

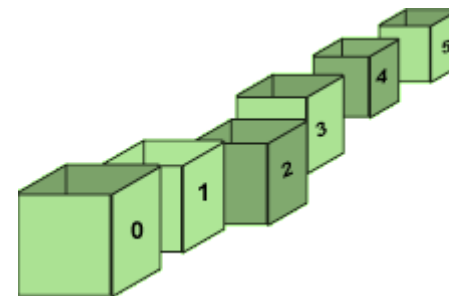
Fundamentos da Programação

Arrays

Conteúdo

- *Arrays* Unidimensionais
- *Arrays* Multidimensionais

Arrays unidimensionas



Variável vs. Vetor

- Podemos imaginar uma variável como sendo uma gaveta onde guardamos algo.

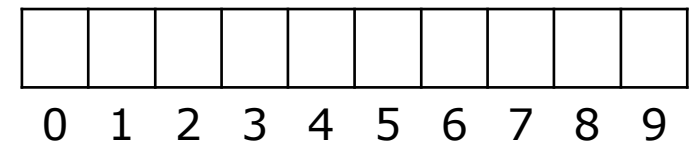


- Um vetor pode ser visto como uma **fila de gavetas**.



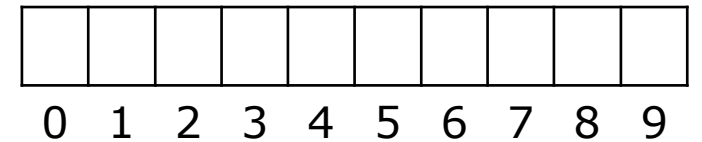
Vetor (Array)

- Conjunto de dados de um mesmo tipo (homogêneos), que são armazenados de forma contígua, e a que é possível aceder individualmente através de um índice.
- Exemplos:
 - Temperaturas médias de cada mês de um ano;
 - Comissões mensais de um vendedor;
 - Notas de um aluno a uma disciplina;
 - Movimentos de uma conta bancária.

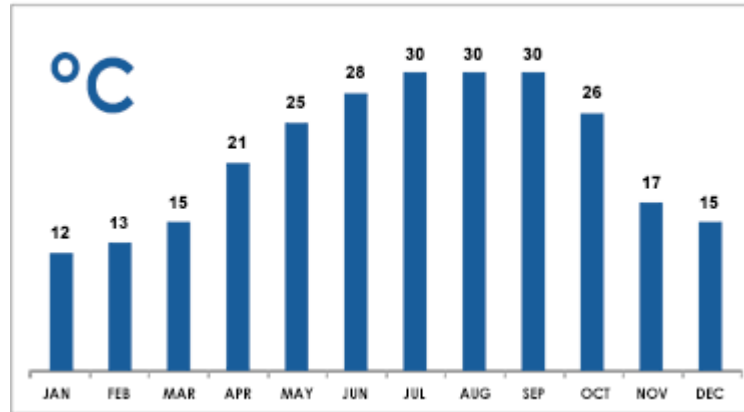


Declaração

- `tipo nome_array[numero_elementos];`
 - **tipo**: tipo de dados dos elementos (ex.: **int**, **float** ou **char**);
 - **nome_array**: nome do *array* (segue as mesmas regras de atribuição de nomes a variáveis);
 - **numero_elementos**: número de elementos (valor inteiro e positivo).
- Exemplos:
 - `unsigned short notaFinal[5];`
 - `int mes[12];`
 - `float comissao[10];`



Array



12	13	15	21	25	28	30	30	30	26	17	15
0	1	2	3	4	5	6	7	8	9	10	11

...	
15	mes[11]
17	mes[10]
26	mes[9]
30	mes[8]
30	mes[7]
30	mes[6]
28	mes[5]
25	mes[4]
21	mes[3]
15	mes[2]
13	mes[1]
12	mes[0]
...	

Manipulação

- `nome_array[índice]`
 - `nome_array`: nome do *array*;
 - `índice`: posição do elemento no *array*.
 - IMPORTANTE: Os índices de um *array* com *n* elementos variam entre **0** e **n-1**.
- Exemplos:
 - `vetor[4] = 2;`
 - `vetor[7] = 5 - 1;`
 - `vetor[0] = vetor[4] + vetor[7];`
 - `vetor[vetor[4]] = 9;`
 - `vetor[10] = 1;` ← ???

6		9		2			4		
0	1	2	3	4	5	6	7	8	9

Manipulação

- A forma mais comum de percorrer o conteúdo de um *array* é usando ciclos.

1	2	3	4	5	6	7	8	9	10
0	1	2	3	4	5	6	7	8	9

```
...  
const int ARRAY_TAM = 10;  
...  
int i, vetor[ARRAY_TAM];  
for (i = 0; i < ARRAY_TAM; i++) {  
    vetor[i] = i + 1;  
}  
...
```



```
...  
int i = 0;  
vetor[i++] = 1;  
vetor[i++] = 2;  
vetor[i++] = 3;  
vetor[i++] = 4;  
vetor[i++] = 5;  
vetor[i++] = 6;  
vetor[i++] = 7;  
vetor[i++] = 8;  
vetor[i++] = 9;  
vetor[i++] = 10;  
...
```

Manipulação



Manipulação

- Exemplo de leitura de um *array*.

```
const int TAM_ARRAY = 10;
...
int i, vetor[TAM_ARRAY];
for(i = 0; i < TAM_ARRAY; i++) {
    scanf("%d", &vetor[i]);
}
```

Particularidades

- Um *array* pode ser inicializado, na sua declaração, com o conjunto de valores que irá conter.
 - Se o número de inicializações for menor que o número de elementos do *array*, os elementos em falta serão inicializados com o valor **0**.

```
...  
int notas[3] = {12, 14, 16};  
int media[4] = {11, 13}; // equiv. a {11, 13, 0, 0}  
...
```

Arrays multidimensionas



Vetor vs. Matriz

- Se um vetor pode ser visto como uma fila de gavetas ...



- ... então uma matriz pode ser vista como **várias** filas de gavetas.




Matrizes e *Arrays* multidimensionais

- Uma matriz é um conjunto de dados de um mesmo tipo (**homogêneos**), distribuídos por **linhas e colunas**, e a que é possível aceder individualmente através de um **par de índices**.
- Uma matriz é um *array* multidimensional com 2 dimensões.

Declaração

- Sintaxe: **tipo nome[dim₁][dim₂][...][dim_n];**
 - **tipo**: tipo de dados dos elementos da matriz (ex.: **int**, **float** ou **char**);
 - **nome**: nome do vetor (segue as mesmas regras de atribuição de nomes a variáveis);
 - **[dim_x]**: tamanho da dimensão x (valor inteiro e positivo).
- Exemplos:
 - **char cubo[3][3][3];**
 - **int galo[3][3];**
 - **float tabela[4][8];**
 - **int matriz[5][5];**



0					
1					
2					
3					
4					
	0	1	2	3	4

Manipulação*

- Sintaxe: `nome_matriz[índice_linha][índice_coluna]`
 - `nome_matriz`: nome que foi dado à matriz;
 - `[índice_linha][índice_coluna]`: posição do elemento na matriz;
 - IMPORTANTE: Os índices linha e coluna de uma matriz variam entre `0` e `nº_linhas-1` e `0` e `nº_colunas-1`, respectivamente.
- Exemplos:
 - `matriz[4][3] = 2;`
 - `matriz[1][2] = 5;`
 - `matriz[3][0] = matriz[4][3] + matriz[1][2];`
 - `matriz[matriz[4][3]][4] = 6;`
 - `matriz[2][5] = 9;` ← ???

0					
1			5		
2					6
3	7				
4				2	
	0	1	2	3	4

*Exemplo com matrizes, mas semelhante em vetores com mais dimensões.

Manipulação

- A forma mais comum de percorrer o conteúdo de uma matriz é usando ciclos.

0	0	1	2	3	4
1	1	2	3	4	5
2	2	3	4	5	6
3	3	4	5	6	7
4	4	5	6	7	8
	0	1	2	3	4

```
const int TAM_MATRIZ = 5;
...
int x, y, matriz[TAM_MATRIZ][TAM_MATRIZ];
for (x = 0; x < TAM_MATRIZ; x++) {
    for (y = 0; y < TAM_MATRIZ; y++) {
        matriz[x][y] = x + y;
    }
}
```

Manipulação

- Exemplo de leitura de uma matriz.

```
const int TAM_COLUNAS 5;
const int TAM_LINHAS 3;
...
int i, j, matriz[TAM_LINHAS][TAM_COLUNAS];
for (i = 0; i < TAM_LINHAS; i++) {
    for (j = 0; j < TAM_COLUNAS; j++) {
        scanf("%d", &matriz[i][j]);
    }
}
```

Particularidades

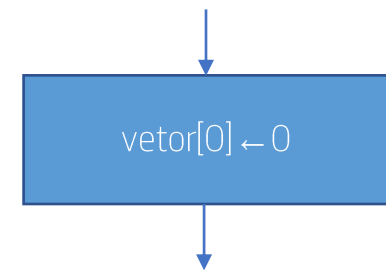
- Quer durante a compilação, quer durante a execução, não é verificado se os índices usados estão de acordo com a dimensão declarada dos vetores.
 - É por isso possível declarar um vetor com, por exemplo, 2 elementos, e tentar depois aceder ao índice 3, o que poderá levar a problemas de acesso à memória.

```
...  
int vetor[2];  
vetor[0] = 1;  
vetor[1] = 2;  
vetor[2] = 3;  
vetor[3] = 4;  
printf("valor: %d", vetor[3]);  
...
```

Vetores



```
...  
vetor[0] ← 0  
...
```



Leitura recomendada

- (Capítulo 6) Damas, L. Linguagem C; FCA – Editora de Informática, Lda, 1999; ISBN 9789727221561.

