Fatec
São Caetano do Sul
Antônio Russo
.
TADS - Análise Desenvolvimento de Sistemas

Estruturas de Dados

Aula #02
Abstração de Dados

Prof.: Veríssimo - carlos.pereira70@fatec.sp.gov.br

27/02/2024

Sobre este documento

Este documento objetiva orientar o aluno quanto ao conteúdo a ser desenvolvido nesta disciplina, bem como a divulgação dos critérios de avaliação que serão empregados.

É muito importante que o aluno faça uma acurada leitura deste documento, pois nele conterá a descrição dos elementos que norteiam o curso.

Contents

Chapter 1

Abstração e Tipos de Dados

1.1 Introdução

Compreender a complexidade dos dados e suas manipulações é essencial para construir sistemas eficientes e escaláveis. Nesta aula exploraremos a essência da programação, e para isto abordaremos os conceitos de abstração de dados e tipos de dados.

Abstração de dados é o conceito fundamental que nos permite encapsular dados e operações relacionadas em uma única entidade, escondendo os detalhes de implementação e fornecendo uma interface clara e distinta para interação. É a arte de criar modelos conceituais que representam entidades do mundo real de forma simplificada e manipulável, permitindo-nos lidar com a complexidade de maneira mais gerenciável.

Quanto aos tipos de dados, estes constituem a base sobre a qual toda a estruturação e manipulação de informações em um programa são construídas. Desde os **tipos primitivos simples**, como **inteiros** e **caracteres**, até os **tipos compostos** mais elaborados, como **arrays** e **classes**, os tipos de dados

formam o alicerce sobre o qual construímos nossos algoritmos e estruturas de dados.

Nesta aula, vamos explorar a relação íntima entre abstração de dados e tipos de dados,entendendo como representamos e manipulamos informações em um ambiente de programação. Ao entender esses conceitos fundamentais, estaremos melhor equipados para criar **sistemas robustos**, **modulares** e de **fácil manutenção**.

1.2 Coneitos Fundamentais

1.2.1 Estrutura de Dados

Uma **estrutura de dados** pode ser definida como uma maneira de organizar e armazenar dados de forma eficiente, facilitando sua manipulação e acesso. De acordo com *Robert Lafore*, em seu livro "Data Structures and Algorithms in Java", uma estrutura de dados especifica não apenas a representação dos dados, mas também as operações que podem ser realizadas sobre esses dados.

Por Exemplo:

- Um exemplo comum de estrutura de dados é o **array**, que organiza os elementos em uma sequência contígua de memória, permitindo acesso rápido aos elementos por meio de índices.

1.2.2 . Importância das Estruturas de Dados na programação

As estruturas de dados desempenham um papel crucial na programação e na resolução de problemas computacionais, pois oferecem maneiras eficientes de organizar e manipular dados.

Segundo Thomas H. Cormen, autor de "Introduction to Algorithms", escolher a estrutura de dados correta pode ter um grande impacto no desempenho e na eficiência do algoritmo que estamos desenvolvendo.

Por Exemplo:

- Considere o problema de buscar um elemento em uma lista. Se usarmos uma lista desordenada, a busca exigirá percorrer todos os elementos até encontrar o desejado, resultando em uma complexidade de tempo de O(n). No entanto, se utilizarmos uma árvore de busca binária, podemos reduzir a complexidade de tempo para O(log n), o que é muito mais eficiente, especialmente para grandes conjuntos de dados.

Portanto, compreender e saber escolher as estruturas de dados apropriadas é fundamental para escrever código eficiente e resolver problemas computacionais de forma otimizada.

1.3 Abstração de Dados

Abstração de dados é um conceito fundamental na ciência da computação que se refere à **criação de modelos conceituais** que representam **entidades do mundo real** de uma forma simplificada e manipulável. Segundo *Donald Knuth*, em sua obra "*The Art of Computer Programming*", a abstração de dados envolve a definição de um conjunto de operações bem definidas que podem ser realizadas sobre os dados, **sem expor os detalhes internos de sua implementação**. Por exemplo:

- - Considere uma **pilha**, que é uma estrutura de dados na qual os elementos são adicionados e removidos de acordo com o princípio "último a entrar, primeiro a sair"

(**LIFO**). Ao abstrair uma pilha, podemos pensar nela apenas em termos de suas operações básicas, como **push** (inserir um elemento), **pop** (remover o último elemento adicionado) e **top** (obter o elemento no topo da pilha), **sem precisar saber como essas operações são implementadas internamente**.

1.3.1 Separando a interface da implementação

Outro aspecto importante da abstração de dados é a **separação clara entre a interface pública** , que define as operações disponíveis para interação externa, e a **implementação privada**, que trata dos detalhes internos da estrutura de dados. Como mencionado por Grady Booch, em "Object-Oriented Analysis and Design with Applications", essa separação permite que os usuários da estrutura de dados usem apenas as operações especificadas na interface, sem precisar conhecer ou depender da implementação subjacente.

Exemplo:

- Para uma **pilha**, a interface pública pode incluir as operações push, pop e top, enquanto a implementação privada pode envolver o uso de um array ou lista encadeada para armazenar os elementos e métodos para manipular esses elementos de acordo com as operações definidas.
-

1.4 Tipos de Dados

1.4.1 Primitivos

Os tipos de dados primitivos são aqueles que **representam valores simples** e **individuais**, geralmente suportados diretamente pelo hardware ou linguagem de programação. Eles incluem tipos como **inteiros, números de ponto flutuante, caracteres e booleanos**. Segundo Herbert Schildt, em "C: The Complete Reference", esses tipos são essenciais para realizar operações básicas e são frequentemente usados como blocos de construção para tipos de dados mais complexos.

1.4.2 Abstratos

Por outro lado, os tipos de dados abstratos (TDA) são aqueles definidos pelo programador para representar um **conceito específico, encapsulando tanto os dados quanto as operações que podem ser realizadas sobre esses dados**. Como mencionado por Grady Booch, em "Object-Oriented Analysis and Design with Applications", os TDAs fornecem uma maneira de modelar entidades do mundo real de forma mais precisa e são fundamentais para a construção de software modular e escalável.

Tipos de dados básicos: inteiros, reais, caracteres

- Inteiros

  - Representam números inteiros sem parte fracionária. Em linguagens como C, podem ser representados por tipos como int e long. Eles são amplamente utilizados para contar, indexar e realizar operações aritméticas. (Referência: Herbert Schildt, "C: The Complete Reference").

•

- Reais

- Representam números com parte fracionária, geralmente implementados usando tipos como float e double em linguagens como C. Eles são usados em situações que exigem precisão decimal, como cálculos financeiros e científicos.

•

-Caracteres

- Representam caracteres individuais, como letras, números e símbolos. Em C, são representados pelo tipo char e são usados em operações de entrada/saída, manipulação de strings e representação de dados de texto.

•

Tipos de dados compostos: arrays, estruturas, classes

- Arrays

- Arrays são coleções homogêneas de elementos do mesmo tipo de dados, acessados por um índice. Eles são amplamente utilizados para armazenar e manipular conjuntos de dados relacionados. Em C, por exemplo, podemos declarar um array de inteiros usando a sintaxe ‘int array[10]‘. (Referência: Herbert Schildt, "C: The Complete Reference").

- Estruturas

- Estruturas (ou structs) permitem agrupar diferentes tipos de dados relacionados em uma única unidade. Elas são úteis para representar entidades complexas que consistem em múltiplos campos com propriedades diferentes.

Por exemplo, podemos definir uma estrutura em C para representar um aluno com campos como nome, idade e número de matrícula.

- Classes

- Classes são tipos de dados abstratos em linguagens de programação orientadas a objetos, como C++. Elas encapsulam dados e comportamentos relacionados em uma única unidade e permitem a criação de instâncias (objetos) dessas classes. Classes são a base da programação orientada a objetos e fornecem recursos como encapsulamento, herança e polimorfismo. (Referência: Grady Booch, "Object-Oriented Analysis and Design with Applications").