Tipo Abstrato de Dados (TAD)

Algoritmos e Estrutura de Dados 1
Prof.: Luiz Gustavo Almeida Martins

Introdução

- Na construção de bons modelos computacionais é necessário expressar os detalhes do problema de dados adequados e desenvolver um algoritmo adequado que atue sobre essa estrutura.

Programas = Estruturas de Dados + Algoritmos

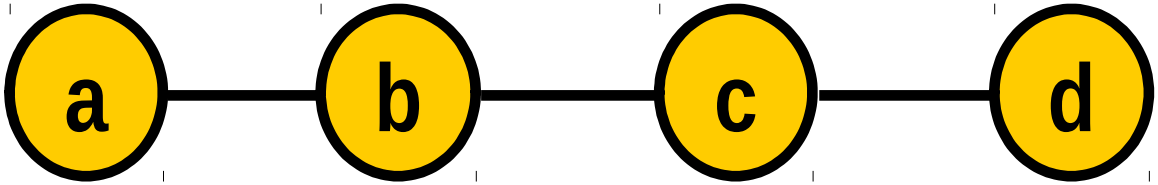
Estrutura de Dados

- Estruturação *conceitual* dos dados

- Reflete um **relacionamento lógico** entre dados, de acordo com o problema considerado

Exemplo de Estrutura de Dados

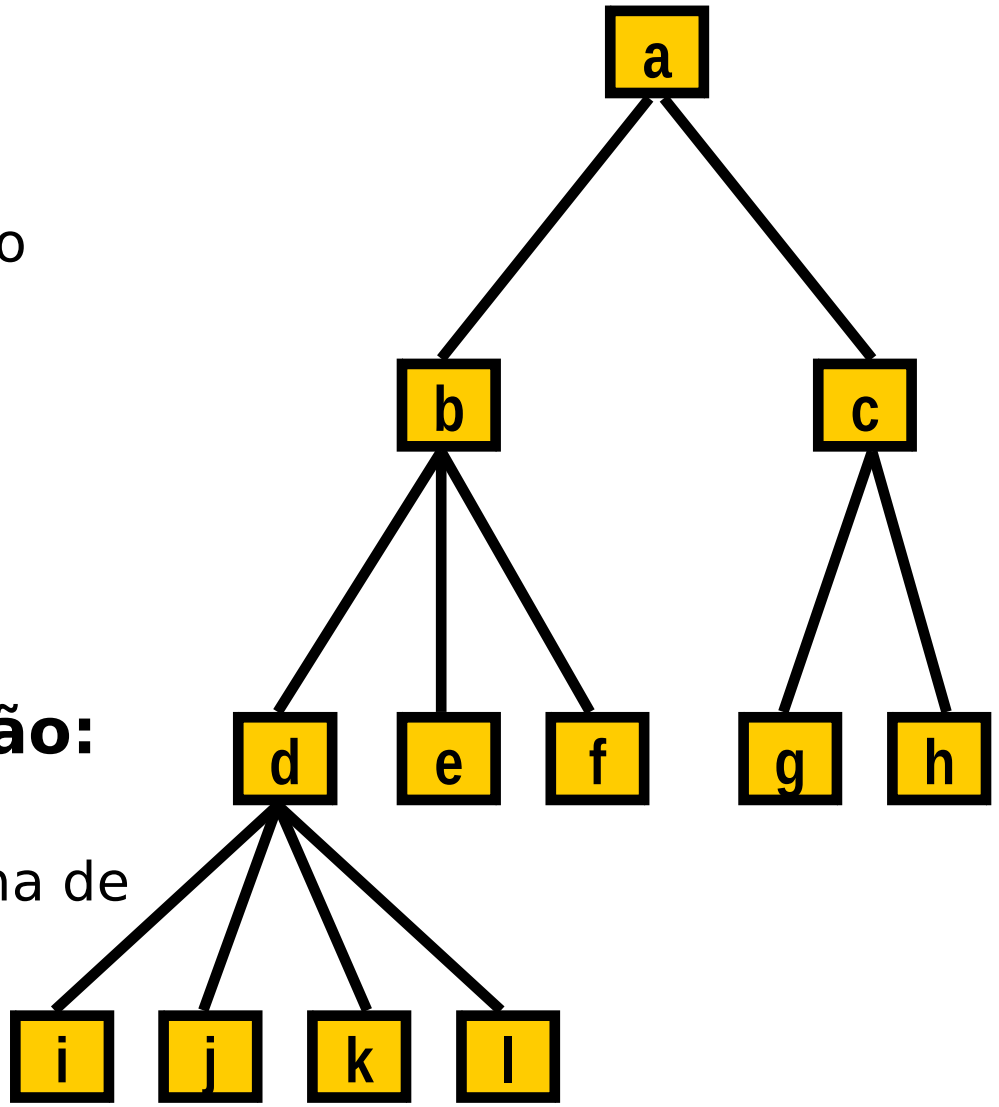
- Listas:
  - Estrutura linear - seqüencial
  - Relação de ordem entre os dados



- **Exemplo de aplicação:**
  - **Domínio:** empresa
  - **Problema:** dados dos funcionários

Exemplo de Estrutura de Dados

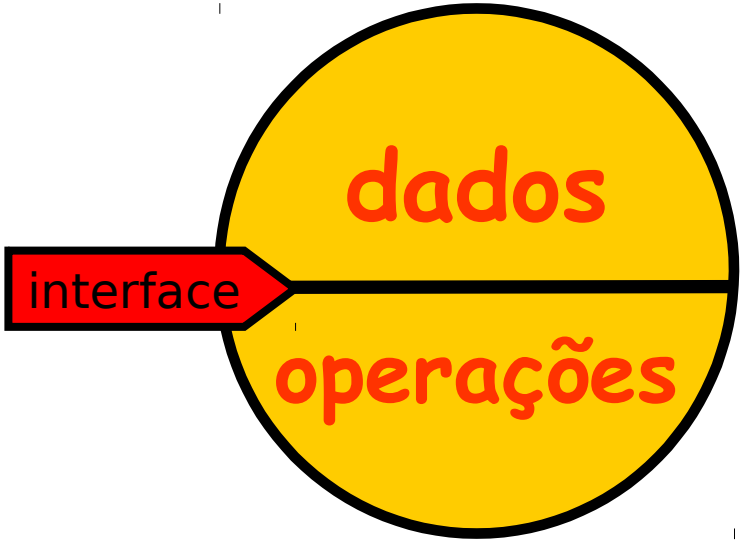
- Árvore:
  - Estrutura espacial
  - Relação de subordinação entre os dados



- **Exemplo de aplicação:**
  - **Domínio:** empresa
  - **Problema:** organograma de funções

Tipos Abstratos de Dados (TAD)

- Um **TAD** é uma forma de definir um **novo tipo** de dado juntamente com as **operações** que o manipulam.



Tipos Abstratos de Dados (TAD)

- Técnica de programação baseada na definição de tipos estruturados.

- **Idéia central:** encapsular (esconder) de quem usa o TAD a forma como ele foi efetivamente implementado.
  - Visibilidade da estrutura interna do tipo fica limitada às operações.
  - Cliente tem acesso somente à forma abstrata do TAD

Tipos Abstratos de Dados (TAD)

- Um TAD é definido por:
  - Um **conjunto de valores** (dados).
  - Um **conjunto de operações** que atuam sobre esses valores. Elas devem ser consistentes com os tipos utilizados para os valores.

Tipos de Dados

- Define o conjunto de valores (domínio) que uma variável pode assumir.
  - **Exemplo:**

    Inteiro

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

    Lógico

    < verdadeiro, falso >

Tipos de Dados

- Basicamente, existem 2 tipos de dados:

  - **Tipos primitivos:**
    - Tipos básicos que representam valores indivisíveis.
    - Fornecidos pela linguagem de programação

      Ex: int, float, char, long int, etc.

  - **Tipos estruturados:**
    - Definem uma coleção de tipos simples ou um agregado de tipos diferentes.

      Ex: vetores (*int [10]*), strings de caracteres (*char**),

      estruturas (*struct aluno {*
      int num-matricula;
      char nome[100];
      })

Operações

- Determinam as **ações** realizadas na manipulação dos dados do TAD.

- **Exemplo de 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

Tipos Abstratos de Dados (TAD)

- Permite a separação entre o **conceito** (o que fazer) e a **implementação** (como fazer).
  - Somente o conceito do TAD é visível externamente.

- Mudança na implementação do TAD não afeta o programa que o usa (programa cliente).
  - Desde que sejam mantidas as interfaces E/S

Especificação de um TAD

- Usada para definir a **parte conceitual** do TAD.

- Deve conter as seguintes informações:

Cabeçalho:

- **Nome:** identificação do TAD
- **Dados:** descrição dos tipos dos dados da estrutura
- **Lista de operações** (apenas nome)

Especificações de cada operação:

- **Entradas:** informação necessária para executar a operação.
- **Pré-condição:** verificada antes de executar a operação
- **Processo:** tarefas que devem ser realizadas para atingir o objetivo da operação.
- **Saída:** valor resultante do processamento e que é **retornado explicitamente** ao seu final.
- **Pós-condição:** indica quaisquer alterações que a operação causou na estrutura (**retorno implícito**).

Estrutura Geral da Especificação

TAD **nome_TAD** é:
Dados: descrição do tipo de dados
Lista de operações: operação1, operação2, ..., operação *N*

Operações:
Operação1:
Entrada:
Pré-condição:
Processo:
Saída:
Pós-condição:
Operação2:
Entrada:
Pré-condição:
Processo:
Saída:
Pós-condição:
...
Operação *N* :
Entrada:
Pré-condição:
Processo:
Saída:
Pós-condição:

Exercício

- Especificar o TAD números racionais (números expressos como quocientes de **2 inteiros**), considerando as operações **multiplica** e **soma**.
  - **Ex:** ½, ¾

TAD Racionais

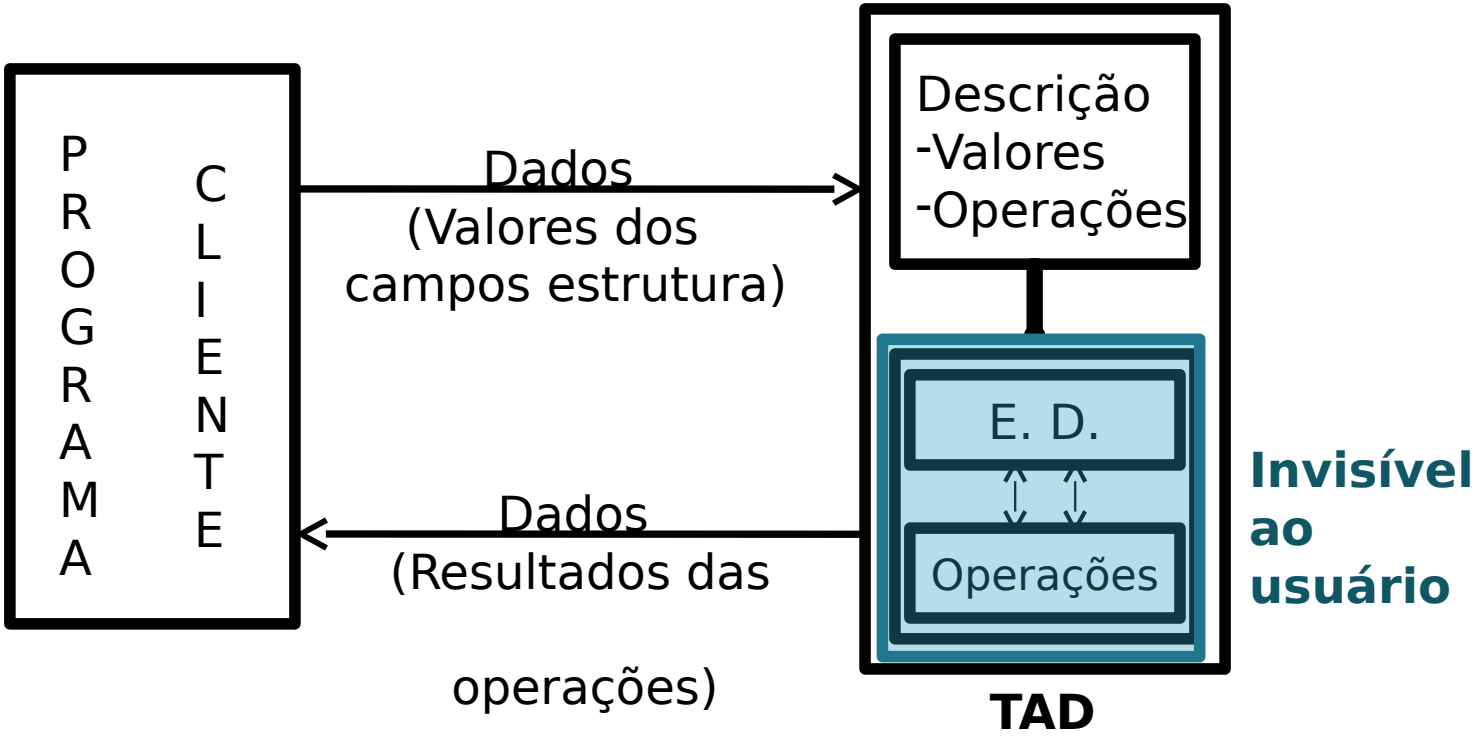
Implementação de um TAD

- Define **como** o TAD deve realizar suas operações.

- TAD e aplicação devem ser implementados em módulos separados.
  - Garante a independência de utilização

- Códigos são compilados separadamente.
  - Mudança na implementação força a recompilação apenas do módulo envolvido.
  - Códigos são ligados na linkedição.

Iteração entre TAD e Programa Usuário



Implementação de um TAD

- Em C:
  - Pelo menos um arquivo de aplicação.

    Ex: principal.c
  - Um arquivo com a implementação do TAD
    - Contém a estrutura e a implementação das funções

      Ex: racional.c
  - Um arquivo cabeçalho (extensão .h) com as declarações de tipo e os protótipos das funções.
    - Permite o reconhecimento por outros módulos.

      Ex: racional.h

Implementação de um TAD

- O programa aplicativo deve incluir a diretiva ***#include “racional.h”*** para que ele possa conhecer os protótipos das funções do TAD.

- Diferença entre **<>** e **“”** na diretiva ***include***:

  - **#include<arq.h>:** significa que os arquivos “.h” estão no diretório padrão de include.
    - Usado para os arquivos de cabeçalho (.h) das funções da biblioteca padrão do C.

  - **#include “arq.h”:** significa que o arquivo “.h” está no diretório corrente (de trabalho), junto com o próprio arquivo “.c”.

Geração da Aplicação

- No Windows:
  - *Frameworks* de desenvolvimento associam os arquivos de uma aplicação através de um projeto.
  - **Ex:** CodeBlocks e DevC

- No Linux:
  - Pode-se usar o projeto (ex: CodeBlocks) ou tratar os programas isoladamente e juntá-los somente na linkedição.
  - **Ex:** compilação e linkedição no gcc

    gcc -c racional.c Gera o arquivo objeto racional.o

    gcc -c principal.c Gera o arquivo objeto principal.o

    gcc -o principal principal.o racional.o

    Gera o arquivo executável

Exercícios

1. Implementar o TAD racional e um programa aplicativo que a utilize.

2. Especificar e implementar o TAD Ponto, para representar um ponto no espaço bidimensional (coordenadas x e y no espaço $\mathbb{R}^2$) e com as seguintes funções
   - **Cria_pto:** cria um ponto a partir de suas cordenadas x e y.
   - **Libera_pto:** operação que libera a posição de um ponto.
   - **Distancia_pto:** calcula a distância entre dois pontos.

   O programa aplicativo deve ler as coordenadas de 2 pontos, digitadas pelo usuário e imprimir na tela a distância entre esses pontos. Para isso, o programa aplicativo deve chamar a função cria p/ criar os 2 pontos, deve chamar a função para calcular a distância e depois liberar a memória dos dois pontos.

Exemplo da Especifição Formal

TAD **Ponto**

Dados: 2 números reais (cordenadas x e y).

Lista de operações: cria_pto, libera_pto e distancia.

Cria_pto:

Entradas: dois valores reais (coordenadas x e y)
Pré-condição: nenhuma
Processo: criar um novo ponto e atribuir suas coordenadas
Saída: novo ponto
Pós-condição: nenhuma

Vantagens do TAD

- **Reuso:**
  - Possibilidade de utilização do mesmo TAD em diversas aplicações diferentes.

- **Independência da implementação:**
  - Possibilidade de alterar o TAD sem alterar as aplicações que o utilizam.

- **Segurança**:
  - Aplicação cliente não pode alterar a representação nem tornar os dados inconsistentes.