Universidade Federal de Ouro Preto
Campus João Monlevade

CSI 488 – Algoritmos e Estruturas de Dados I

TAD – Tipos Abstratos de Dados

Prof. Mateus Ferreira Satler

Índice

1. Introdução

- **Qual a diferença entre um algoritmo e um programa?**
  - **Algoritmo:**
    - Sequência de ações executáveis para a solução de um determinado tipo de problema
      - Exemplo: “Receita de Bolo”
    - Em geral, algoritmos trabalham sobre...
  - **Estruturas de Dados**
    - Conjunto de dados que representa uma situação real
    - Abstração da realidade
    - Estruturas de Dados e Algoritmos estão intimamente ligados

1. Introdução

- **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:
    - Ex.: variável `int` pode assumir valores 1, 2, 3, ...
  - O programador pode definir novos tipos de dados em termos de outros já definidos:
    - Tipos estruturados, por exemplo, vetores.

1. Introdução

- **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.

1. Introdução

- **Representação dos Dados**
  - Os dados podem estar representados (estruturados) de diferentes maneiras.
    - Normalmente, a escolha da representação é determinada pelas operações que serão utilizadas sobre eles.
  - Exemplo: números inteiros
    - Representação por palitinhos: II + III = IIIII
      - Boa para pequenos números (operações simples)
    - Representação decimal: 1278 + 321 = 1599
      - Boa para números maiores (operação complexa)

1. Introdução

- **Programas**
  - Um programa é uma formulação concreta de um algoritmo abstrato, baseado em representações de dados específicas.
  - Os programas são feitos em alguma linguagem que pode ser entendida e seguida pelo computador:
    - Linguagem de máquina
    - Linguagem de alto nível (uso de compilador)
      - Aqui vamos utilizar a Linguagem C

1. Introdução

- Há diferentes implementações possíveis para solucionar um mesmo problema.

- Por exemplo, desenvolver um programa para tratar assuntos relacionados a:
  - Alunos: armazena nome, matrícula, médias, faltas, etc.
  - Pacientes: armazena nome, endereço, convênio, histórico hospitalar, etc.

- Como abordar o problema?
  - Criar vetores para armazenar as informações?
  - **Agrupar informações que se relacionam de forma a criar um contexto maior.**

2. TAD – Tipos Abstratos de Dados

- Os **TADs** agrupam a **estrutura de dados** juntamente com as **operações** que podem ser feitas sobre esses dados.

- O TAD encapsula a estrutura de dados.
  - Os usuários do TAD só tem acesso a algumas operações disponibilizadas sobre esses dados.

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

2. TAD – Tipos Abstratos de Dados

- Um TAD especifica o tipo de dado (domínio e operações) sem referência a detalhes da implementação.

- Minimiza código do programa que usa detalhes de implementação:
  - Dando mais liberdade para mudar implementação com menor impacto nos programas.
  - Minimiza custos.

- Os programas que usam o TAD não "conhecem" as implementações dos TADs:
  - Fazem uso do TAD através de operações.

2. TAD – Tipos Abstratos de Dados

- 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

- Exemplo: tipo ***real***
  - $v = \Re$
  - $o = \{+, -, *, /, =, <, >, <=, >=, \neq\}$

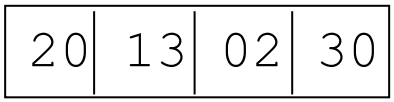
2. TAD – Tipos Abstratos de Dados

- **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.
  - Pro exemplo, o programador pode criar uma lista de clientes e aplicar operações sobre ela, mas não sabe como ela é representada internamente.

2.1. Ex.: Lista de Números Inteiros

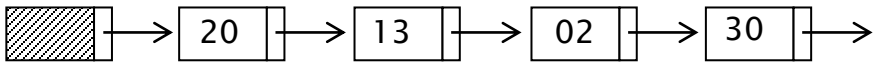
- **Operações:**
  - Faz Lista Vazia
  - Insere número no começo da lista
  - Remove de uma posição i

Implementação por Vetores:




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

Implementação por Listas Encadeadas




```
void Insere(int x, Lista L) {
    p = CriaNovaCelula(x);
    L^.primeiro = p;
    ...  }
```

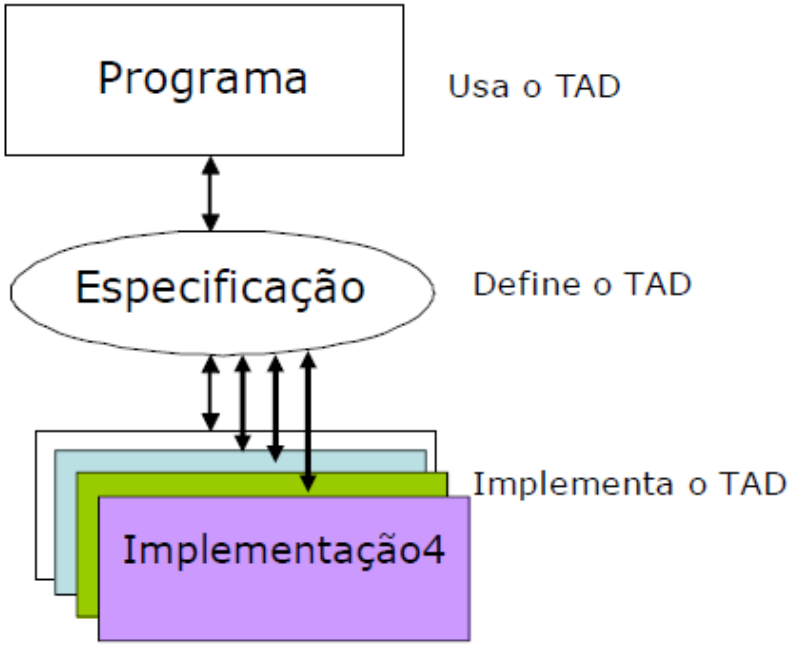
Programa usuário do TAD:

```
int main() {
   Lista L;
   int x;

   x = 20;
   FazListaVazia(L);
   Insere(x,L);
   ...
}
```

2. TAD – Tipos Abstratos de Dados

- Quando se usa um TAD, tem-se:
  - **Programas usuários**: A parte que usa o TAD
  - **Implementação**: A parte que implementa o TAD



- As camadas de software são independentes.
- Modificações na implementação do TAD não geram (grandes) mudanças no programa.
- Essa abordagem também permite o reuso de código.
  - A mesma implementação pode ser usada por vários programas.

3. Implementação de TADs

- Em linguagens orientadas por objeto (C++, Java) a implementação é feita através de classes.
- Em linguagens estruturadas (C, pascal) a implementação é feita pela definição de tipos juntamente com a implementação de funções e procedimentos.
- Como ainda não foi visto o conceito de orientação por objetos, vamos utilizar os conceitos de C (**`Typedef`** e **`Structs`**)
  - Orientação por objetos (classes, etc) vai ser vista em outras disciplinas (POO)

3. Implementação de TADs

- **Estruturas (Structs) em C**
  - Uma estrutura é uma coleção de uma ou mais variáveis, possivelmente de tipos diferentes, colocadas juntas sob um único nome para manipulação conveniente.
  - Por exemplo, para representar um aluno são necessárias as informações nome, matrícula, conceito.
  - Ao invés de criar três variáveis, é possível criar uma única variável contendo três campos.
  - Em C, usa-se a construção **`struct`** para representar esse tipo de dado.

3. Implementação de TADs

- Sintaxe:

```
struct nome_tipo_registro {
  tipo_1 variavel_1;
  tipo_2 variavel_2;
  ...
  tipo_n variavel_n;
};
```

- Devemos declarar variáveis deste novo tipo assim:

```
struct nome_tipo_registro variavel_registro;
```

3. Implementação de TADs

- Exemplo:

```
struct Aluno {
   string nome;
   int matricula;
   char conceito;
};

main() {
   struct Aluno al, aux;

   al.nome = “Pedro”;
   al.matricula = 200712;
   al.conceito = ‘A’;
   aux = al;
}
```

al:

Pedro	
200712	A

aux:

Pedro	
200712	A

3. Implementação de TADs

- **Declaração de Tipos**
  - Para simplificar, uma estrutura ou mesmo outros tipos de dados podem ser definidos como um novo tipo.
  - Uso da construção **typedef**
  - Sintaxe:

```
typedef struct nome_tipo_registro {
  tipo_1 variavel_1;
  tipo_2 variavel_2;
  ...
  tipo_n variavel_n;
}nome_tipo;
```

  - Devemos declarar variáveis deste novo tipo assim:

```
nome_tipo variavel_registro;
```

3. Implementação de TADs

- Acesso aos campos do registro
  - Operador **.**
  - Operador **->**

- Revisar:

  - Vetores de registros

  - Registros como parâmetros de sub-rotinas

  - Alocação dinâmica de registros

4. Modularização em C

- 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 regra de programação, evita-se acessar o dado diretamente, fazendo o acesso só através das funções.
  - Mas, diferentemente de C++ e Java, não há uma forma de proibir o acesso.

4. Modularização em C

- Uma boa técnica de programação é implementar os TADs em arquivos separados do programa principal.

- Para isso geralmente separa-se a **declaração** e a **implementação** do TAD em dois arquivos:

  - **`Nome_do_TAD.h`** : com a declaração
  - **`Nome_do_TAD.c`** : com a implementação

- O programa ou outros TADs que utilizam o seu TAD devem dar um **`#include`** no arquivo **`.h`**

4. Modularização em C

- Um programa em *C* pode ser dividido em **vários arquivos.**

1. Arquivos **fonte** tem a extensão **`.c`**

   - Denominados de módulos.

   - Cada módulo deve ser compilado separadamente:
     - **Resultado**: arquivos objeto **não executáveis**
       - Arquivos em linguagem de máquina com extensão **`.o`** ou **`.obj`**

4. Modularização em C

- 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`**

- *CodeBlocks* faz tudo isso automaticamente.

4. Modularização em C

2. Arquivo de **cabeçalhos** tem a extensão **`.h`**

   - Cabeçalhos das funções oferecidas pelo módulo e, eventualmente, os tipos de dados que ele exporta.
     - typedefs, structs, etc.

   - Segue o **mesmo nome** do módulo **`.c`** ao qual está associado, porém com a extensão **`.h`**

4.1. Exemplo

- Implemente um TAD **`ContaBancaria`**, com os campos **`número`** e **`saldo`** onde os clientes podem fazer as seguintes operações:
  - Iniciar uma conta com um número e saldo inicial
  - Depositar um valor
  - Sacar um valor
  - Imprimir o saldo
- Faça um pequeno programa para testar o seu TAD.

4.1. Exemplo – Resposta

- Arquivo: **ContaBancaria.h**

```
// definição do tipo
typedef struct ContaBancariaEst {
  int numero;
  double saldo;
} ContaBancaria;

// cabeçalho das funções e procedimentos
void inicializa (ContaBancaria*, int, double);
void deposito   (ContaBancaria*, double);
void saque      (ContaBancaria*, double);
void imprime    (ContaBancaria);
```

4.1. Exemplo – Resposta

- Arquivo: **ContaBancaria.c**

```
#include <stdio.h>
#include “Contabancaria.h”

void inicializa (ContaBancaria* conta, int numero, double saldo) {
  conta->numero = numero;
  conta->saldo = saldo; }

void deposito (ContaBancaria* conta, double valor) {
  conta->saldo += valor; }

void saque (ContaBancaria* conta, double valor) {
  conta->saldo -= valor; }

void imprime (ContaBancaria conta) {
  printf("Numero: %d\n", conta.numero);
  printf("Saldo: %f\n", conta.saldo); }
```

4.1. Exemplo – Resposta

- Arquivo: **Main.c**

```
#include <stdio.h>
#include “ContaBancaria.h”

void main() {

  ContaBancaria conta1;

  inicializa( &conta1, 918556, 300.00 );
  printf( "Antes da movimentacao:\n " );
  imprime( conta1 );
  deposito( &conta1, 50.00 );
  saque( &conta1, 70.00 );
  printf( "Depois da movimentacao:\n " );
  imprime( conta1 );
}
```

5. Referências

- Material de aula dos Profs. Luiz Chaimowicz e Raquel O. Prates, da UFMG: https://homepages.dcc.ufmg.br/~glpappa/aeds2/AEDS2.1%20Conceitos%20Basicos%20TAD.pdf
- DEITEL, P; DEITEL, H. *C How to Program*. 6a Ed. Pearson, 2010.
- LANGSAM,Y.; AUGENSTEIN, M.J.; TENENBAUM, A.M. Data Structures using C and C++, 2a edição . Prentice Hall of India. 2007.
- CORMEM, T. H.; et al. Introduction to algorithms, 3a edição, The MIT Press.
- DROZDEK A. Estrutura de dados e algoritmos em C++,1a edição Cengage Learning.