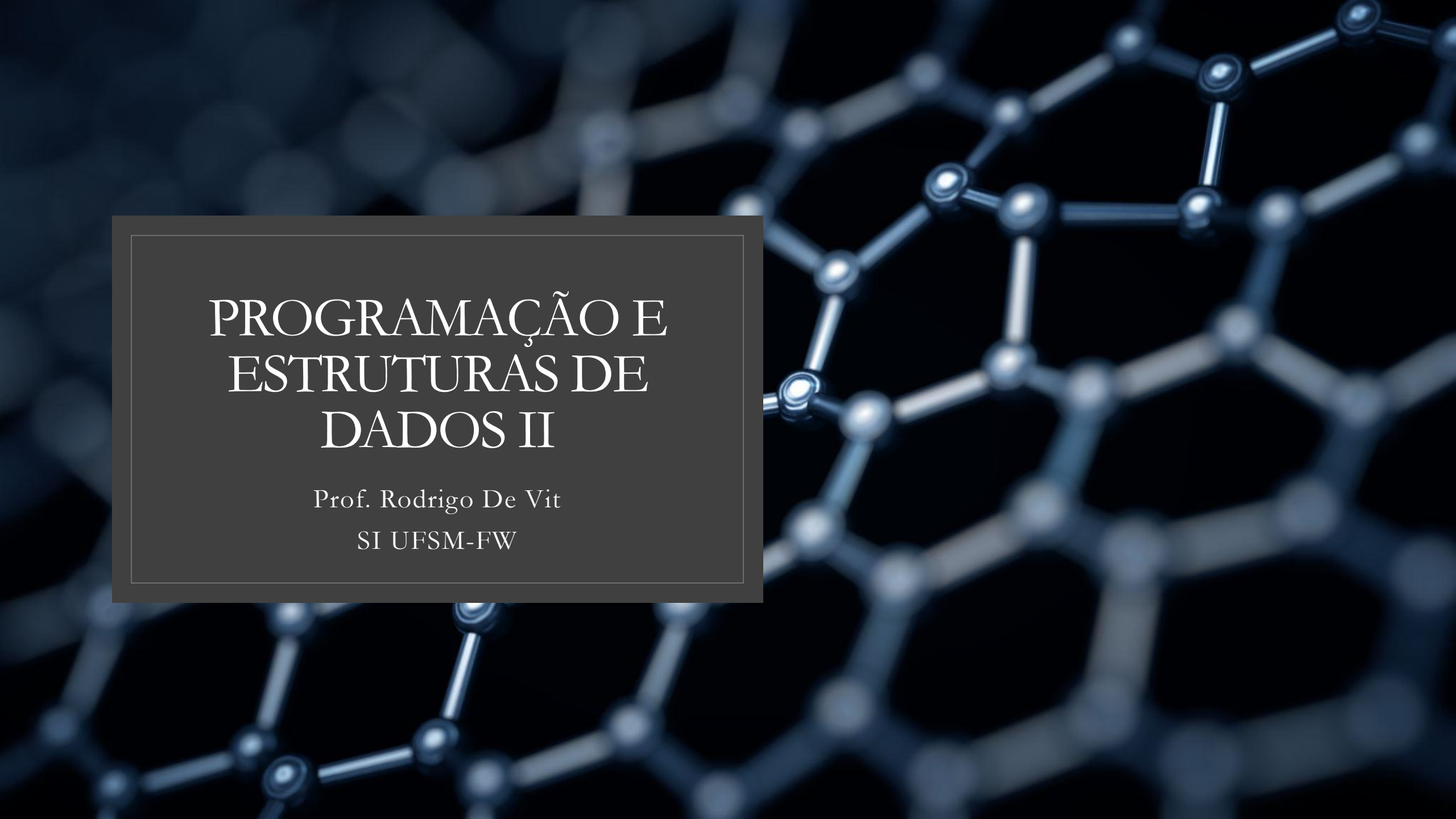


PROGRAMAÇÃO E ESTRUTURAS DE DADOS II

Prof. Rodrigo De Vit

SI UFSM-FW

Revisão

- UNIDADE 4 - DADOS, EXPRESSÕES E ALGORITMOS SEQUENCIAIS: 4.1 - Tipos de dados; 4.2 - Constantes e variáveis; 4.3 – Expressões; 4.4 – Atribuição; 4.5 - Entrada e saída.

- UNIDADE 5 - ALGORITMOS ESTRUTURADOS: 5.1 - Execução condicional.; 5.2 - Estruturas de repetição; 5.3 - Contadores e acumuladores.

- UNIDADE 6 - DADOS ESTRUTURADOS: 6.1 - Variáveis compostas homogêneas; 6.1.1 – Unidimensionais; 6.1.2 – Multidimensionais; 6.2 - Variáveis compostas heterogêneas; 6.3 - Ponteiros e estruturas dinâmicas.

- UNIDADE 7 – MODULARIZAÇÃO: 7.1 – Subprogramas; 7.2 - Argumentos.

Aula de hoje

- **Aula (02), 13 e 14 de agosto:**
- UNIDADE 1 - ESTRUTURAS LINEARES E ENCADEADAS
- 1.1 - Abstração de dados.
- Neste tópico, discutiremos uma importante técnica de programação baseada na definição de ***Tipos Abstratos de Dados*** **(TAD)**. Veremos também como a linguagem C pode nos ajudar na implementação de um TAD, através de alguns de seus mecanismos básicos de ***modularização*** (divisão de um programa em vários arquivos fontes).

- Editar arquivo prog1.c e str.h.

- Agora, ao invés de repetir manualmente os cabeçalhos dessas funções, todo módulo que quiser usar as funções de str.c precisa apenas incluir o arquivo str.h.

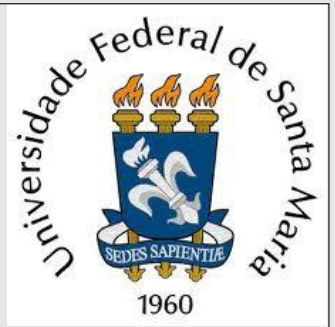
Tipos abstratos de dados - TAD

- Geralmente, **um módulo agrupa vários tipos e funções com funcionalidades relacionadas**, caracterizando assim uma finalidade bem definida. Podemos, por **exemplo**, criar um **TAD para representar matrizes alocadas dinamicamente**. Para isso, **criamos um tipo "matriz"** e **uma série de funções que o manipulam**. Podemos pensar, por exemplo, em funções que acessem e manipulem os valores dos elementos da matriz. **Criando um tipo abstrato, podemos "esconder" a estratégia de implementação.** Quem usa o tipo abstrato precisa apenas conhecer a funcionalidade que ele implementa, não a forma como ele é implementado. **Isto facilita a manutenção e o re-uso de códigos. (POO)**
- **Exemplo 1: TAD Ponto**
- Como nosso primeiro exemplo de TAD, vamos considerar a **criação** de um **tipo de dado** para representar um **ponto no R^2**. Para isso, **devemos definir um tipo abstrato, que denominaremos de Ponto, e o conjunto de funções que operam sobre esse tipo.** Neste exemplo, vamos considerar as seguintes operações:
  - **cria**: operação que cria um ponto com coordenadas x e y;
  - **libera**: operação que libera a memória alocada por um ponto;
  - **acessa**: operação que devolve as coordenadas de um ponto;
  - **atribui**: operação que atribui novos valores às coordenadas de um ponto;
  - **distancia**: operação que calcula a distância entre dois pontos.

Tipos abstratos de dados – TAD
Exemplo 1: TAD ponto

- **Editar ponto.h.** Note que a composição da estrutura Ponto (struct ponto) não é exportada pelo módulo. Dessa forma, **os demais módulos que usarem esse TAD não poderão acessar diretamente os campos dessa estrutura**. **Os *clientes*** desse TAD **só terão acesso às informações que possam ser obtidas através das funções exportadas pelo arquivo ponto.h. O arquivo de implementação do módulo (arquivo ponto.c) deve sempre incluir o arquivo de interface do módulo**. Isto é necessário por duas razões. **Primeiro**, podem existir definições na interface que são necessárias na implementação. No nosso caso, por exemplo, precisamos da definição do tipo Ponto. A **segunda razão** é garantirmos que as funções implementadas correspondem às funções da interface. Como o protótipo das funções exportadas é incluído, o compilador verifica, por exemplo, se os parâmetros das funções implementadas equivalem aos parâmetros dos protótipos. Além da própria interface, precisamos naturalmente incluir as interfaces das funções que usamos da biblioteca padrão.
- Editar ponto.c.

Tipos abstratos de dados – TAD
Exemplo 2: TAD matriz



- A implementação de um TAD fica "escondida" dentro de seu módulo. Assim, podemos experimentar diferentes maneiras de implementar um mesmo TAD, sem que isso afete os seus clientes. Para ilustrar essa independência de implementação, vamos considerar a **criação de um tipo abstrato de dados para representar matrizes de valores reais alocadas dinamicamente**, com dimensões m por n fornecidas em tempo de execução. Para tanto, devemos definir um tipo abstrato, que denominaremos de Matriz, e o conjunto de funções que operam sobre esse tipo. Neste exemplo, vamos considerar as seguintes operações:
  - **cria**: operação que cria uma matriz de dimensão m por n;
  - **libera**: operação que libera a memória alocada para a matriz;
  - **acessa**: operação que acessa o elemento da linha i e da coluna j da matriz;
  - **atribui**: operação que atribui o elemento da linha i e da coluna j da matriz;
  - **linhas**: operação que devolve o número de linhas da matriz;
  - **colunas**: operação que devolve o número de colunas da matriz.
- Editar matriz.h, matriz1.c e matriz2.c.