

Algoritmos e Estrutura de Dados II

Aula 11

Tipos Abstratos de Dados

Claudiane Maria Oliveira
claudiane@gmail.com

Tipos Abstratos de Dados

2

- Em um cenário real de desenvolvimento, pode haver diferentes implementações possíveis para o mesmo tipo de dado.
- Exemplo:
 - um inteiro (tipo de dado) pode ser implementado de formas bem diferentes, dependendo do Sistema Operacional sobre o qual o programa irá executar.

Tipos Abstratos de Dados

3

➤ Tipo de dado fracao - Exemplo 1

```
1  #include <iostream>
2
3  using namespace std;
4
5  /*Exemplo 1 - TAD Fracao*/
6
7  struct Fracao {
8      int numerador;
9      int denominador;
10 };
11
12
13 int main(){
14     Fracao f;
15     cout << "Informe o numerador ";
16     cin >> f.numerador;
17     cout << "Informe o denominador ";
18     cin >> f.denominador;
19     cout << "a fracao eh " << f.numerador << "/" << f.denominador;
20
21 }
```

Tipos Abstratos de Dados

4

➤ Tipo de dado fracao - Exemplo 2

```
1  #include <iostream>
2
3  using namespace std;
4
5  /*
6   * Exemplo 2 - TAD Fracao */
7
8  #define NUMERADOR 0
9  #define DENOMINADOR 1
10
11
12
13  struct Fracao {
14      int elementos[2];
15  };
16
17  int main(){
18      Fracao f;
19      cout << "Informe o numerador ";
20      cin >> f.elementos[NUMERADOR];
21      cout << "Informe o denominador ";
22      cin >> f.elementos[DENOMINADOR];
23      cout << "a fracao eh " << f.elementos[NUMERADOR] << "/" << f.elementos[DENOMINADOR];
24
25  }
```

Importante observar que, ao mudar a maneira como o tipo de dado é implementado, o programa que o usava também precisou ser alterado.!

Tipos Abstratos de Dados

5

- Em programas reais, as implementações dos tipos de dados são modificadas constantemente para melhorar a performance e a clareza dos mesmos, bem como para corrigir bugs, entre outras coisas.
- Essas mudanças podem gerar grande impacto nos programas usuários (que usam os módulos) do tipo de dado.
- Em muitos casos, será necessário re-implementar e re-compilar o programa usuário = CUSTO MUITO ALTO!



Tipos Abstratos de Dados

6

- Como podemos modificar as implementações dos tipos de dados com o menor impacto possível para os programas usuários?

Tipos Abstratos de Dados

7

- Como podemos modificar as implementações dos tipos de dados com o menor impacto possível para os programas usuários?
- **esconder** de quem usa um determinado tipo de dado a forma concreta como este tipo foi implementado.
- **encapsulamento**

Tipos Abstratos de Dados

8

- Os tipos de dados existem para serem usados por outros programas, com o intuito de acessar as informações neles armazenadas, por meio de operações apropriadas.
- Do ponto de vista do programador, muitas vezes é conveniente pensar nos tipos 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)**.

Tipos Abstratos de Dados

9

- Define um novo tipo de dado e um conjunto de operações para manipular dados desse tipo; e
- Encapsula e protege os dados de acessos indevidos.
- Exemplo:
 - Tipo de Dado → Aluno;
 - Operações → obterNumeroMatricula,
 - matricularEmDisciplina,
 - cancelarMatriculaEmDisciplina, etc.

Tipos Abstratos de Dados

10

- Um TAD facilita o entendimento, a manutenção e a reutilização de código.
- O segredo está no conceito de abstração:
 - capacidade de desconsiderar detalhes da informação disponível (simplificar) para se ter uma visão mais geral (mais abstrata).
 - a forma de implementação do TAD não precisa ser conhecida por quem vai utilizá-lo, apenas sua funcionalidade.

Tipos Abstratos de Dados

11

- Em programação, abstração geralmente é obtida por meio de mecanismos de encapsulamento e visibilidade.
- O encapsulamento permite que as partes que não devem ser acessadas diretamente pelo usuário fiquem protegidas.

Tipos Abstratos de Dados

12

- Em programação, abstração geralmente é obtida por meio de mecanismos de encapsulamento e visibilidade.
 - A visibilidade permite que os detalhes da implementação de um produto sejam omitidos e, junto com o encapsulamento, protege-os de acessos indevidos.
- Em Programação Orientada a Objetos esses são conceitos amplamente conhecidos e utilizados.

Tipos Abstratos de Dados

13

➤ Tipo fracao - Exemplo 3

```
1  #include <iostream>
2  |
3  #include "Fracao.hpp"
4
5  using namespace std;
6
7  /*
8   * Exemplo 3 - TAD Fracao */
9
10 int main(){
11     int numerador, denominador;
12
13     cout << "Informe o numerador ";
14     cin >> numerador;
15     cout << "Informe o denominador ";
16     cin >> denominador;
17
18     Fracao *f1 = criarFracao(numerador,denominador);
19     cout << "a fracao eh " << acessarNumeradorFracao(f1) << "/" << acessarDenominadorFracao(f1);
20
21 }
22
23
```



Tipos Abstratos de Dados

14

➤ Tipo fracao - Exemplo 3

```
1  #include <iostream>
2
3  #include "Fracao.hpp"
4
5  using namespace std;
6
7  /*
8   * Exemplo 3 - TAD Fracao */
9
10 int main(){
11
12     int numerador, denominador;
13
14     cout << "Informe o numerador ";
15     cin >> numerador;
16     cout << "Informe o denominador ";
17     cin >> denominador;
18
19     Fracao *f1 = criarFracao(numerador,denominador);
20
21
22     cout << "a fracao eh " << acessarNumeradorFracao(f1) << "/" << acessarDenominadorFracao(f1);
23
24 }
25
```

Observe que não faço ideia de como o tipo de dado foi implementado. Idealmente, a implementação desse tipo deve estar “invisível” e inacessível ao meu programa (ocultamente de informação).

Tipos Abstratos de Dados

15

- Isso é o que chamamos de abstração: capacidade de desconsiderar detalhes da informação disponível (simplificar) para se ter uma visão mais geral (mais abstrata).
 - a forma de implementação do TAD não precisa ser conhecida por quem vai utilizá-lo, apenas sua funcionalidade.

Tipos Abstratos de Dados

16

- Vantagens do uso de TADs:
- **Reuso:** uma vez definido, implementado e testado, o TAD pode ser reusado por diferentes programas.
 - **Manutenção:** mudanças na implementação do TAD geralmente não afetam o código fonte dos programas que o utilizam (decorrência do ocultamento de informação).
 - **Legibilidade:** o programador pode se concentrar mais no problema a ser resolvido e menos nos detalhes de implementação dos tipos de dados que ele utiliza.

Tipos Abstratos de Dados

17

- Para implementação de TADs em C, utilizamos o conceito de módulos: um arquivo com estruturas e funções que serão utilizadas por outros programas.
- Módulos são compostos de duas partes:
 - A **interface** ou arquivo de cabeçalho (.hpp), que define as estruturas e tipos de dados utilizados por ele, bem como o protótipo das funções disponibilizadas para manipulação desses dados.
 - A **implementação** (arquivo .cpp): que completa a definição das estruturas e tipos de dados e implementa as funções disponibilizadas no arquivo de cabeçalho.

Tipos Abstratos de Dados

18

- Exemplo: considere um programa simples (Fracao.cpp), que contém funções como:

```
// Criar a Fração
Fracao *criarFracao(int num, int den){

    Fracao *f = new Fracao();
    f->denominador = den;
    f->numerador = num;
    return f;
}

// Acessar o valor do numerador de uma fracao
int acessarNumeradorFracao(Fracao *f){
    return f->numerador;
}

// Acessar o valor do denominador de uma fracao
int acessarDenominadorFracao(Fracao *f){
    return f->denominador;
}
```

Tipos Abstratos de Dados

19

- Suponha que você queira disponibilizá-lo como um módulo, de forma que ele possa ser utilizado por outras pessoas em seus programas.
- Para fazer isso da forma correta, você deve especificar a interface do seu módulo (arquivo .h).
- Esse arquivo contém apenas os protótipos das funções soma e multiplica: tipo de retorno, nome da função e tipos de parâmetros exigidos pelas funções.

```
8 | Fracao *criarFracao(int, int);  
9 | int acessarNumeradorFracao(Fracao *);  
10 | int acessarDenominadorFracao(Fracao *);
```



Tipos Abstratos de Dados

20

- Deve-se incluir essa interface no arquivo com a implementação das funções, da seguinte forma:

#include "Fracao.hpp"

Note que os arquivos de cabeçalhos das funções da biblioteca padrão do C++ (que acompanham seu compilador) são incluídos da forma:

– #include <arquivo>.

Os arquivos de cabeçalhos dos seus módulos são incluídos da forma:

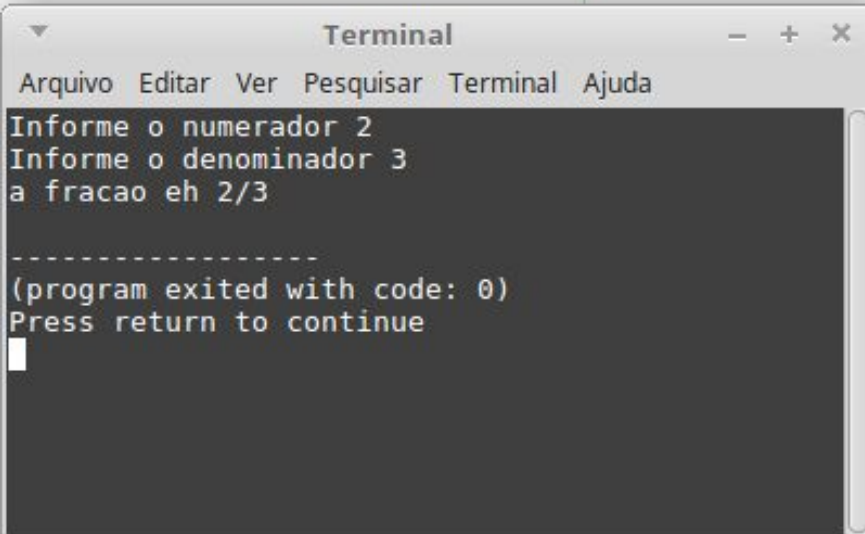
– #include “arquivo.hpp”.

Tipos Abstratos de Dados

21

- Agora, para utilizar o módulo “Fracao”, basta incluir a interface do mesmo em seu programa.

```
1  #include <iostream>
2
3  #include "Fracao.hpp"
4
5  using namespace std;
6
7  /*
8   * Exemplo 3 - TAD Fracao */
9
10 int main(){
11
12     int numerador, denominador;
13
14     cout << "Informe o numerador ";
15     cin >> numerador;
16     cout << "Informe o denominador ";
17     cin >> denominador;
18
19     Fracao *f1 = criarFracao(numerador,denominador);
20
21
22     cout << "a fracao eh " << acessarNumeradorFracao(f1) << "/" << acessarDenominadorFracao(f1);
23
24
25 }
```



A terminal window titled "Terminal" with a menu bar containing "Arquivo", "Editar", "Ver", "Pesquisar", "Terminal", and "Ajuda". The terminal output shows the program's execution: it prompts for the numerator (2) and denominator (3), displays the fraction "a fracao eh 2/3", and then shows the message "(program exited with code: 0) Press return to continue".

```
Terminal
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda
Informe o numerador 2
Informe o denominador 3
a fracao eh 2/3
-----
(program exited with code: 0)
Press return to continue
```



Compilação e “Linkedição” de um TAD

22

- Cada módulo deve ser compilado separadamente, assim como o programa principal (que usa os módulos). Exemplo:
 - `g++ -cpp fracao.cpp usaFracao.cpp`
- O resultado será a geração de arquivos-objeto não executável (.o ou .obj).
- Posteriormente, os arquivos-objeto devem ser mesclados em um único arquivo executável, por meio de um processo denominado linkedição. Para isso, faz-se:
- `g++ -o MeuPrograma fracao.o usaFracao.o`

Compilação e “Linkedição” de um TAD

23

- Assim, a construção de um programa executável requer a linkedição dos módulos (que podem ser mantidos já pré-compilados em uma biblioteca) junto com o programa
- O programador não precisa olhar o código do módulo de definição para usar o TAD! Basta conhecer a interface de acesso

Compilação e “Linkedição” de um TAD

24

- Para fazer com que o Geany considere a existência de módulos em seu programa, abra-o, clique em “Construir → Definir comandos de construção”.
- Na janela que aparecer:
 - altere a linha “Compile” para `g++ -Wall -c *.cpp`
 - altere a linha “Build” para `g++ -Wall -o "%e" *.o`

Considerações Importantes

25

- O arquivo `fracao.cpp` não possui função `main`.
- O arquivo `usaFracao.cpp` não tem conhecimento sobre como as funções do módulo “Fracao” foram implementadas → encapsulamento e visibilidade = abstração.
- Você poderia fornecer o arquivo de interface + o arquivo de implementação compilado para seus usuários e mesmo assim eles conseguiriam usar o módulo.
- O arquivo `Fracao.cpp` não será recompilado! Isso reduz bastante o tempo para compilação e implantação de programas em C++

Considerações Importantes

26

- Pode acontecer de um arquivo de interface ser incluído mais de uma vez em um mesmo programa, provocando erros de redeclaração de estruturas e funções.
- Para evitar esse tipo de problema, utilize condições de guarda com as diretivas `#ifndef`, `#define` e `#endif`.

```
1  #ifndef FRACA0_HPP
2  #define FRACA0_HPP
3
4  // protótipo das funcoes
5
6  #endif
```

Considerações Importantes

27

- Ao definir um registro em uma interface, qualquer usuário terá acesso e poderá alterar os valores dos campos desse registro e esse é um comportamento indesejado para um TAD (por que?).
- Para resolver esse problema, você pode simplesmente declarar o nome do registro na interface e completar a declaração desse registro no arquivo de implementação (.cpp).
 - Dessa forma, os campos do registro ficarão invisíveis ao usuário do módulo (abstração). Contudo, quem usar o módulo só conseguirá instanciar registros de forma dinâmica.

Considerações Importantes

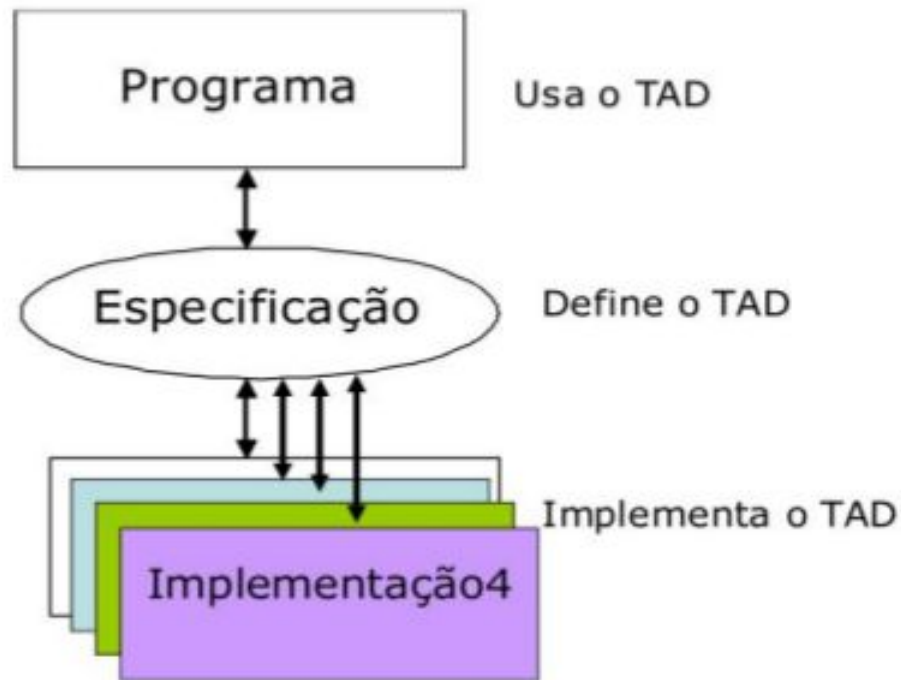
28

Exemplo.h	Exemplo.c
<pre>struct MeuRegistro; MeuRegistro* criar();</pre>	<pre>#include "Exemplo.h" struct MeuRegistro { int campo1; double campo2 }; MeuRegistro* criar() {...}</pre>
UsaExemplo.c	
<pre>#include "Exemplo.h" int main() { MeuRegistro *mr = criar(); mr->campo1; // impossível: gera erro de compilação. }</pre>	

Tipos Abstratos de Dados

29

➤ Resumindo o TAD



Tipos Abstratos de Dados

30

- Um TAD especifica tudo que se precisa saber para utilizá-lo.
- Não faz referência à maneira com a qual o tipo de dado será (ou é) implementado.
- Quando usamos TADs, nossos sistemas ficam divididos em programas usuários (a parte que usa o TAD) e a parte que implementa o TAD.

Exercícios

31

- Implementar um TAD que represente uma fração, do tipo N/M , onde N e M são números inteiros positivos e $M > 0$.
- Crie funções capazes de criar uma fração, acessar e alterar o valor de seus elementos (numerador e denominador), multiplicar duas frações e apagar da memória uma fração.
- Observações:
 - Se os valores de N e/ou M forem negativos, os mesmos devem ser convertidos em números positivos antes de criar a fração.
 - Se o valor de M for nulo, deve-se criar uma fração com denominador igual a 1.

Links para as aulas

32

➤ **Aula de AED2 - Realizada no dia 07/04/2020 - 19:00**

<https://drive.google.com/file/d/1J25TJFMNYDugMzNjB3atJfjLKn6pu673/view?usp=sharing>

➤ **Aula de LAB2 - Realizada no dia 07/04/2020 - 20:55**

<https://drive.google.com/file/d/1NHMceecQixqbR54f9KkgNWC5cek-M-q8B/view?usp=sharing>



FAGAMMON
FACULDADE PRESBITERIANA GAMMON



Algoritmos e Estrutura de Dados II

Aula 11

Tipos Abstratos de Dados

Claudiane Maria Oliveira
claudiane@gmail.com

