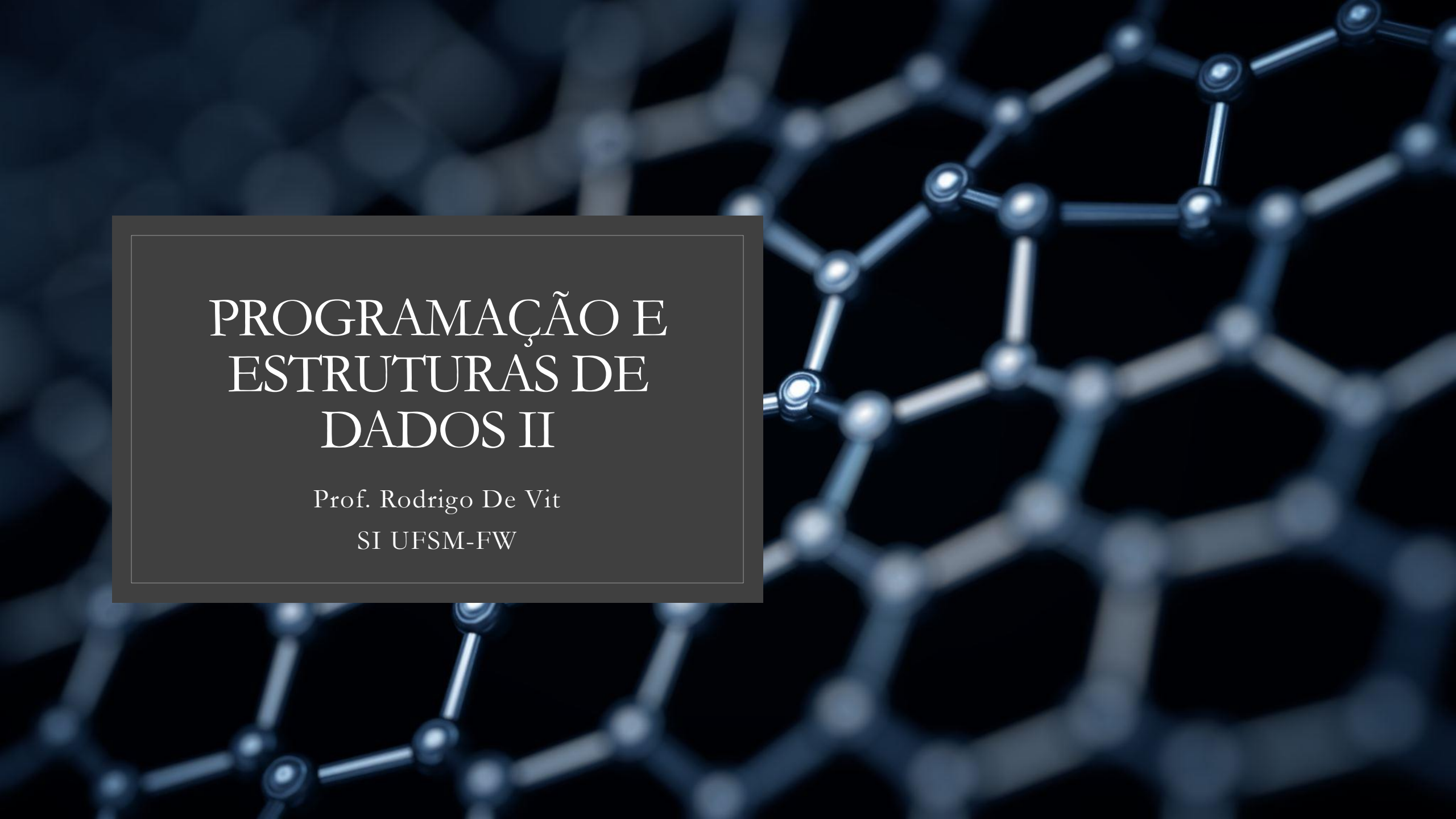

PROGRAMAÇÃO E ESTRUTURAS DE DADOS II

Prof. Rodrigo De Vit

SI UFSM-FW

Revisão

- **Aula (03), 20 e 21 de agosto e Aula (04), 27 e 28 de agosto:**
- UNIDADE 1 - ESTRUTURAS LINEARES E ENCADEADAS
- 1.2 - Estrutura Dinâmica- Listas Encadeadas.
- 1.2.1 - Conceituação e ponteiros.
- 1.2.2 - Listas lineares.
- 1.2.4 - Listas duplamente encadeadas.

Pilhas e Filas

- **Aula (03), 20 e 21 de agosto e Aula (04), 27 e 28 de agosto:**
- UNIDADE 1 - ESTRUTURAS LINEARES E ENCADEADAS
- 1.2.3 - Manipulação de pilhas e filas.

Pilhas

- **Uma das estruturas de dados mais simples é a pilha.** Possivelmente por essa razão, é a estrutura de dados mais utilizada em programação, sendo inclusive **implementada diretamente pelo *hardware* da maioria** das máquinas modernas. A ideia fundamental da pilha é que **todo o acesso a seus elementos é feito através do seu topo**. Assim, **quando um elemento novo é introduzido na pilha, passa a ser o elemento do topo, e o único elemento que pode ser removido da pilha é o do topo.** Isto faz com que **os elementos da pilha sejam retirados na ordem inversa à ordem em que foram introduzidos**: o primeiro que sai é o último que entrou (a sigla **LIFO – *last in, first out*** – é usada para descrever esta estratégia).
- Existem duas operações básicas que devem ser implementadas numa estrutura de pilha: **a operação para empilhar um novo elemento, inserindo-o no topo, e a operação para desempilhar um elemento, removendo-o do topo**. É comum nos referirmos a essas duas operações pelos termos em inglês ***push* (empilhar) e *pop* (desempilhar)**.

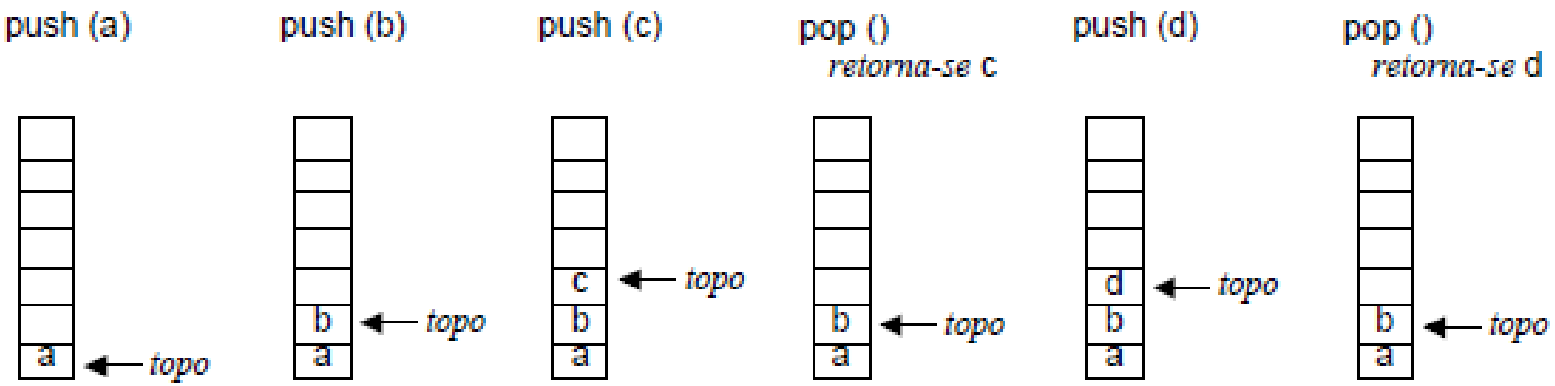

Figura 10.1: Funcionamento da pilha.

Pilhas

- **Interface do tipo pilha**

O arquivo `pilha.h`, que representa a interface do tipo, pode conter o seguinte código:

```
typedef struct pilha Pilha;

Pilha* cria (void);
void push (Pilha* p, float v);
float pop (Pilha* p);
int vazia (Pilha* p);
void libera (Pilha* p);
```

criar uma estrutura de pilha;
inserir um elemento no topo (*push*);
remover o elemento do topo (*pop*);
verificar se a pilha está vazia;
liberar a estrutura de pilha.

A função `cria` aloca dinamicamente a estrutura da pilha, inicializa seus campos e retorna seu ponteiro; as funções `push` e `pop` inserem e retiram, respectivamente, um valor real na pilha; a função `vazia` informa se a pilha está ou não vazia; e a função `libera` destrói a pilha, liberando toda a memória usada pela estrutura.

Pilhas

Implementação de pilha com vetor

A estrutura que representa o tipo pilha deve, portanto, ser composta pelo vetor e pelo número de elementos armazenados.

```
#define MAX  50

struct pilha {
   int n;
   float vet[MAX];
};
```

Vide PilhaV.c

Pilhas

Implementação de pilha com lista

- **Quando o número máximo de elementos que serão armazenados na pilha não é conhecido**, devemos implementar a pilha usando uma estrutura de dados dinâmica, no caso, empregando uma **lista encadeada**. Os elementos são armazenados na lista e a pilha pode ser representada simplesmente por um ponteiro para o primeiro nó da lista.

A estrutura que representa o tipo pilha deve, portanto, ser composta pelo vetor e pelo número de elementos armazenados.

```
#define MAX  50

struct pilha {
   int n;
   float vet[MAX];
};
```

O nó da lista para armazenar valores reais pode ser dado por:

```
struct no {
   float info;
   struct no* prox;
};
typedef struct no No;
```

A estrutura da pilha é então simplesmente:

```
struct pilha {
   No* prim;
};
```

Vide PilhaL.c