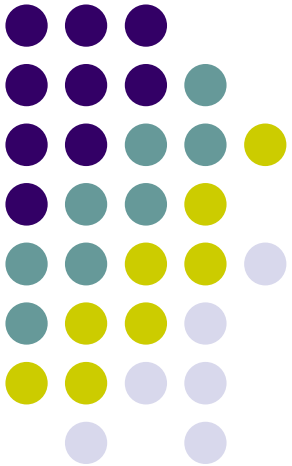
# Algoritmos e Estruturas de Dados

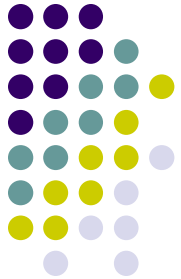
## Listas



PUC Minas

**Profa. Joyce C. P. Carvalho**
**joyce@pucminas.br**

# Estruturas de dados: Listas

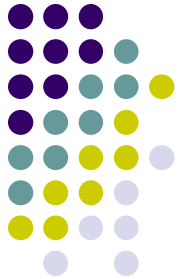


- Lista: Estrutura de dados básica para interligação de elementos

- Sequência de zero ou mais itens $x_1$, $x_2$, $x_3$, …, $x_n$

- Podem crescer ou diminuir de tamanho durante a execução de um programa, de acordo com a demanda.

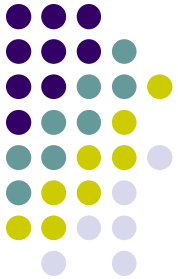
# Listas



- Aplicação: situações nas quais é difícil prever a demanda por memória para um conjunto de itens.

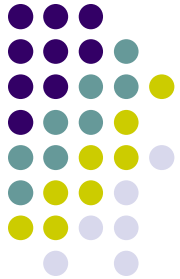
- Exemplos:
  - Controle de processos em um sistema operacional
  - Índice de palavras de um documento

# Listas



- Operações mais comuns:
  - Inserir elementos
  - Retirar elementos
  - Localizar elementos
  - Concatenação de listas

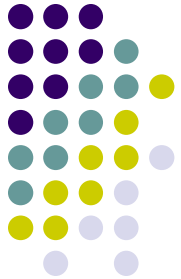
# Definição de uma classe Lista



- Processo de abstração: o conjunto final de operações depende da aplicação. Sugestão:

1. Criar uma lista vazia
2. Inserir um novo item imediatamente após o *i-ésimo* item
3. Retirar o item da posição *i* ou o item de valor *v*
4. Localizar e examinar o *i-ésimo* elemento ou elemento *v*
5. Concatenar duas ou mais listas em uma lista única
6. Verificar se a lista está vazia
7. Imprimir toda a lista

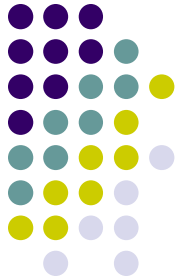
# Implementações de listas lineares



- Várias estruturas de dados podem ser usadas para representar listas lineares, cada uma com vantagens e desvantagens particulares.

- Representações mais utilizadas:
  - Vetores
  - Apontadores

# Implementação de listas por meio de vetores

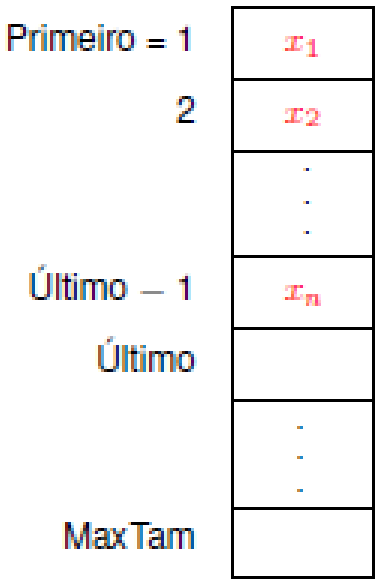


- Os itens da lista são armazenados em posições contíguas de memória.
- A lista pode ser percorrida em qualquer direção.
- A inserção de um novo item pode ser realizada após o último item com custo constante.
- A inserção de um novo item no meio da lista requer um deslocamento de todos os itens localizados após o ponto de inserção.

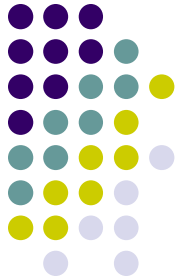
# Implementação de listas por meio de vetores

- Retirar um item do início da lista requer um deslocamento de itens para preencher o espaço deixado vazio.

- Os itens são armazenados em um vetor de tamanho suficiente para armazenar a lista.

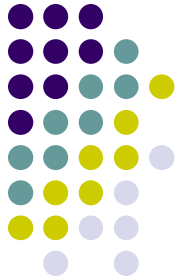


# Listas usando vetores



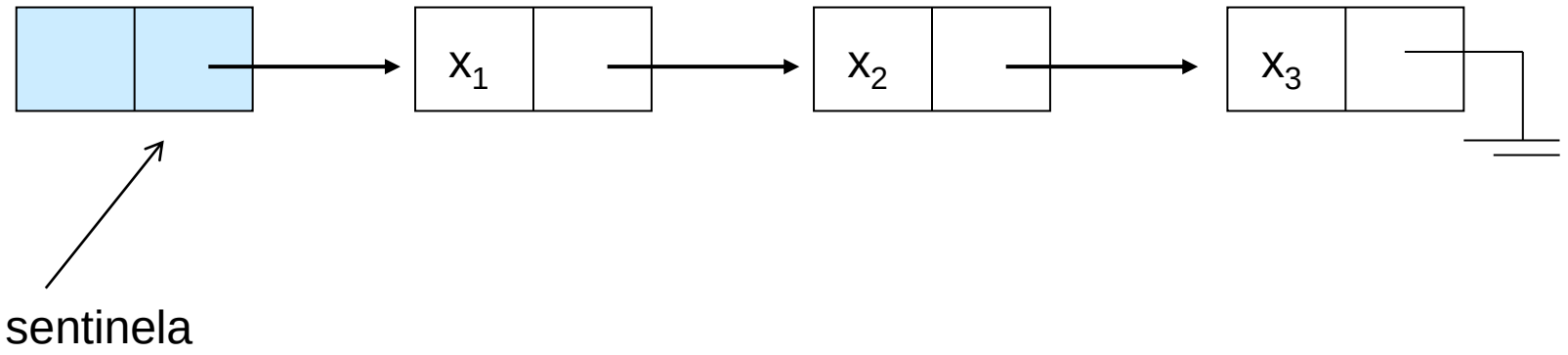
- Vantagem:
  - Economia de memória (os apontadores são implícitos nesta estrutura)
- Desvantagem:
  - Custo para inserir ou retirar itens da lista, que pode causar um deslocamento de todos os itens, no pior caso;

# Listas encadeadas



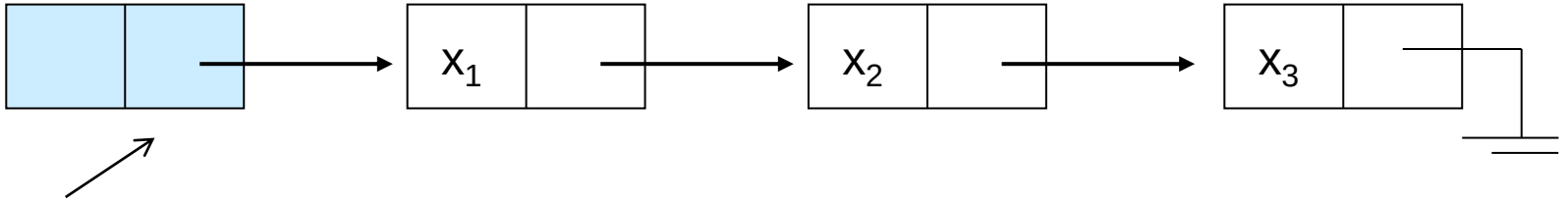
- Implementação de uma lista usando referências
  - Tem tamanho “infinito”: pode-se inserir um novo objeto sempre que necessário
  - A estrutura de um item da lista inclui uma referência para o próximo item
    - Sempre pode-se criar um novo item e apontar para ele

# Listas encadeadas



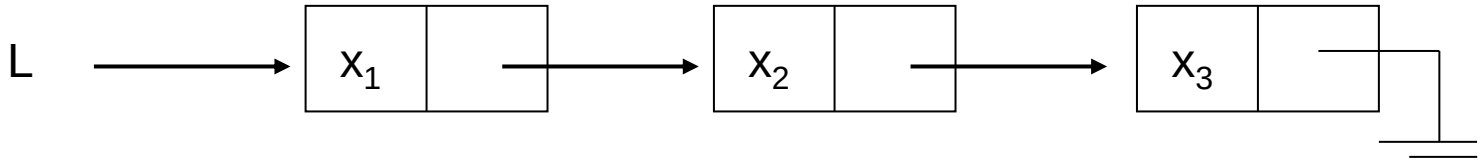
# Listas encadeadas

**Lista encadeada com cabeça**

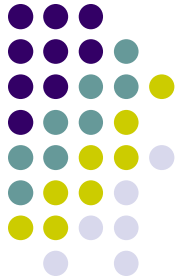



sentinela

**Lista encadeada sem cabeça**



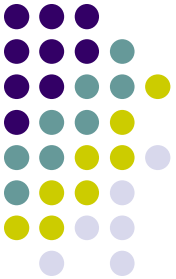
# Listas encadeadas



- **Lista com cabeça**
  - Primeiro elemento da lista não contém informação – elemento sentinela

- **Lista sem cabeça**
  - Não existe elemento sentinela

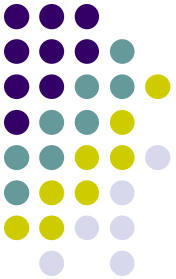
# Listas Encadeadas



- Classe com os dados

```
public class Elemento
{
    public int Num;
    public Elemento Prox;


    public Elemento()
    {
        Num = 0;
        Prox = null;
    }
}
```

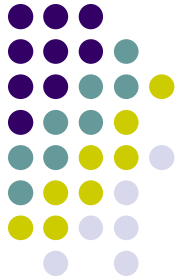
# Listas Encadeadas



- Classe Lista (sem sentinela)

```
public class Lista
{
    private Elemento Início;            // Primeiro elemento da lista
    private Elemento Fim;               // Último elemento da lista


    public Lista()                      // Construtor da Classe
    {
        Início = null;
        Fim = null;
    }
```

# Lista: Operações



- Inserção:
  - Inserir na primeira,
  - na última,
  - na i-ésima posição.

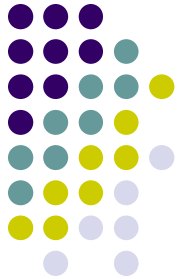
```
public void InserirInício(int Valor)
{
    Elemento Novo = new Elemento();

    Novo.Num = Valor;

    if (Início == null)
    {
        Início = Novo;
        Fim = Novo;

        Novo.Prox = null;
    }
    else
    {
        Novo.Prox = Início;
        Início = Novo;
    }

}
```

# Lista: Operações

- Inserção:
  - Inserir na primeira,
  - na última,
  - na i-ésima posição.

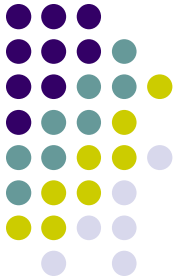
```
public void InserirFinal(int Valor)
{
    Elemento Novo = new Elemento();

    Novo.Num = Valor;

    if (Início == null)
    {
        Início = Novo;
        Fim = Novo;

        Novo.Prox = null;
    }
    else
    {
        Fim.Prox = Novo;
        Fim = Novo;
        Fim.Prox = null;
    }

}
```

# Lista: Operações



- Localizar e examinar i-ésimo elemento:
  - A partir do início, avançar *i* posições na lista
- Retirar da posição *i*:
  - Primeiro, localizar
  - Referências auxiliares
  - Retirar item e atualizar referências

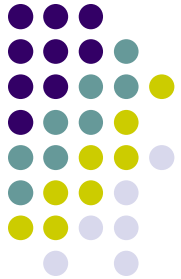
# Lista: Operações



- Concatenar duas listas:
  - Simples: atualizar referências.

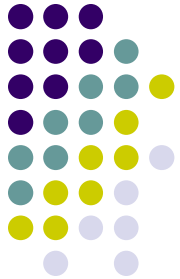
- Imprimir toda a lista:
  - Mais simples ainda: percorrer a lista até o final e imprimir dados relevantes

# Exercício



- Complete a classe Lista (sem sentinela) implementando os seguintes métodos:
  - Verificar se a lista está vazia;
  - Retirar o item da posição *i* (passado como parâmetro);
  - Retirar o item de valor *v* (passado como parâmetro);
  - Retornar o item da posição *i* (passado como parâmetro);
  - Retornar o item de valor *v* (passado como parâmetro);
  - Concatenar duas listas (uma delas é passada como parâmetro);
  - Imprimir toda a lista.

# Listas duplamente encadeadas



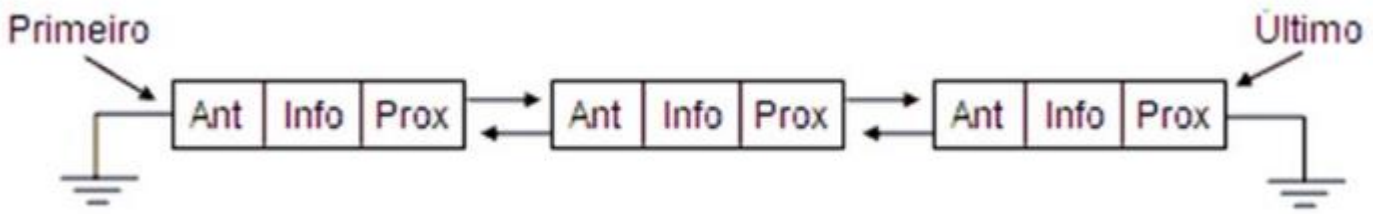
- Apresenta referências para os elementos “anterior” e “próximo”

- Possibilita, assim, que a lista seja percorrida nos dois sentidos

- Permite também acessar os vizinhos do elemento atual.

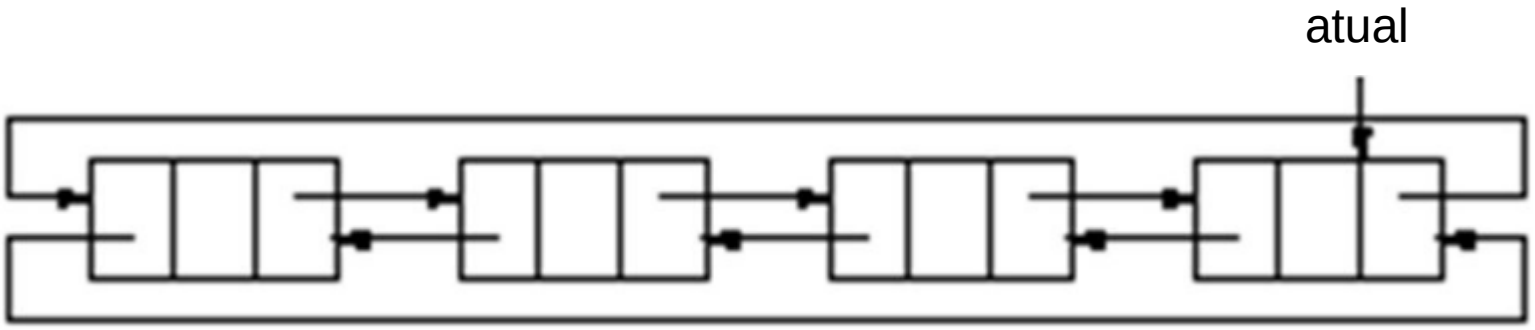
# Listas duplamente encadeadas

- Diferenças nas operações de inserção e remoção:
  - Atualização de referências

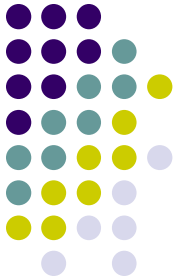


# Lista Circular

- O último elemento aponta novamente para o primeiro
- A lista pode ser totalmente percorrida a partir de qualquer elemento
- Criação do elemento de controle “atual”

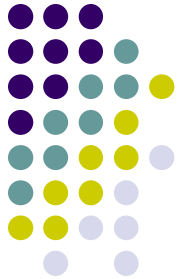


# Lista Circular



- Exemplos de aplicações:
  - *buffer* circular
  - implementação de arquivos
  - gerenciamento e escalonamento de processos
  - reaproveitamento de estruturas

# Exercício



- Implemente o TAD ListaDupla (lista duplamente encadeada, sem célula cabeça), considerando que a célula armazena um dado do tipo inteiro. Ele deve conter todos os métodos do TAD Lista.

# Dúvidas?