TAD: Tipo Abstrato de Dado

BCC202- Estrutura de Dados
DECOM-UFOP
Profª. ASN

Material elaborado com base nos *slides* do Prof. Reinaldo Fortes (curso de 2014/01)

Algoritmo

O que é um algoritmo?

Informalmente, um algoritmo é um procedimento computacional bem definido que:

- recebe um conjunto de valores como entrada e
- produz um conjunto de valores como saída.

Equivalentemente, um algoritmo é uma ferramenta para resolver um problema computacional. Este problema define a relação precisa que deve existir entre a entrada e a saída do algoritmo.

Conteúdo

- **Introdução**
  - Algoritmos e Estruturas de Dados
- **Tipo Abstrato de Dado (TAD)**
  - Conceito
  - Motivação
  - Implementação
  - Recaptulando
- **Conclusão**
- **Exercícios**

Exemplos de Problemas: teste de primalidade

Problema: determinar se um dado número é primo.

Exemplo:

Entrada: 9411461

Saída: É primo.

Exemplo:

Entrada: 8411461

Saída: Não é primo.

Conteúdo

- **Introdução**
  - Algoritmos e Estruturas de Dados
- **Tipo Abstrato de Dado (TAD)**
  - Conceito
  - Motivação
  - Implementação
  - Recaptulando
- **Conclusão**
- **Exercícios**

Exemplos de Problemas: ordenação

Definição: um vetor $A[1 \ldots n]$ é crescente se $A[1] \leq \ldots \leq A[n]$.

Problema: rearranjar um vetor $A[1 \ldots n]$ de modo que fique crescente.

Entrada:

1										*n*
33	55	33	44	33	22	11	99	22	55	77

Saída:

1										*n*
11	22	22	33	33	33	44	55	55	77	99

Instância de Problema

Uma instância de um problema é um conjunto de valores que serve de entrada para esse.

Exemplo:
Os números 9411461 e 8411461 são instâncias do problema de primalidade.

Exemplo:
O vetor

1										*n*
33	55	33	44	33	22	11	99	22	55	77

é uma instância do problema de ordenação.

Estrutura de Dados

Uma estrutura de dados pode ser dividida em dois pilares fundamentais: ***dado*** e ***estrutura***

Dado

Elemento que possui valor agregado e que pode ser utilizado para solucionar problemas computacionais. Os dados possuem tipos específicos.

Estrutura

Elemento estrutural que responsável por carregar as informações dentro de uma estrutura de *software*.

Estrutura de Dados

"**Estrutura de dados** é o ramo da computação que estuda os diversos mecanismos de organização de dados para atender aos diferentes requisitos de processamento."

Estrutura de Dados

Uma estrutura de dados pode ser dividida em dois pilares fundamentais: ***dado*** e ***estrutura***

Dado

Tipos de dados:
- Inteiro (int)
- Texto (string)
- Caracter (char)
- Ponto flutuante (float)
- Ponto flutuante (double)

Estrutura

Estruturas:
- Vetores multidimensionais
- Pilhas
- Filas
- Listas

Estrutura de Dados

"**Estrutura de dados** é o ramo da computação que estuda os diversos mecanismos de organização de dados para atender aos diferentes requisitos de processamento."

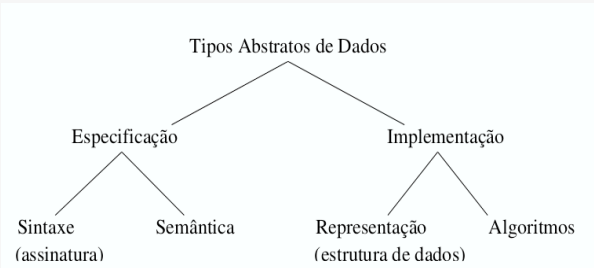
Aplicações

- Implementação de estruturas de banco de dados.
- Compiladores e interpretadores
- Editores de texto
- Kernel de sistemas operacionais.
- …

Conteúdo

- **Introdução**
  - Algoritmos e Estruturas de Dados
- **Tipo Abstrato de Dado (TAD)**
  - Conceito
  - Motivação
  - Implementação
  - Recaptulando
- **Conclusão**
- **Exercícios**

TADs



→ Separação de especificação e implementação: permite o uso do TAD sem conhecer nada sobre a sua implementação.

Portanto, um TAD pode ter mais de uma implementação.

TADs: Tipos Abstratos de Dados

- ***Abstração***: "é a habilidade de concentrar nos aspectos essenciais de um contexto qualquer, ignorando características menos importantes ou acidentais."
- Quando definimos um TAD (Tipo ***Abstrato*** de Dados), nos concentramos nos aspectos essenciais do tipo de dado (operações) e nos abstraímos de como ele foi implementado.

TADs

- A **sintaxe** de um TAD é conhecida como um conjunto de assinaturas de operações que especifica a sua interface.
- A especificação sintática de um TAD pode não ser suficiente para descrever o comportamento.
  - Especificar a sua semântica de maneira independente da implementação.

TADs: Tipos Abstratos de Dados

O "*Tipo Abstrato de Dado (TAD)"* é uma especificação de um conjunto de dados e operações que podem ser executadas sobre esses dados.

Representação Semântica de um TAD

- **Especificação Algébrica**: consiste na definição de um conjunto de equações que interralacionam as operações do TAD.
- **Predicados**: são pré-condições, pós-condições e invariantes aplicadas às operações do TAD que são representadas utilizando expressões lógicas.

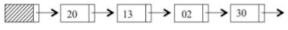
TAD: Lista de Números Inteiros

- Insere um número no começo da lista.

Implementação por Vetor:


```
1 void Insere(int x, Lista L) {
2   for(i=0;...) {...}
3   L[0] = x;
4 }
```

Implementação por Lista Encadeada:




```
1 void Insere(int x, Lista L) {
2   p = CriaNovaCelula(x);
3   L.primeiro= p;
4   ...
5 }
```

Programa do usuário da TAD:

```
1 int main() {
2   Lista L;
3   int x;
4   x = 20;
5   FazListaVazia(L);
6   Insere(x,L);
7   ...
8 }
```

TADs: Implementação

- Em linguagens orientadas a objeto (*C++*, *JAVA*) a implementação é feita usando classes e a especificação usando interfaces.
  - Orientação a objetos: POO.
- Em linguagens estruturadas (*C*, *pascal*) a implementação é feita pela definição de tipos juntamente com a implementação de funções.
  - Conceitos de C/C++ (***typedef*** e ***structs***).

TADs: Motivação

- Usuário do TAD vs. Programador do TAD
  - Usuário só "enxerga" a interface, não a implementação.

Práticas de Programação de TAD

- Para implementar um ***Tipo Abstrato de Dados*** em C, usa-se a definição de ***tipos*** juntamente com a implementação de ***funções*** que agem sobre aquele tipo.
- Como boa prática de programação, evita-se acessar o dado diretamente, fazendo o acesso somente através de das funções.

TADs: Motivação

- **Reúso**: abstração de detalhes da implementação específica.
- **Manutenção**: mudanças na implementação do TAD não afetam o código fonte dos programas que o utilizam (ocultamento de informação)
- **Corretude**: código testado em diferentes contextos.

Práticas de Programação de TAD

- Uma boa técnica de programação é implementar TADs em arquivos separados do programa principal.
- Para isso geralmente separa-se a declaração e a implementação do TAD em dois arquivos:
  - ***nome_tad.hpp***: com as declarações.
  - ***nome_tad.cpp***: com a implementação das declarações.
- Os programas, ou outros TADs, que utilizam o seu TAD devem usar:

  #include nome_arquivo.h

C++

- Uma boa técnica de programação é implementar TADs em arquivos separados do programa principal.
- Para isso geralmente separa-se a declaração e a implementação do TAD em dois arquivos:
  - ***nome_tad.hpp***: com as declarações.
  - ***nome_tad.cpp***: com a implementação das declarações.
- Os programas, ou outros TADs, que utilizam o seu TAD devem usar:

  #include nome_arquivo.h

Estrutura (struct)

- Uma estrutura (*struct*) é uma coleção de campos que podem ser referenciados pelo mesmo nome. A estrutura permite que informações relacionadas mantenham-se juntas.
- A declaração de uma estrutura define um tipo de dado, ou seja, informa ao computador o número de bytes que será necessáiro reservar para uma variável que venha a ser declarada como sendo desse tipo.

Exemplo Completo de TAD: Matriz

- Implemente um TAD *Matriz*, número de linhas e de colunas e uma coleção de elementos do tipo *float*.A TAD deve permitir as seguintes operações:
  - **Criar Matriz**: alocar dinamicamente uma matriz.
  - **Apagar Matriz**: desalocar dinâmica de uma matriz.
  - **Adicionar Elemento**: adição de um elemento a uma posição específica da matriz.
  - **Ler Elemento**: retornar o elemento pertencente a uma posição específica da matriz.
  - **Imprimir**: imprimir no *Console* todos os elementos de uma matriz.
- Faça um pequeno programa para testar o seu TAD.

Estrutura (struct)

Encapsulamento de Dados:

```
typedef struct matriz Matriz;
```

→ matriz.h

```
struct matriz{

    int num_linhas;
    int num_colunas;
    float* elementos;
};
```

→ matriz.cpp

Definindo o TAD

```
#ifndef matriz_hpp
#define matriz_hpp


typedef struct matriz Matriz;

Matriz* criarMatriz(int, int);

void criarMatriz(Matriz**, int, int);
void deletarMatriz(Matriz**);
void adicionarElemento(Matriz*, int, int, float);
float lerElemento(Matriz*, int, int);
void imprimir(Matriz*);

#endif /* matriz_hpp */
```

Exemplo de TAD: Matriz

- matriz.h
- matriz.cpp
- Makefile
- input.txt

Conclusões

- Conceitos e exemplos de Tipos Abstratos de Dados (TADs)
  - Especificação *vs* Implementação.
  - Boas práticas de *Engenharia de Software*.
  - Leitura/escrita de dados.
  - Alocação dinâmica de memória, se pertinente.

Exercício 2

- Incremente o exercício anterior, implementando uma TAD **Time de Futebol**.
  - Cada time possui os campos `nome`, `treinador`, `vitórias`, `empates`, `derrotas`, `jogadores` (um vetor de jogadores).
  - Implemente as operações:
    - `Atribui`: atribui valores para os campos. Um time precisa ter no mínimo cinco jogadores.
    - `Imprime`: imprime os dados/estatísticas do time.
    - `Pontuação`: retorna o número de pontos ganhos pelo time. Uma vitória corresponde a 3 pontos ganhos e um empate a 1 ponto ganho.
- Crie o `main` para testar seu TAD.
- Utilize alocação dinâmica.

Exercícios

Exercícios

Exercício 02

Faça um programa semelhante ao anterior, mas agora os elementos são do tipo `TRegistro*`, definido no Slide 17.

A leitura dos valores das identidades e do salário devem ser feitas via `scanf`.

Importante: não se esqueça de liberar a memória alocada. Note que agora, além de alocar memória para o vetor, você também deverá alocar memória para cada um de seus elementos. Desta forma, será necessário liberar toda a memória alocada para os elementos do vetor, para em seguida, desalocar a memória do vetor.

Exercício 1

- Implemente um TAD **Jogador de Futebol**.
  - Cada jogador possui os campos `nome`, `jogos`, `gols` e `assistências`.
  - Implemente as operações:
    - `Atribui`: atribui valores para os campos.
    - `Imprime`: imprime os dados/estatísticas do jogador.
    - `EhBom`: testa se o jogador é bom!!! (defina seu próprio critério de bom jogador)
- Crie o `main` para testar seu TAD.
- Utilize alocação dinâmica.