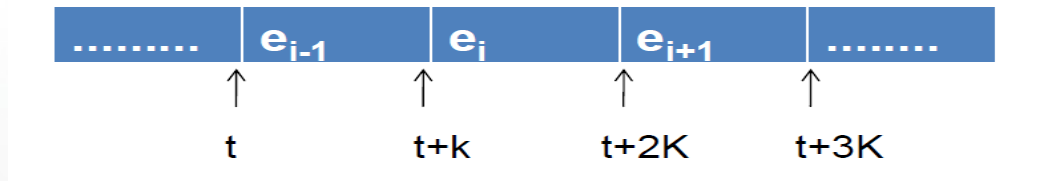
Alocação de Memória

- **Alocação dinâmica:**
 - O espaço de memória é alocado em tempo de execução.
 - Uma lista com alocação dinâmica cresce à medida que novos elementos precisam ser armazenados (e diminui à medida que elementos anteriormente armazenados são retirados da lista).

Acesso aos elementos de uma lista

- **Acesso sequencial**

- Os elementos de uma lista são armazenados de forma consecutiva na memória.
- Exemplo: considere que cada elemento da lista tenha tamanho k
- o endereço de um elemento e_i é facilmente calculado.





Acesso aos elementos de uma lista

- Acesso encadeado
 - Os elementos de uma lista podem ocupar quaisquer áreas de memória, não necessariamente consecutivas para preservar a relação de ordem de uma lista linear, cada elemento da lista deve armazenar sua informação e o endereço de memória onde se encontra o próximo elemento.
 - O endereço do elemento e_i não pode ser facilmente calculado.



Acesso aos elementos de uma lista

- Combinações possíveis:
 - Alocação estática X alocação dinâmica.
 - Acesso sequencial X acesso encadeado.



Listas encadeadas

- Uma lista encadeada é uma **representação de uma sequência de objetos**, todos do mesmo tipo, na memória rápida (RAM) do computador.
- Cada elemento da sequência é armazenado em uma célula da lista: o primeiro elemento na primeira célula, o segundo na segunda e assim por diante.

Estrutura de uma lista encadeada

- Uma *lista encadeada* (= *linked list* = lista ligada) é uma sequência de *células*;
- Cada célula contém um objeto de algum tipo e o endereço da célula seguinte.
- Exemplo de definição de um objetos inteiros:



```
4 struct cel {  
5     int         conteudo;  
6     struct cel *prox;  
7 };
```




Listas encadeadas

- É conveniente tratar as células como um novo tipo-de-dados e atribuir um nome a esse novo tipo:

```
11 | typedef struct cel celula; // célula
```

- Uma célula `c` e um ponteiro `p` para uma célula podem ser declarados assim:

```
celula c; celula *p;
```



Listas encadeadas

- Se c é uma célula então $c.conteudo$ é o conteúdo da célula e $c.prox$ é o endereço da próxima célula.
- Se p é o endereço de uma célula, então:
 - $p->conteudo$ é o conteúdo da célula e
 - $p->prox$ é o endereço da próxima célula.
- Se p é o endereço da última célula da lista então $p->prox$ vale **NULL**.

$p->conteudo$ é uma simplificação de $(*p).conteudo$

Listas encadeadas

- **p->conteudo** é o conteúdo da célula e
- **p->prox** é o endereço da próxima célula.





Listas com cabeça e sem cabeça

- Uma lista encadeada pode ser organizada de duas maneiras diferentes, uma óbvia e outra menos óbvia.
 - Lista *com* cabeça.
 - Lista *sem* cabeça.

Lista *com* cabeça.

- O conteúdo da primeira célula é irrelevante: ela serve apenas para marcar o início da lista.
- A primeira célula é a *cabeça* (= *head cell* = *dummy cell*) da lista.
- Digamos que *ini* é o endereço da primeira célula. Então *ini*->*prox* == NULL se e somente se a lista está vazia.





Lista *com* cabeça.

- Para criar uma lista vazia, basta dizer
celula c, *ini;
c.prox = NULL;
ini = &c;
- ou, se preferir alocar a primeira célula dinamicamente:
celula *ini;
ini = malloc (sizeof (celula));
ini->prox = NULL;



Lista *sem* cabeça.

- O conteúdo da primeira célula é tão relevante quanto o das demais.
- Nesse caso, a lista está vazia se o endereço de sua primeira célula (que não existe) é NULL.
- Para criar uma lista vazia basta fazer

celula *ini;

ini = NULL;

Exemplo

Imprime o conteúdo de uma lista encadeada *com* cabeça:

```
void imprima (celula *ini) {  
    celula *p;  
    for (p = ini->prox; p != NULL; p = p->prox)  
        printf ("%d\n", p->conteudo);  
}
```


Exemplo

Imprime o conteúdo de uma lista encadeada *sem* cabeça:

```
void imprima (celula *ini) {  
    celula *p;  
    for (p = ini; p != NULL; p = p->prox)  
        printf ("%d\n", p->conteudo);  
}
```



Endereço de uma lista encadeada

- O endereço de uma lista encadeada é o endereço de sua primeira célula.
- Se p é o endereço de uma lista, convém, às vezes, dizer simplesmente **p é uma lista**.
- Listas são eminentemente recursivos. Para tornar isso evidente, basta fazer a seguinte observação: se p é uma lista não vazia então **$p \rightarrow \text{prox}$** também é uma lista.



Busca em uma lista encadeada

- Veja como é fácil verificar se um objeto x pertence a uma lista encadeada, ou seja, se é igual ao conteúdo de alguma célula da lista:



Busca em uma lista encadeada

```
celula *busca (int x, celula *ini)
{
    celula *p;
    p = ini->prox;
    while (p != NULL && p->conteudo != x)
        p = p->prox;
    return p;
}
```



Versão recursiva da mesma função:

```
celula *buscaR (int x, celula *ini) {  
    if (ini->prox == NULL)  
        return NULL;  
    if (ini->prox->conteudo == x)  
        return ini->prox;  
    return buscaR (x, ini->prox);  
}
```



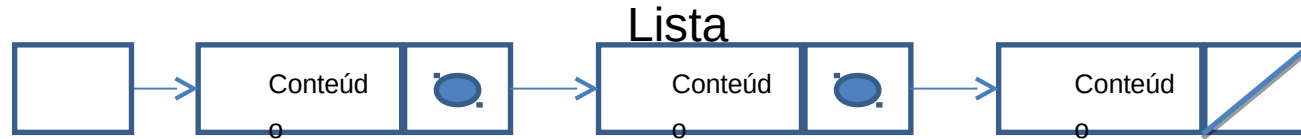
Inserção em uma lista

- Considere o problema de *inserir* (= *to insert*) uma nova célula em uma lista encadeada.
- Suponha que quero inserir a nova célula entre a posição apontada por *p* e a posição seguinte. (É claro que isso só faz sentido se *p* é diferente de NULL.)

Inserção em uma lista

→ X = 68 *p = end_primeiro

```
void insere (int x, celula *p) {  
    celula *nova; nova = malloc (sizeof (celula));  
    nova->conteudo = x;  
    nova->prox = p->prox;  
    p->prox = nova;  
}
```

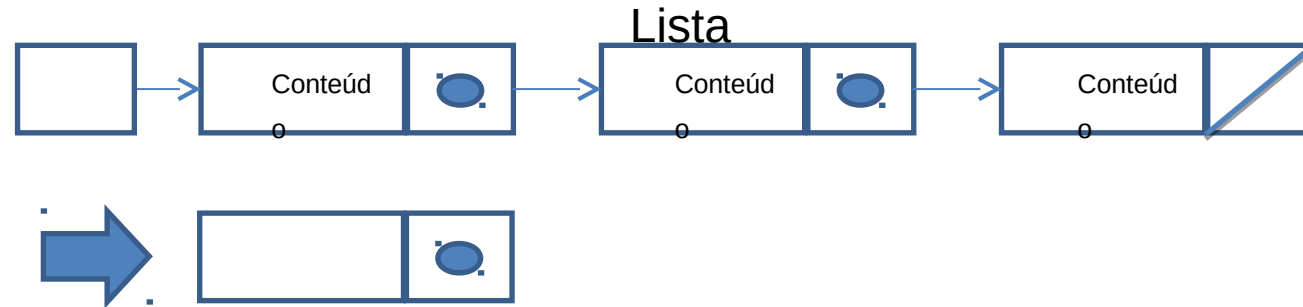


X = 68

*p = end_primeiro

Inserção em uma lista

```
void insere (int x, celula *p) {  
    celula *nova; nova = malloc (sizeof (celula));  
    nova->conteudo = x;  
    nova->prox = p->prox;  
    p->prox = nova;  
}
```

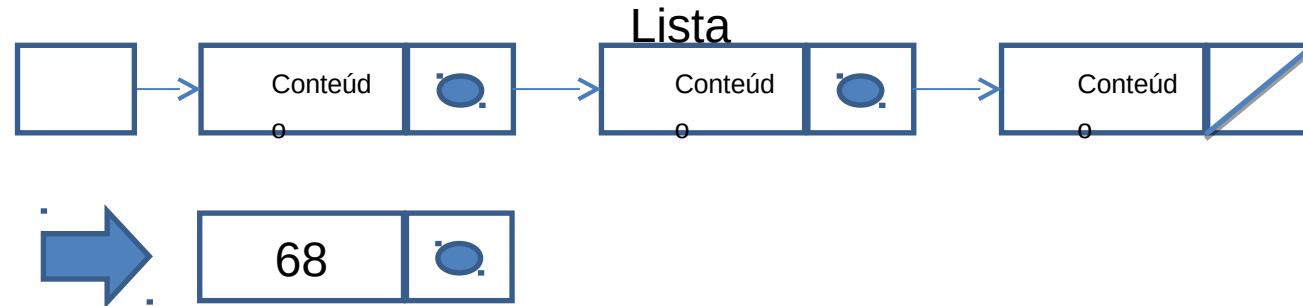


X = 68

*p = end_primeiro

Inserção em uma lista

```
void insere (int x, celula *p) {  
    celula *nova; nova = malloc (sizeof (celula));  
    nova->conteudo = x;  
    nova->prox = p->prox;  
    p->prox = nova;  
}
```

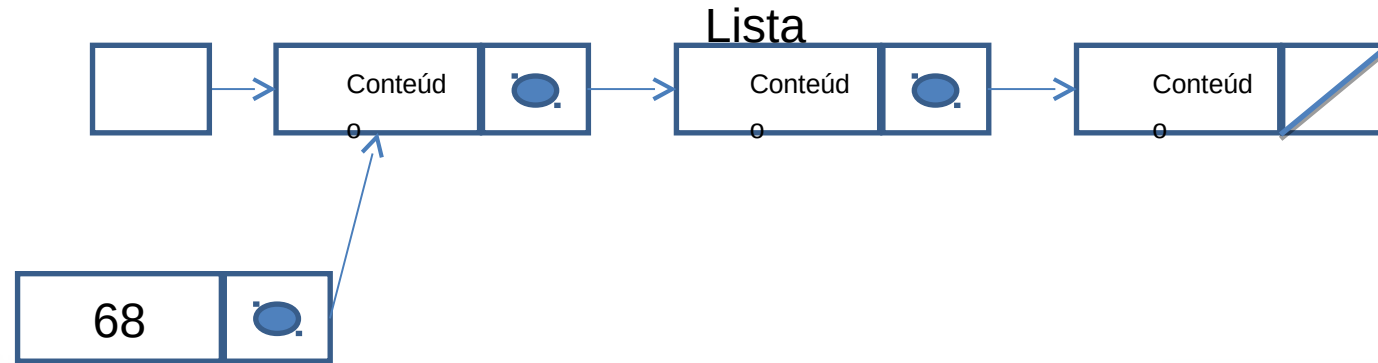


X = 68

*p = end_primeiro

Inserção em uma lista

```
void insere (int x, celula *p) {  
    celula *nova; nova = malloc (sizeof (celula));  
    nova->conteudo = x;  
    nova->prox = p->prox;  
    p->prox = nova;  
}
```

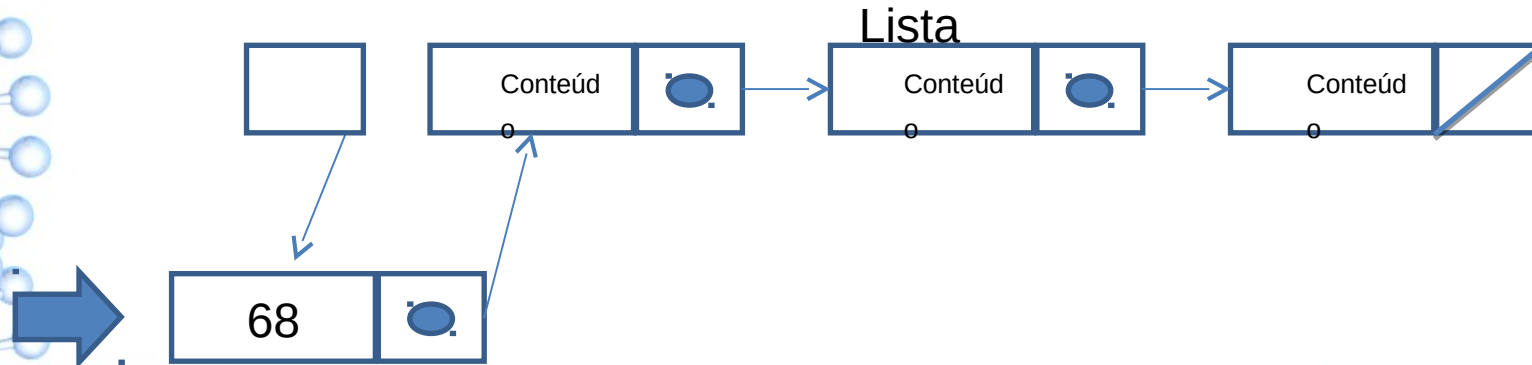


X = 68

*p = end_primeiro

Inserção em uma lista

```
void insere (int x, celula *p) {  
    celula *nova; nova = malloc (sizeof (celula));  
    nova->conteudo = x;  
    nova->prox = p->prox;  
    p->prox = nova;  
}
```

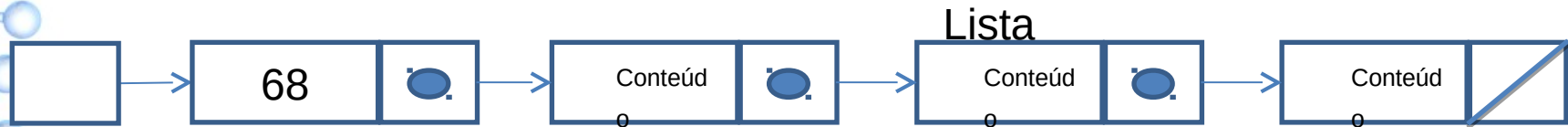


X = 68

*p = end_primeiro

Inserção em uma lista

```
void insere (int x, celula *p) {  
    celula *nova; nova = malloc (sizeof (celula));  
    nova->conteudo = x;  
    nova->prox = p->prox;  
    p->prox = nova;  
}
```





Inserção em uma lista

- Não é preciso movimentar células para criar espaço para uma nova célula, como fizemos para inserir um novo elemento em um vetor. Basta mudar os valores de alguns ponteiros.



Remoção em uma lista

- Considere o problema de *remover* (= *to remove* = *to delete*) uma certa célula da lista. Como especificar a célula em questão? A ideia mais óbvia é apontar para a célula que quero remover. Mas é fácil perceber que essa ideia não é boa; é melhor apontar para a célula *anterior* à que quero remover. Infelizmente, essa decisão de projeto traz uma nova dificuldade: não há como pedir a remoção da *primeira* célula. Portanto, vamos nos limitar às listas com cabeça.

Remoção em uma lista

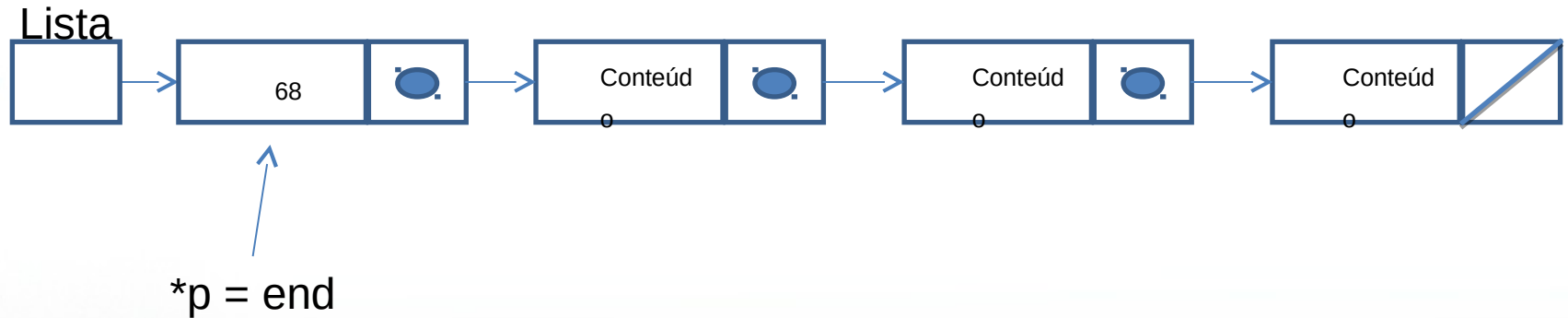
```
void remove (celula *p) {  
    celula *morta;  
    morta = p->prox;  
    p->prox = morta->prox;  
    free (morta);  
}
```

Lista



Remoção em uma lista

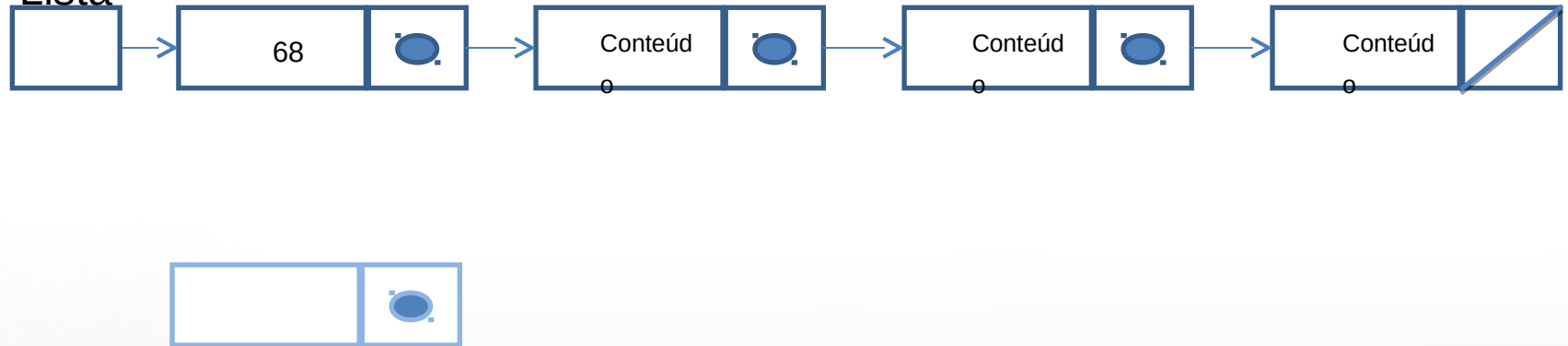
```
void remove (celula *p) {  
    celula *morta;  
    morta = p->prox;  
    p->prox = morta->prox;  
    free (morta);  
}
```



Remoção em uma lista

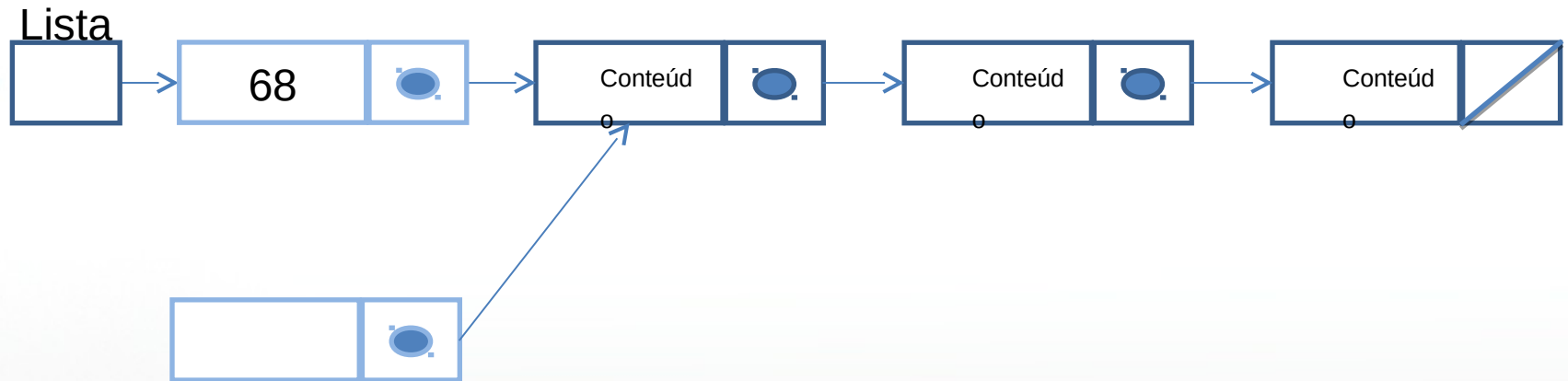
```
void remove (celula *p) {  
    celula *morta;  
    morta = p->prox;  
    p->prox = morta->prox;  
    free (morta);  
}
```

Lista



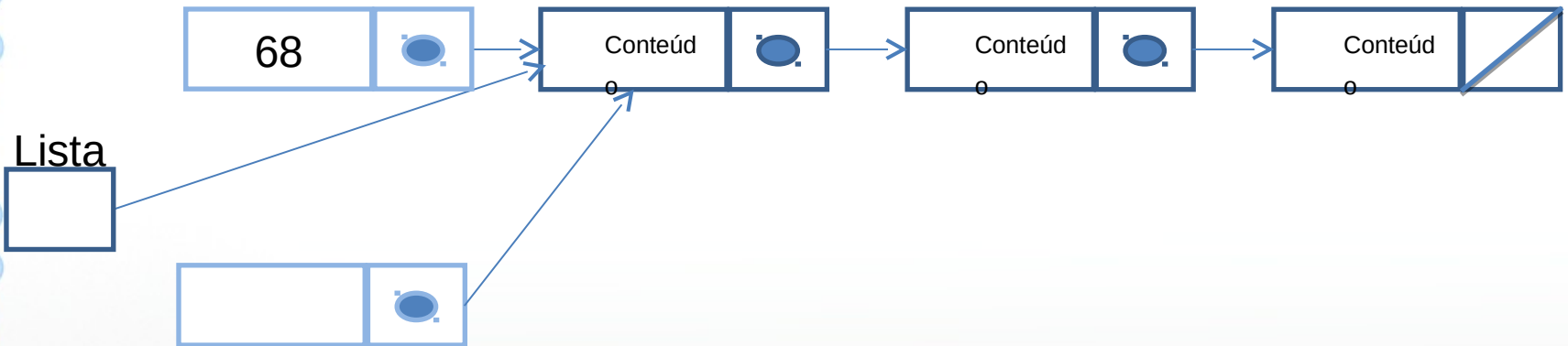
Remoção em uma lista

```
void remove (celula *p) {  
    celula *morta;  
    morta = p->prox;  
    p->prox = morta->prox;  
    free (morta);  
}
```



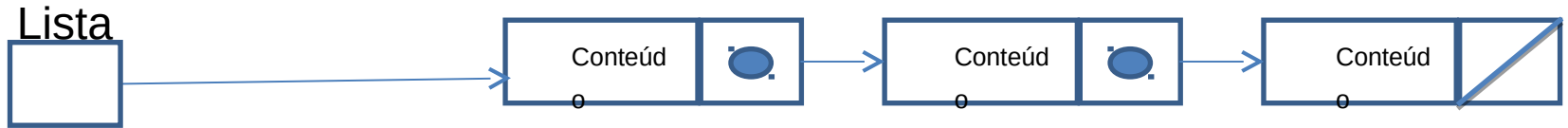
Remoção em uma lista

```
void remove (celula *p) {  
    celula *morta;  
    morta = p->prox;  
    p->prox = morta->prox;  
    free (morta);  
}
```



Remoção em uma lista

```
void remove (celula *p) {  
    celula *morta;  
    morta = p->prox;  
    p->prox = morta->prox;  
    free (morta);  
}
```





Remoção em uma lista

- Não é preciso copiar informações de um lugar para outro, como fizemos para remover um elemento de um vetor: basta mudar o valor de um ponteiro.
- Note também que a função de remoção não precisa conhecer o endereço da lista, ou seja, não precisa saber onde a lista começa.



Listas

- Você pode inventar uma grande variedade de listas encadeadas. Por exemplo, você pode fazer uma lista encadeada *circular*: a última célula aponta para a primeira. A lista pode ou não ter uma célula-cabeça (você decide). Para especificar uma lista circular, basta fornecer um endereço (por exemplo, o endereço da última célula).



Listas

- Outra variedade útil é a lista *duplamente encadeada*: cada célula contém o endereço da célula anterior e o endereço da célula seguinte. A lista pode ou não ter uma célula-cabeça (você decide). A lista pode até ter uma célula-rabo se você achar isso útil.



Prática

- Crie uma lista simplesmente encadeada com as funções:
 - Inserir elementos
 - Remover Elementos
 - Exibir Elementos.



Links - HELP

- <http://www.ic.unicamp.br/~ra069320/PED/MC102/1s2008/Apostilas/Cap10.pdf>
- <http://www.ime.usp.br/~pf/algoritmos/aulas/lista.html>
- <http://www.inf.puc-rio.br/~inf1007/material/slides/listasencadeadas.pdf>
- <http://www.inf.ufsc.br/~ine5384-hp/Capitulo4/EstruturasListaEncadeada.html>
- <http://www.inf.pucrs.br/~pinho/Laprol/Listas/Listas1.htm>
- <https://www.youtube.com/watch?v=5FG7FC-6Fng>
- <https://www.youtube.com/watch?v=YNt47Y1ZAmQ>
- <https://www.youtube.com/watch?v=-xzfTsafixl>
- <https://www.youtube.com/watch?v=Rx8zt5wb1ZU>
- https://www.youtube.com/watch?v=K_o4m5OOOp2o