UFMG
UNIVERSIDADE FEDERAL DE MINAS GERAIS

Programação e Desenvolvimento de Software 2

Tipos Abstratos de Dados

Prof. Douglas G. Macharet
douglas.macharet@dcc.ufmg.br

DCC
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Introdução

- Algoritmo
  - Sequência de ações executáveis
  - Transformam uma entrada em uma saída
  - Trabalham sobre estruturas de dados
- Estruturas de dados
  - Informações que representam uma situação real
  - Abstração da realidade
  - Suportam as operações dos algoritmos

Introdução

- Como representar as variáveis no programa?
- O que é um tipo de dado?
- O que os tipos "int" e "bool" representam?
  - Em quais operações podem ser utilizados?
- Programa possuirá apenas tipos como esses?
  - Primitivos
  - Como criar conceitos mais genéricos?

Introdução

- Diferença entre Programa e Algoritmo?
  - Um programa é uma realização concreta de um algoritmo abstrato, baseado em representações de dados específicas
  - Programas precisam ser implementados numa linguagem que pode ser entendida e seguida pelo computador
    - C, C++, Java, Python, ...

Introdução

- Quando um programa é considerado "bom"?
  1. Funciona (faz o que foi especificado)
  2. Fácil de entender e modificar
  3. Razoavelmente eficiente (recursos)
- Bons programas fazem uso de Abstração
  - Conceito (ideia) → Implementação (concreto)
- Tipos Abstratos de Dados
  - Ajudam alcançar (2), que facilita (1)

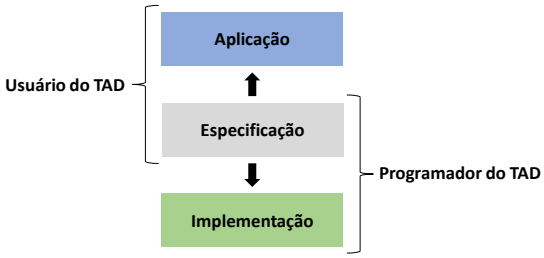
Tipos Abstratos de Dados (TADs)

- Generalização de tipos primitivos
- Encapsula uma estrutura de dados
  - Conjunto de valores
  - Conjunto de operações sobre esses valores
- Usuário do TAD x Programador do TAD
  - Acesso às operações disponibilizadas
  - Conhece a interface, não a implementação

Tipos Abstratos de Dados (TADs)

- Desvincular o tipo da sua implementação
  - Ocultação de informação (*information hiding*)
- Especificação (interface)
  - Com que tipo de "coisa" estamos trabalhando?
  - Quais operações podem ser feitas nela?
- Implementação
  - Como essa "coisa" deve ser implementada?
  - E suas operações?

Tipos Abstratos de Dados (TADs)



Tipos Abstratos de Dados (TADs)

- Um TAD é um tipo caracterizado não pelos valores que o compõe, mas pelas operações que se aplicam sobre ele

O que o TAD representa e faz é mais importante do que como ele faz!

Tipos Abstratos de Dados (TADs)

- Quais as vantagens disso?
  - Integridade, manutenção, reutilização, ...
  - Podemos modificar a implementação interna do TAD sem modificar o código que usa o TAD
- Exemplo
  - Ponto, Aluno, Carro
  - Lista, Árvore, Pilha, Fila
  - Quais operações são aplicáveis nesses casos?

Tipos Abstratos de Dados (TADs)

- Lista de números: 4, 1, 5, 6, ...
  - Quais operações podemos ter?
    - Criar uma nova lista, inserir um número, remover, ordenar, ...
  - Como deve ser feita a implementação?

Usuário do TAD

```
int main() {
  Lista *L;
  int x = 20;

  L = criar_lista();
  inserir(L, x);
  ...
}
```

Programador do TAD

- Vetor: 20 13 02 30

```
void inserir(Lista *l, int x) {
  l->arranjo[...] = x;
  ...
}
```

- Lista:

```
void inserir(Lista *l, int x) {
  Celula *c = criar_celula(x);
  l->ultimo = c;
}
```


Implementação de TADs

- Em linguagens orientadas por objeto a implementação é feita através de classes
  - C++, Java, C#, ...
- Em linguagem estruturadas (C, Pascal), a implementação é feita com definições de tipos e implementação de funções

Structs

- Coleção de variáveis, possivelmente de tipos diferentes, juntas sob um único nome
- Ponto 3D
  - x, y, z
  - Ao invés de criar três variáveis, é possível criar uma única variável contendo três campos.
- Em C++ uma struct é similar à uma classe
  - Diferença no nível de proteção (outra aula)

Structs
Exemplo 1

```
#include <cstdlib>
#include <iostream>
#include <cmath>

using namespace std;

struct Ponto3D {
  float x;
  float y;
  float z;

  float calcularDistancia(Ponto3D* p2) {
    float dx = p2->x - x;
    float dy = p2->y - y;
    float dz = p2->z - z;
    return sqrt(dx*dx + dy*dy + dz*dz);
  }
};
```


Structs
Exemplo 1

```
int main() {

  Ponto3D* p1 = new Ponto3D;
  p1->x = 0;
  p1->y = 0;
  p1->z = 0;

  Ponto3D* p2 = new Ponto3D;
  p2->x = 5;
  p2->y = 5;
  p2->z = 5;

  cout << p1->calcularDistancia(p2) << endl;

  delete p1;
  delete p2;

  return 0;
}
```


Structs
Exemplo 2

- Como representar um Aluno?
  - Quais atributos/dados?
    - Nome, matrícula, curso, RSG...
  - Quais operações sobre esses dados?
    - Matricular, Calular RSG, ...

Structs
Exemplo 2

```
#include <iostream>
#include <string>

using namespace std;

struct Aluno {
  string nome;
  int matricula;

  float calcularRSG() {
    // Fazer a conta necessaria
    return 0;
  }
};

int main() {
  Aluno aluno;
  aluno.nome = "Jose da Silva";
  aluno.matricula = 2018123345;
  cout << aluno.nome << endl;
  return 0;
}
```


Structs

- Essas implementações seguem corretamente os conceitos de TADs que vimos?
- Qual o problema?
  - Especificação e Implementação estão juntas
- Como resolver isso?
  - Separar → Modularizar
  - Alguma ideia de como implementar?

Modularização

- Separação em arquivos
  - Especificação: NomeDoTAD.hpp
  - Implementação: NomeDoTAD.cpp
- Parte do programa e outros TADs que o utilizam devem dar #include no arquivo .hpp
- O arquivo .hpp define o contrato do TAD

Modularização
Exemplo 3

- Como representar uma Circunferência?
  - Quais atributos/dados?
    - Coordenadas centro, raio, ...
  - Quais operações sobre esses dados?
    - Calular Área, Perímetro, ...

Modularização
Exemplo 3

Circunferencia.hpp

```
#ifndef M_PI
#define M_PI  3.14159265358979323846
#endif

#ifndef CIRCUNFERENCIA_H
#define CIRCUNFERENCIA_H

#include <cmath>

struct Circunferencia {
    double _x, _y;
    double _raio;

    Circunferencia(double, double, double);
    double calcularArea();
};

#endif
```

https://en.wikipedia.org/wiki/Include_guard

Modularização
Exemplo 3

A qual TAD se refere.

Circunferencia.cpp

```
#include "Circunferencia.hpp"

Circunferencia::Circunferencia(double x, double y, double raio) {
  _x = x;
  _y = y;
  _raio = raio;
}

double Circunferencia::calcularArea() {
    return M_PI * pow(_raio, 2);
}
```

Operação sendo implementada.

Modularização
Exemplo 3

main.cpp

```
#include <iostream>
#include "Circunferencia.hpp"

using namespace std;

int main() {

  Circunferencia* circ = new Circunferencia(0, 0, 10);
  cout << circ->calcularArea() << endl;

  delete circ;

  return 0;
}
```

```
g++ -std=c++11 -Wall main.cpp Circunferencia.cpp -o main
```


Tipos Abstratos de Dados (TADs)
Definição

- As descrições desses tipos devem ser
  - Precisas e sem ambiguidades
  - Completas
    - Tão completas quanto o necessário
  - Não excessivamente detalhadas
    - Tão abstrata quanto possível

Tipos Abstratos de Dados (TADs)
Operações

- Quais operações devem estar em um TAD?
- Filosofia egoísta
  - Se algo não é útil, então não é necessário
  - Informe ao usuário apenas o necessário
    - Lembre-se do princípio anterior
    - Desejo x Necessidade
    - Liberdade para alterações futuras

Desenvolvimento

- Níveis de detalhamento
  - Especificação/Requisitos
    - Mundo real
  - Projeto/Desenho/Modelagem
    - Tipos Abstratos de Dados
  - Implementação
    - Classes

Desenvolvimento

- Desenvolvimento Orientado a Objetos
  - Construção de programas como uma coleção estruturada de implementações (talvez) parciais de tipos de dados abstratos (classes)
- Modularização
  - Programação Orientada a Objetos
  - Programação (design) por contrato

Desenvolvimento
Contrato

- O que é um contrato?
  - Acordo entre duas ou mais partes
- Quais contratos existem em um programa?
  - Requisitos de funcionalidade, desempenho, ...
  - Operações disponíveis em um TAD
    - Entradas e saídas

Desenvolvimento
Contrato

- Como garantir que o contrato estará sendo cumprido pelo Usuário do TAD?
  - Tipos
  - Asserções
  - Exceções
- Requisitos ←→ Implementação
  - Garantir a consistência entre esses artefatos

Desenvolvimento

- Qual implementação é melhor?

```
struct TriangRetangulo {
  float base, altura;
}
```

```
struct TriangRetangulo {
  float base, hipot, ang;
}
```

- A implementação não é o mais importante
  - Pense no tipo como um conjunto de operações
    - Criar, SetarBase, SetarAltura, CalcularArea, ...
  - Usuário deve acessar os dados pelas operações!

Desenvolvimento

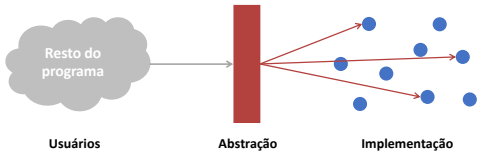
- Esses TADs são iguais ou diferentes?

```c
struct Ponto {
  float x;
  float y;
}
```

```c
struct Ponto {
  float r;
  float theta;
}
```

- Iguais (conceito) e Diferentes (dados)!
- Tipo/TAD
  - Elementos com propriedades semelhantes
  - Clientes dependem do conceito geral

Desenvolvimento



- Implementação não deve ser conhecida
  - A abstração deve prover os métodos que serão utilizados para manipular os dados