

Tipos de Dados, Tipos Abstratos de Dados Estruturas de Dados

Tipo de dados, tipo abstrato de dados, estruturas de dados

- Termos parecidos, mas com significados diferentes

Tipo de dado

- Em linguagens de programação o tipo de dado de uma variável, constante ou função define o conjunto de valores que a variável, constante ou função podem assumir
 - p.ex., variável *boolean* pode assumir valores *true* ou *false*
- Programador pode definir novos tipos de dados em termos de outros já definidos
 - Tipos estruturados, p.ex., *arrays*, *records*

Estrutura de Dados

- Um tipo estruturado é um exemplo de estrutura de dados
 - Tipos estruturados são estruturas de dados já pré-definidas na linguagem de programação
 - O programador pode definir outras estruturas de dados para armazenar as informações que seu programa precisa manipular
 - Vetores, registros, listas encadeadas, pilhas, filas, árvores, grafos, são exemplos de estruturas de dados típicas utilizadas para armazenar informação em memória principal

Perspectivas para Tipos de Dados

- Tipos de dados podem ser vistos como métodos para interpretar o conteúdo da memória do computador
- Mas, podemos interpretar o conceito de tipo de dado sob outra perspectiva
 - não em termos do que um computador pode fazer (interpretar os bits...), mas em termos do que os usuários (programadores) desejam fazer (p.ex., somar dois inteiros...)
 - O programador não se importa muito com a representação no hardware, mas sim com o conceito matemático de inteiro
 - Um tipo inteiro 'suporta' certas operações...




Tipo Abstrato de Dados (TAD)

- Os tipos e estruturas de dados existem para serem usados pelo programa para acessar informações neles armazenadas, por meio de operações apropriadas
 - Do ponto de vista do programador, muitas vezes é conveniente pensar nas estruturas de dados em termos das operações que elas suportam, e não da maneira como elas são implementadas
 - Uma estrutura de dados definida dessa forma é chamada de um **Tipo Abstrato de Dados (TAD)**
-

Tipo Abstrato de Dados (TAD)

- TAD, portanto, estabelece o conceito de tipo de dado divorciado da sua representação
- Definido como um modelo matemático por meio de um par (v, o) em que
 - v é um conjunto de valores
 - o é um conjunto de operações sobre esses valores
 - Ex.: tipo *real*
 - $v = \mathbb{R}$
 - $o = \{+, -, *, /, =, <, >, <=, >=\}$

Tipo abstrato de dados

mundos real	dados de interesse	ESTRUTURA de armazenamento	possíveis OPERAÇÕES
 pessoa	<ul style="list-style-type: none"> a idade da pessoa 	<ul style="list-style-type: none"> tipo inteiro 	<ul style="list-style-type: none"> nasce ($i \leftarrow 0$) aniversário ($i \leftarrow i + 1$)
 cadastro de funcionários	<ul style="list-style-type: none"> o nome, cargo e o salário de cada funcionário 	<ul style="list-style-type: none"> tipo lista ordenada 	<ul style="list-style-type: none"> entra na lista sai da lista altera o cargo altera o salário
 fila de espera	<ul style="list-style-type: none"> nome de cada pessoa e sua posição na fila 	<ul style="list-style-type: none"> tipo fila 	<ul style="list-style-type: none"> sai da fila (o primeiro) entra na fila (no fim)

Definição de TAD

- Requer que operações sejam definidas sobre os dados sem estarem atreladas a uma representação específica
 - ocultamento de informação (*information hiding*)
- Programador que usa um tipo de dado *real*, *integer*, *array* não precisa saber como tais valores são representados internamente
 - mesmo princípio pode ser aplicado a listas, pilhas, ...
 - se existe uma implementação disponível de uma lista, p. ex., um programador pode utilizá-la como se fosse uma 'caixa preta', acessá-la por meio das operações que ela suporta

Definição de TAD

- O conceito de TAD é suportado por algumas linguagens de programação procedimentais
 - Ex. Modula 2, Ada
- Para definir um TAD
 - programador descreve o TAD em dois módulos separados
 - Um módulo contém a definição do TAD: representação da estrutura de dados e implementação de cada operação suportada
 - Outro módulo contém a interface de acesso: apresenta as operações possíveis
 - Outros programadores podem, por meio da interface de acesso, usar o TAD sem conhecer os detalhes representacionais e sem acessar o módulo de definição

Definição de TAD

- Os módulos (ou units) são instalados em uma biblioteca e podem ser reutilizados por vários programas
 - A execução do programa requer a linkedição dos módulos de definição (que podem ser mantidos já pré-compilados em uma biblioteca) junto com o programa
 - Mas o programador não precisa olhar o código do módulo de definição para usar o TAD!
 - Basta conhecer a interface de acesso

Implementação de um TAD

- Uma vez definido um TAD e especificadas as operações associadas, ele pode ser implementado em uma linguagem de programação
- Uma estrutura de dados pode ser vista, então, como uma implementação de um TAD
 - implementação do TAD implica na escolha de uma ED para representá-lo, a qual é acessada pelas operações que ele define
- ED é construída a partir dos tipos básicos (*integer*, *real*, *char*) ou dos tipos estruturados (*array*, *record*) de uma linguagem de programação

Características de um TAD

- Característica essencial de TAD é a separação entre a definição conceitual – par (v, o) – e a implementação (ED específica)
 - ❑ O programa só acessa o TAD por meio de suas operações, a ED **nunca** é acessada diretamente
 - ❑ "ocultamento de informação"

Características de um TAD

- Programador tem acesso a uma descrição dos valores e operações admitidos pelo TAD
 - Programador não tem acesso à implementação
 - Idealmente, a implementação é 'invisível' e inacessível
 - Ex. pode criar uma lista de clientes e aplicar operações sobre ela, mas não sabe como ela é representada internamente
 - Quais as vantagens?
-

Vantagens do uso de TADs

- Reuso: uma vez definido, implementado e testado, o TAD pode ser acessado por diferentes programas
- Manutenção: mudanças na implementação do TAD não afetam o código fonte dos programas que o utilizam (decorrência do ocultamento de informação)
 - módulos do TAD são compilados separadamente
 - uma alteração força somente a recompilação do arquivo envolvido e uma nova link-edição do programa que acessa o TAD
 - O programa mesmo não precisa ser recompilado!
- Correção: TAD foi testado e funciona corretamente

Modularização em C

- Programa em C pode ser dividido em vários arquivos
 - Arquivos **fonte** com extensão **.c**
 - Denominados de módulos
- Cada módulo deve ser compilado separadamente
 - Para tanto, usa-se um **compilador**
 - Resultado: **arquivos objeto** não executáveis
 - Arquivos em linguagem de máquina com extensão **.o** ou **.obj**
- Arquivos objeto devem ser juntados em um **executável**
 - Para tanto, usa-se um *ligador* ou **link-editor**
 - Resultado: um único arquivo em linguagem de máquina
 - Usualmente com extensão **.exe**