

CI1055: Algoritmos e Estruturas de Dados I

Profs. Drs. Marcos Castilho, Bruno Müller Jr, Carmem Hara

Departamento de Informática/UFPR

19 de agosto de 2020

Resumo

Introdução ao tipo de dados estruturado vetores

Objetivos da aula

- Introduzir um tipo de dados estruturado: os vetores
- Perceber os detalhes da implementação de vetores em *Pascal*

Estruturas de dados: motivação

- Os problemas abordados até o momento lidavam com pequenas quantidades de informação
- Quando aumenta a quantidade de informação necessária para resolver um problema, são necessários mecanismos para armazenar esta informação
- Alguns desses mecanismos já estão embutidos nas linguagens de programação: vetores, matrizes e registros
- As próximas aulas irão abordar estas estruturas de dados, as categorias de problemas para as quais elas são apropriadas e sua sintaxe na linguagem Pascal

- Conceitos
- Vetores em Pascal

Problema: Escreva um programa Pascal que lê até 200 valores inteiros e os imprima na ordem inversa da leitura.

- Solução óbvia: utilizar 200 variáveis do tipo longint
- Solução proposta: utilizar uma estrutura de dados chamada vetor capaz de armazenar os 200 elementos
- Mas... O que são vetores, como armazenar informação neles e como recuperar informação armazenada neles?

- A figura mostra conceitualmente o que é um vetor
- Cada quadrado armazena exatamente uma informação de um mesmo tipo (inteiro, real, etc)
- Qualquer quadrado pode ser acessado individualmente através do índice

1	2	3	4	5	6	7	8	9	10
15	12	27	23	7	2	0	18	19	21

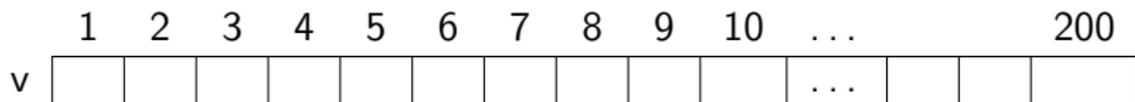
- Qual o conteúdo do quadrado cujo índice é 3?
- Em quase todas as linguagens de programação, o acesso a este quadrado é dado sintaticamente por “[3]”
- Considere que o nome do vetor é “m”. O que faz o comando $m[3]:=m[3]+1$?

1	2	3	4	5	6	7	8	9	10
15	12	27	23	7	2	0	18	19	21

- O comando abaixo declara a variável v com 200 longint
- A construção “1..200” é uma enumeração em Pascal. Significa que os índices para acessar o vetor são: 1,2,3,..., 199, 200;
- Observe os comandos para acessar os elementos deste vetor

Exemplos

```
1 var v: array [1..200] of longint;
```



- (a) `v[0] := 0;`
- (b) `v[1] := 0;`
- (c) `v[200] := 0;`
- (d) `v[201] := 0;`
- (e) `i:=0;`
- (f) `v[i] := 0;`
- (g) `v[i-1] := 0;`
- (h) `v[1] := v[2]+v[3];`
- (i) `v[i] := v[j]+v[k];`
- (j) `write(v[5]);`

Vetores em Pascal

- Cada comando abaixo também declara a variável *v* com 200 elementos, mas com outros índices;
- Observe os comandos para acessar os elementos de cada declaração de vetor. Quais são válidos e quais não são?

```
var v: array [0..199] of longint;
```

```
var v: array [201..400] of longint;
```

```
var v: array [-199..0] of longint;
```

```
var v: array [-300..-99] of longint;
```

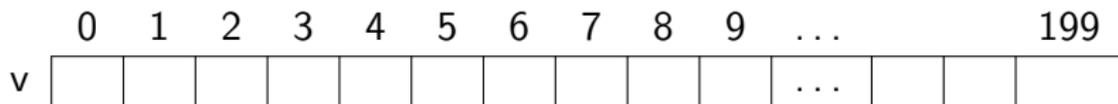
```
var v: array [-99..100] of longint;
```

```
const min=11, max=210;
```

```
var v: array [min..max] of longint;
```

Exemplos

```
1 var v: array [0..199] of longint;
```



- (a) $v[0] := 0;$ (h) $v[1] := v[2]+v[3];$
(b) $v[1] := 0;$ (i) $v[i] := v[j]+v[k];$
(c) $v[200] := 0;$ (j) $v[47] := \text{sqrt}(47);$
(d) $v[201] := 0;$ (k) $v[1] := -2;$
(e) $i:=0;$ (l*) $v[2] := 5;$
(f) $v[i] := 0;$ (m*) $v[v[1]] := v[1]+1;$
(g) $v[i-1] := 0;$ (n*) $v[v[1]-1] := 1;$

$v[2.5] := 1;$

Como funciona em memória

- Veremos como é a diferença em memória de variáveis de tipo básico em relação às variáveis do tipo vetor

Variáveis de tipo básico

```
1 var n: longint;  
2   x: real;
```

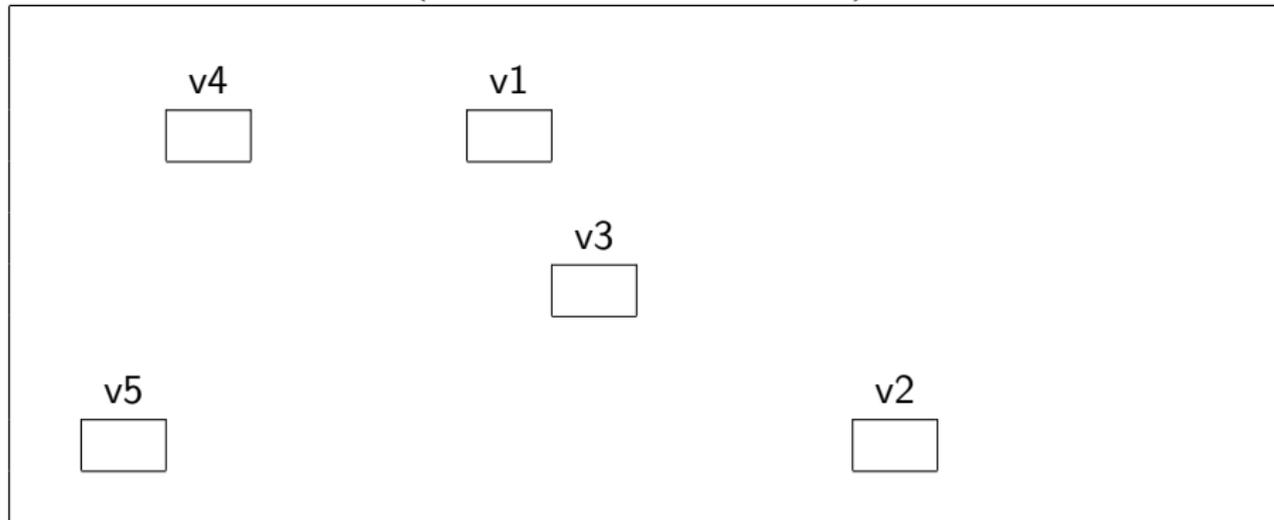
RAM (Random Access Memory)



Alocando espaço para 5 inteiros

```
1 var v1, v2, v3, v4, v5: longint;
```

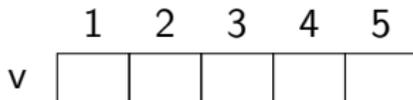
RAM (Random Access Memory)



Alocando espaço para 5 inteiros

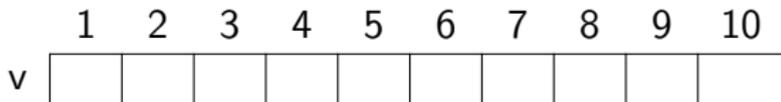
```
1 var v: array [1..5] of longint;
```

RAM (Random Access Memory)



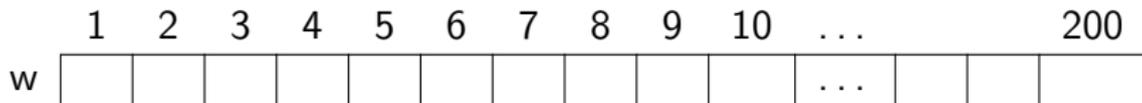
Melhor sobrar ou faltar?

```
1 var n, tam_w: longint;  
2   v: array [1..10] of longint;  
3   w: array [1..200] of longint;  
4  
5 begin  
6   tam_w:= 10;  
7   (* etc *)
```



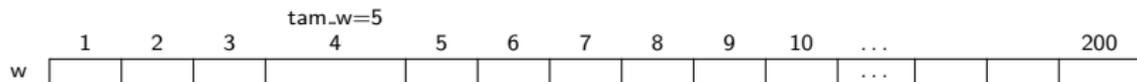
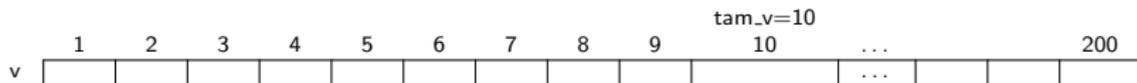
tam_v

10



O programador pode controlar diferentes vetores com tamanhos diferentes

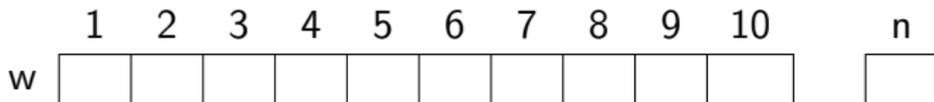
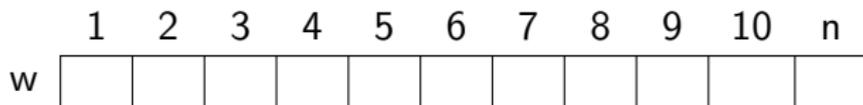
```
1 const MIN=1; MAX=200;
2 type vetor = array [MIN..MAX] of longint;
3 var tam_v, tam_w, tam_s: longint;
4     v, w, s: array [1..200] of longint;
5
6 begin
7     tam_v:= 10; tam_w:= 5; tam_s:= 0;
8     (* etc *)
```



Acessos indevidos

```
1 var n: longint;  
2   v, w: array [1..10] of longint;
```

O que ocorre se: $v[11] := 3$; ???



- O programador não pode acessar posições indevidas!
- O comportamento do programa é incerto
- Depende da linguagem e do compilador e do problema:
 - pode dar `Runtime error`
 - pode aparentemente funcionar, mas você certamente estragou alguma posição de memória

- este material está no livro no capítulo 9, seções 9.1 e 9.2

- Slides feitos em \LaTeX usando beamer
- Licença

Creative Commons Atribuição-Uso Não-Comercial-Vedada a Criação de Obras Derivadas 2.5 Brasil License.<http://creativecommons.org/licenses/by-nc-nd/2.5/br/>

Creative Commons Atribuição-Uso Não-Comercial-Vedada a Criação de Obras Derivadas 2.5 Brasil License.<http://creativecommons.org/licenses/by-nc-nd/2.5/br/>