Estruturas Encadeadas

Roland Teodorowitsch

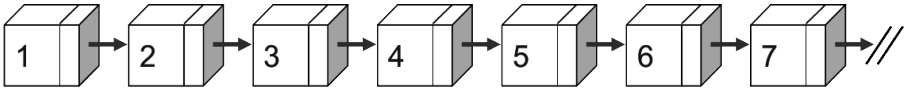
Programação Orientada a Objetos - ECo - Curso de Engenharia de Computação - PUCRS

20 de outubro de 2022

Estruturas Encadeadas

Estruturas Encadeadas

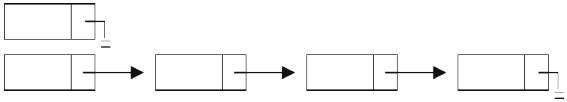
- Costumam ser implementadas usando alocação dinâmica, que é feita durante a execução (aumenta e diminui em tempo de execução)
- Os nodos são ligados entre si para indicar a relação de ordem existente entre os itens



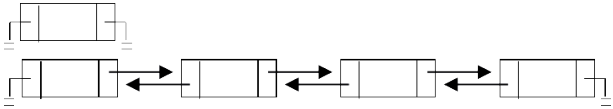
- São úteis quando:
  - Não é possível prever o número de entradas de dados em tempo de compilação
  - A operação que tiver de ser feita sobre essas entradas adequar-se melhor a uma estrutura encadeada

Estruturas Encadeadas

- Consistem de um determinado número de nodos, cada um com uma referência para o próximo nodo
- Duas representações usuais são:
  - Encadeamento simples

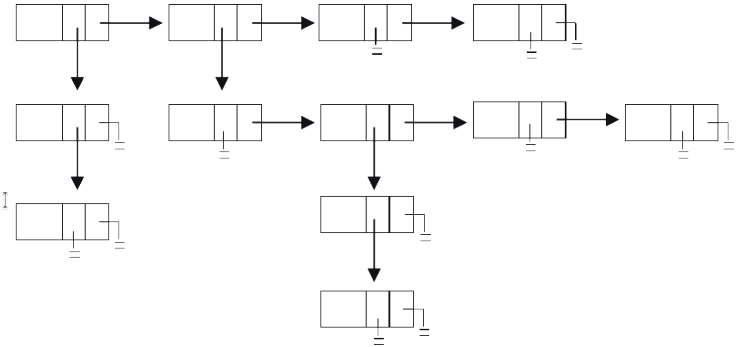


  - Encadeamento duplo



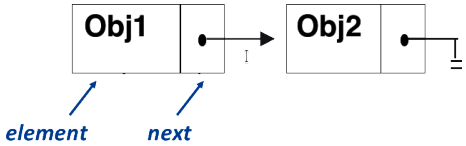
Estruturas Encadeadas

- Eventualmente um nodo pode armazenar mais de uma referência para outros nodos
- Dessa forma é possível criar estruturas de diferentes formatos
- Exemplo:



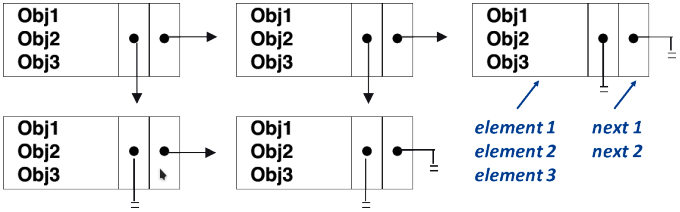
Nodos

- Nodos são conectados (encadeados) por *links*
- Normalmente estes nodos têm dois atributos:
  - ***element*** representando o elemento armazenado no nodo
  - ***next*** representando o endereço do próximo nodo no encadeamento da lista (contém referência para um objeto da mesma classe)



Nodos

- Nodos podem ter mais do que dois atributos
- Exemplo: nodo com cinco atributos (armazena 3 elementos e possui 2 *links*)



Implementação de um Nodo

- Existem diversas opções de implementação
- **Classes internas privadas** são uma opção interessante
- Por exemplo, cria-se uma classe interna na classe que implementa uma lista encadeada
- Pode-se manter os atributos do nodo público sem expor sua implementação para além da implementação da própria lista encadeada
- Quem utiliza a lista não precisa saber que um nodo existe

Implementação de um Nodo

- **Classes internas** são classes declaradas dentro de outras classes
- Objetos da classe interna têm um relacionamento especial com o objeto da classe externa que o cria:
  - É permitido ao objeto da classe interna acessar diretamente todos os atributos e os métodos do objeto da classe externa, não importando se são publicos ou privados

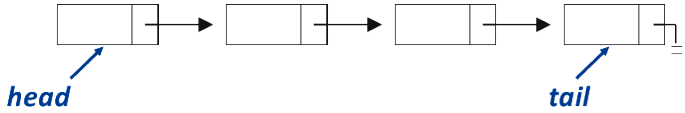
Implementação de um Nodo

- Exemplo: código de uma classe interna Nodo com elementos inteiros

```
class ListaSimplesmenteEncadeada {  // Classe externa

  private:

    class Nodo {  // Classe interna
      public:
        int elemento;
        Nodo *prox;
        Nodo(int e){
            elemento = e;
            prox = nullptr;
        }
    };

    // (...)
```

Armazenamento em Estruturas de Dados Encadeadas

- Para armazenar dados em estruturas encadeadas basta ir encadeando nodos na estrutura à medida que é necessário armazenar novas informações
- É fundamental armazenar uma referência para o início da estrutura encadeada de maneira a não "perder" os dados
  - Acesso direto ao primeiro elemento (*head* ou *prim*) é obrigatório
  - Acesso direto ao último elemento (*tail* ou *ult*) é desejável

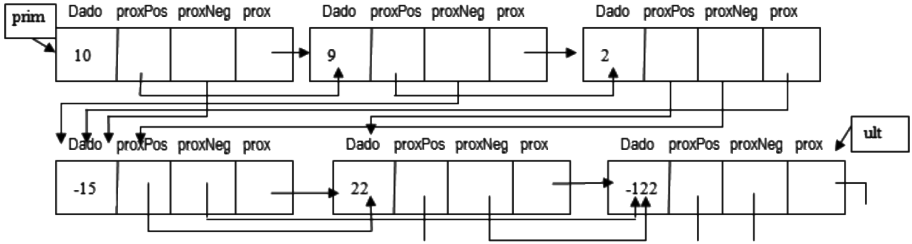


- Nodos que se encontram "nas pontas" da estrutura podem armazenar uma referência nula (`nullptr` em C++). Desta maneira são facilmente identificados

Lista de Exercícios

Lista de Exercícios

1. Implemente uma classe *ListaLinkSinal* que armazena números inteiros. Os nodos desta lista, além do dado e da referência para o próximo nodo, armazenam também uma referência para o próximo valor positivo e para o próximo valor negativo da lista (conforme a figura a seguir). Esta lista deve ter: um método *add* que acrescenta elementos no final da lista; um método *soPositivos* que retorna uma cadeia de caracteres com todos os valores positivos armazenados na lista; e um método *soNegativos* que retorna uma cadeia de caracteres com todos os valores negativos armazenados na lista. Escreva um exemplo de uso que utilize os métodos criados.



Lista de Exercícios

2. Dada a classe *Pessoa*, que se encontra no Material de Apoio. Implemente a classe *ListaEncadeada* que define uma lista encadeada com os métodos descritos abaixo.
   a) `bool estaVazia()`
   Informa se a lista está vazia (devolve `true`) ou não (devolve `false`).
   b) `void inserePessoaNoInicio(string nome, string endereco)`
   Crie um método que coloque um elemento no início da lista.
   c) `void inserePessoaNoFinal(string nome, string endereco)`
   Crie um método que coloque um elemento no final da lista.
   d) `Pessoa *retornaUltimo()`
   Retorna um apontador para o último elemento da lista. Caso a lista esteja vazia, o método deve devolver `nullptr`.
   e) `Pessoa *buscaPessoa(string nome)`
   Crie um método que devolva um ponteiro para um certo elemento da lista. Como parâmetro este método deve receber o nome de uma pessoa a ser buscada na lista. Caso o nome não exista na lista, o método deve devolver `nullptr`.

Lista de Exercícios

2. (Continuação)
   f) `Pessoa *buscaAnterior(string nome)`
   Crie um método que devolva um ponteiro para o nodo da lista que encontra-se ANTES do nodo que contém um certo elemento da lista. Como parâmetro este método deve receber o nome de uma pessoa a ser buscada na lista. Caso o nome não exista na lista ou caso o nome fornecido seja o primeiro da lista, o método deve devolver `nullptr`.
   g) `Pessoa *retira(string nome, string endereco)`
   Crie um método que retire um elemento da lista. Como parâmetro este método deve receber o nome de uma pessoa a ser buscada na lista. Caso o nome não exista na lista, o método deve devolver `nullptr`. Lembre-se de fazer um teste específico para remover o primeiro elemento da lista. O método NÃO deve "deletar" o nodo da memória. Dica: use o método `buscaAnterior`.
   h) `Pessoa *retiraDoInicio()`
   Crie um método que remova um primeiro elemento da lista. O método deve retornar um apontador para o elemento retirado da lista. O método NÃO deve "deletar" o nodo da memória. Caso a lista esteja vazia, o método deve devolver `nullptr`.

Lista de Exercícios

2. (Continuação)
   i) `Pessoa *retiraDoFinal()`
   Crie um método que remova o último elemento da lista. O método deve retornar um apontador para o elemento retirado da lista. O método NÃO deve "deletar" o nodo da memória. Caso a lista esteja vazia, o método deve devolver `nullptr`.
   j) `int remove(string nome, string endereco)`
   Crie um método que remova um elemento da lista. O método deve "deletar" o nodo da memória. Dica: use o método `retira` .
   k) `void inserePessoaEmOrdem(string nome, string endereco)`
   Crie um método que coloque um elemento na lista, de forma que os elementos fiquem sempre em ordem crescente do nome da pessoa. Dica: Modifique o método de busca de forma que ele devolva um apontador para o primeiro nodo da lista no qual o atributo 'nome' seja "maior" que no nome a ser inserido na lista.

Créditos

Créditos

- Estas lâminas contêm trechos de materiais disponibilizados pelos professores Rafael Garibotti, Bernardo Copstein e Márcio Sarroglia Pinho (exercício 2).