



Universidade do Minho

Escola de Engenharia

MiETI :: Métodos de Programação II

2020/21

Apresentação

1. Introdução

António Esteves

Fevereiro 2021

Docentes

António J. Esteves

Departamento de Informática

E-mail: esteves@di.uminho.pt

Tel: 253604481

Horário de atendimento:

quinta-feira, 11h-13h, BBCU

Pedro R. Henriques

Departamento de Informática

E-mail: prh@di.uminho.pt

Tel: 968412287

Horário de atendimento:

a combinar com o docente

Tiago João F. Baptista

Departamento de Informática

E-mail: tiago96baptista@gmail.com

Horário de atendimento:

a combinar com o docente

Horário das aulas

António J. Esteves

T, quinta-feira, 9h-11h, online

PL2, segunda-feira, 14h-17h, Edifício 11 sala 1.36/online

Pedro R. Henriques

PL1, segunda-feira, 8h-11h, Edifício 11 sala 1.36/online

Tiago João F. Baptista

PL3, segunda-feira, 17h-20h, Edifício 11 sala 1.36/online

Objetivos de ensino

- Consolidar os conhecimentos sobre **algoritmia**
- **Implementar algoritmos na linguagem C**
- Cobrir **tópicos avançados** da linguagem C
- Apresentar **estruturas de dados** relevantes

Resultados de aprendizagem

- Ser capaz de **elaborar algoritmos** e
de os **implementar** numa linguagem **imperativa** como o C
- Conceber e escrever **programas** em C **estruturados**
- Perceber as **estruturas de dados** como listas, pilhas, filas e árvores e
implementar em C programas que utilizem essas estruturas

Programa detalhado

1. Introdução

- Características mais relevantes da linguagem C
- **Estrutura** dum programa em C
- Tipos de dados primários
- **Arrays** / vetores
- **Funções**
- **Organização em memória** dum programa em execução
- **Alcance** das variáveis

Programa detalhado

2. Tópicos avançados em C

- **Estruturas**
- Definição de tipos de dados com *typedef*
- Acesso a **ficheiros**
- **Alocação dinâmica** de memória
- **Apontadores:**
 - ◆ Apontadores para apontadores
 - ◆ *Arrays* de apontadores
 - ◆ Apontadores *void*

Programa detalhado

2. (continuação)

- Bibliotecas normalizadas: `stdio.h`, `ctype.h`, `stdlib.h`, `assert.h`, `stdarg.h`, `time.h`
- Utilização de programas com parâmetros
- Acesso a **ficheiros** de texto e ficheiros binários

Programa detalhado

- 3. Definição de **estruturas de dados** avançadas e respetiva implementação em C
 - Tipo de dados **sequência**: definição e algoritmos para inserir, remover, procurar
 - ◆ Caso geral (cabeça, cauda, conjunto elementos)
 - ◆ **Pilhas**
 - ◆ **Filas**
 - Listas estáticas (em *array*)
 - **Listas ligadas**
 - ◆ listas circulares, listas duplamente ligadas
 - **Árvores binárias**

Bibliografia

- António Esteves. *Sebenta de Apoio às Aulas Teóricas de Métodos de Programação II*, versão 2021
- Brian Kernighan and Dennis Ritchie. *The C Programming Language*, 2nd Edition, Prentice-Hall, 1988. ISBN: 9780131103627
- Thomas Cormen, Charles Leiserson, Ronald Rivest, and Clifford Stein. *Introduction to Algorithms*, 2nd Edition, The MIT Press, 2001
- David Harel. *Algorithmics: The Spirit of Computing*, 3rd Edition, Addison-Wesley, 2004

Metodologia de ensino

- Uma sessão teórica de 2H e uma sessão prática de 3H por semana
- As **sessões teóricas** serão essencialmente para exposição de matéria, demonstração de exemplos no computador, resolução de exercícios em papel, responder às fichas
- Nas **sessões práticas** serão aplicados os conhecimentos adquiridos nas sessões teóricas, através da elaboração de algoritmos, codificação e teste do código em C, trabalhar no projeto, avaliação do projeto
- As sessões práticas servirão ainda para esclarecer dúvidas
- As **horas de estudo não presencial** deverão ser usadas para trabalhar no projeto proposto e ler a bibliografia necessária para consolidar os conhecimentos sobre a matéria lecionada nas aulas

Metodologia de avaliação

- Questões semanais respondidas nas aulas T sobre a matéria da UC **(10%)** → **NQT**
- Dois testes escritos **(40%)** → **NT** = **(NT1+NT2)/2**
- Um projeto em **grupos de 2** elementos **(50%)** → **NP**
- Aprovação na UC exige:
 - ♦ $NP \geq 7.5$
 - ♦ $NT1 \geq 7.5$ e $NT2 \geq 7.5$ ou Exame recurso ≥ 7.5
 - ♦ $(0.1*NQT + 0.4*NT + 0.5*NP) \geq 10$
- NP = nota do projeto
- NT = média das notas nos 2 testes ou nota do exame recurso

Sessões laboratoriais

- Devem ter acesso a computador pessoal nas aulas PL e T
- Registrar-se em <https://codeboard.io>
- Preparar o ambiente de desenvolvimento de acordo com o documento disponível na pasta “Aulas Práticas”
- Resolução de exercícios nas aulas:
 - ◆ Os exercícios do guião fornecido e outros exercícios extra a fornecer pontualmente
 - ◆ Cada exercício será apresentado e explicado
 - ◆ Posteriormente os alunos terão um tempo máximo para o resolver
 - ◆ A solução não será fornecida
- As aulas também são para **dar apoio ao projeto** e para o docente **avaliar a evolução do projeto** (em especial após as entregas definidas no enunciado do projeto)

Planeamento das aulas

Dia	Prática	Dia	Teórica
		18/fev	Tipos de dados primários e <i>arrays</i>
22/fev	Inteiros e <i>arrays</i> de caracteres	25/fev	Funções
01/mar	<i>Arrays</i> e <i>strings</i>	04/mar	Estruturas e <i>arrays</i> de estruturas
08/mar	<i>Arrays</i> e estruturas	11/mar	Acesso a ficheiros
15/mar	<i>Arrays</i> e ficheiros (AV1)	18/mar	Apontadores e <i>arrays</i> multi-dimensionais
22/mar	Vetores e matrizes	25/mar	Memória dinâmica
páscoa	páscoa	páscoa	páscoa
páscoa	páscoa	08/abr	TESTE 1
12/abr	Apontadores e mult matrizes	15/abr	Listas ligadas
19/abr	Listas ligadas	22/abr	Pilhas e filas
26/abr	Listas ligadas (AV2)	29/abr	Árvores e árvores binárias
03/mai	Listas ligadas	06/mai	Árvores binárias de pesquisa
10/mai	Árvores binárias	13/mai	Árvores binárias de pesquisa equilibradas
17/mai	Árvores binárias	20/mai	Resolução de teste modelo e revisões
24/mai	Trabalho no projeto	27/mai	TESTE 2
31/mai	(AV3 → ajustar ao teste de SD)		

1. Introdução

- Características mais relevantes da linguagem C
- Estrutura dum programa em C
- Tipos de dados primários: **int**, **float**, **double**, **char**, **void**
- *Arrays* (ou vetores)
- Funções
- Organização em memória dum programa em execução
- Alcance das variáveis

Caraterísticas mais relevantes da linguagem C


- Utiliza poucas **palavras chave** → `if`, `else`, `for`, `while`, `do`, `return`, `switch`, `case`, `break`, `continue`, `struct`, `union`, `int`, ...
- Suporta tipos de **dados compostos** → `struct`, `union`
- **Apontador** → endereço de (memória onde está guardada) uma variável)
- **Bibliotecas externas** normalizadas → entrada/saída, funções matemáticas, acesso à memória, outras facilidades
- Utilizada em conjunto com um **pré-processador** de diretivas → `#define`, `#include`, ...

Caraterísticas relevantes da linguagem C

- O código C é
 - ◆ “**rápido**” ?! como é compilado, é por natureza executado mais rapidamente que código equivalente interpretado
 - ◆ **Compacto** ⇔ pouco verboso
 - ◆ de uso **genérico**
 - ◆ maioritariamente **independente da arquitetura alvo**
- O C é utilizado para programar sistemas muito diversos
 - ◆ compiladores
 - ◆ sistemas operativos
 - ◆ sistemas embebidos, micro-controladores, ...
 - ◆ computação genérica

Caraterísticas relevantes da linguagem C

O C é uma linguagem ...

- com verificação **estática** do **tipo** das variáveis → efetuada pelo compilador
 - ◆ vs. dinâmica → efetuada durante a execução
- com **fraca segurança nos tipos** das variáveis → permite atribuir valores/expressões de tipo errado a variáveis
- incentiva a **programação estruturada**
 - ◆ uso extensivo de construtores de controlo do fluxo, de repetição e de funções
 - ◆ melhor clareza, qualidade, tempo de desenvolvimento
- **imperativa** 

Linguagem de programação imperativa

- Descreve a computação como uma **sequência de instruções**, ou comandos, a executar pelo computador
- As instruções **mudam o estado** do programa
- **estado** ⇔ definido pelos **valores das variáveis**
- É análoga ao comportamento imperativo das linguagens naturais, em que se dão ordens:

faz um telefonema
↓
a seguir faz café
↓
depois envia um e-mail

Estrutura dum programa em C

Seção de documentação → `// exemplo de programa`

Seção para ligação → `#include <stdio.h>`

Seção de definições → `#define ESCALA 4`

Seção de declarações globais → `void minhaFuncao();`
`int v = 10;`

Seção da função main() → `void main()`

{

Parte com declarações

→ `int k = 5;`

Parte executável

→ `v = v + k * ESCALA;`
`minhaFuncao();`
`}`

}

Seção de subprogramas

Função 1

Função 2

.....

.....

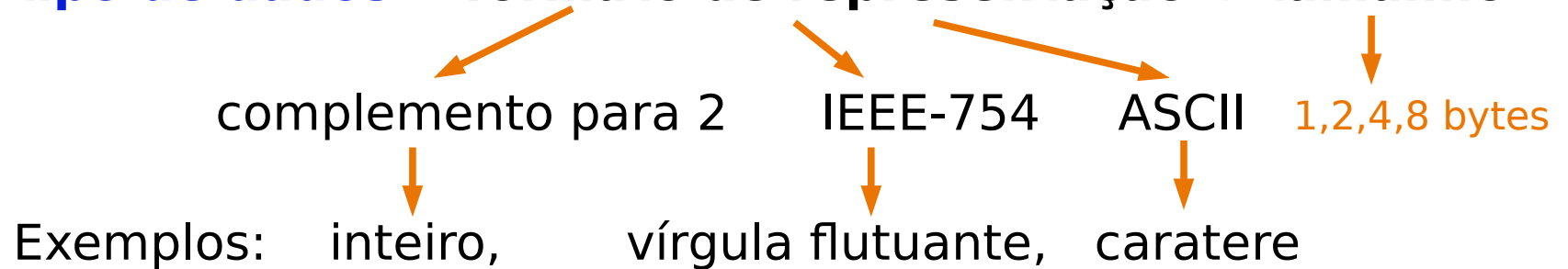
Função N

(Funções
definidas
pelo
programador)

→ `void minhaFuncao()`
`{`
`printf("valor de v: %d\n", v);`
`}`

Tipos de dados do C

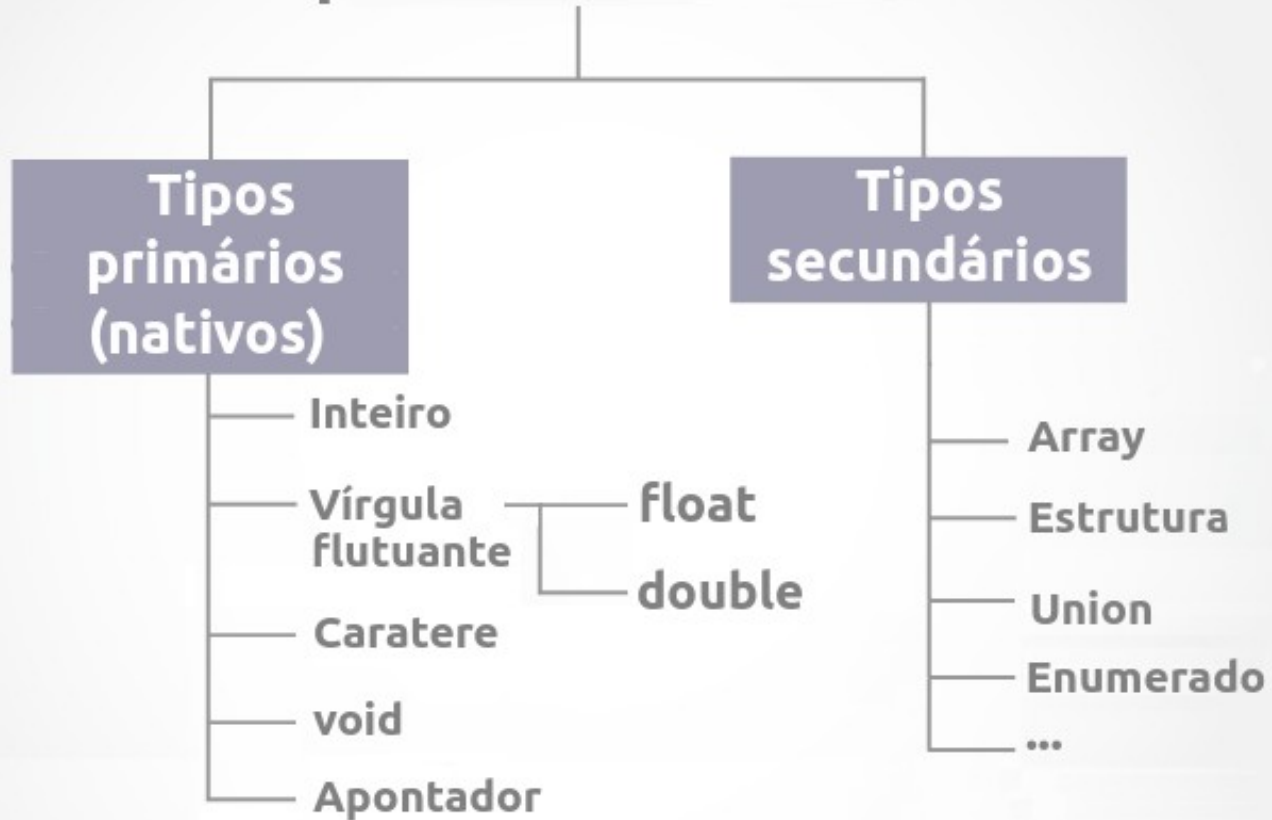
tipo de dados \equiv **formato de representação** + **tamanho**



Os tipos de dados utilizam-se:

- Na declaração/definição de variáveis
- Nos argumentos das funções
- No valor devolvido pelas funções

Tipos de dados do C



Tipos de dados primários

DECLARAÇÃO DE VARIÁVEL DO TIPO int:

```
int Contador;
```

ATRIBUIÇÃO DE UM VALOR À VARIÁVEL:

```
Contador = 5;
```

DECLARAÇÃO DE VARIÁVEL DO TIPO float:

```
float distancia;
```

ATRIBUIÇÃO DE UM VALOR À VARIÁVEL:

```
distancia = 4.75;
```

DECLARAÇÃO DE VARIÁVEL DO TIPO char:

```
char letra;
```

ATRIBUIÇÃO DE UM VALOR À VARIÁVEL:

```
letra = 'w';
```

FUNÇÃO COM VALOR DEVOLVIDO DO TIPO void:

```
void printMensagem(char *);
```

FUNÇÃO COM ARGUMENTO DO TIPO void:

```
double randomNumber(void);
```

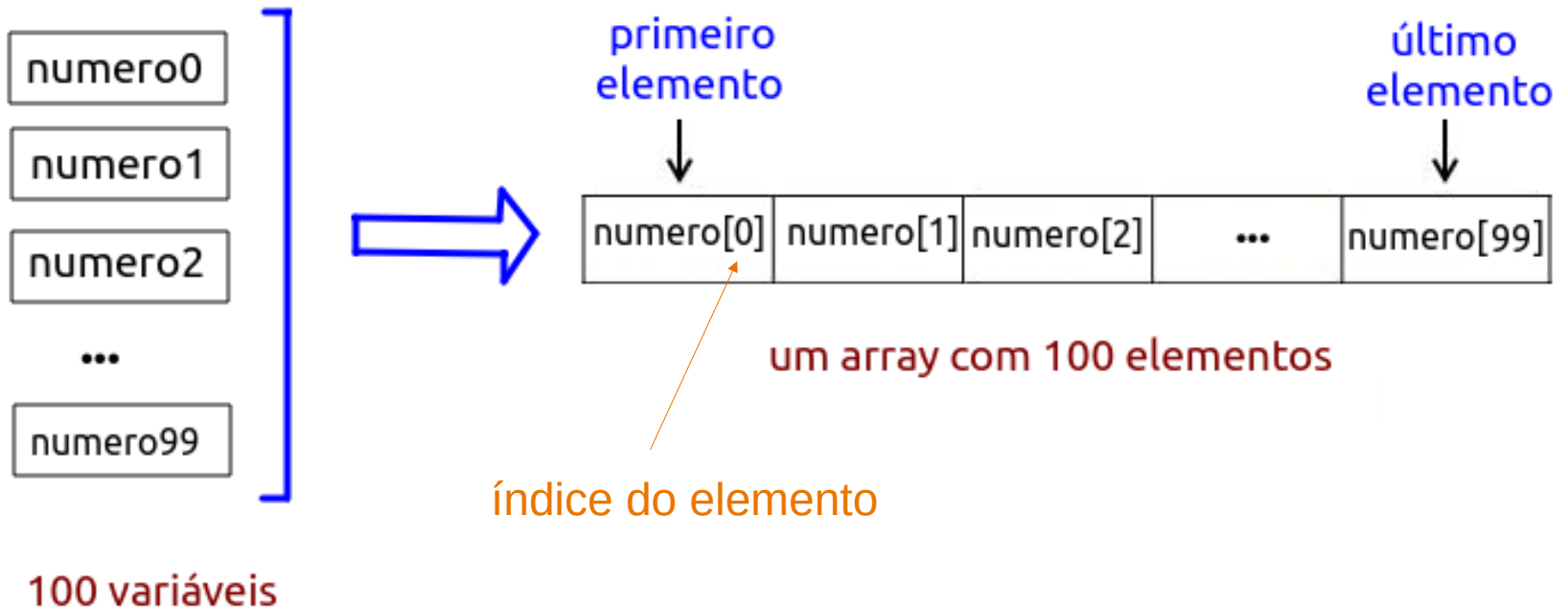
Qualificadores do C

- Alteram o significado dos tipos de dados
 - ↳ produzem novos tipos de dados
- Qualificadores de tamanho → **long**
→ **short**
- Qualificadores de sinal → **signed**
(só para inteiros) → **unsigned**
- Qualificador de constante → **const**

<pre>long double lf; long int li; short int si;</pre>	<pre>unsigned int peso; unsigned short int idade;</pre>
<pre>const float fConversao = 3.7; const int fEscala = 5; fEscala = 6; ←-- ERRO!!</pre>	<pre>codeboard.io/projects/217194/</pre>

Arrays ou Vetores

- Motivação para utilizar **arrays**:



- **Array** →
 - uma sequência de elementos do mesmo tipo
 - número de elementos fixo
 - elementos acessados através de um índice

Arrays ou Vetores

- Sintaxe da **declaração** dum array:

```
tipo nomeArray [ tamanhoArray ]; // genérica
double saldo[5]; // um exemplo
```

- **Inicialização** dum array:

```
saldo[0] = 1000.0; // 1) inicializar um a um
```

```
saldo[1] = 2.0;
```

```
saldo[2] = 3.4;
```

```
saldo[3] = 7.0;
```

```
Saldo[4] = 50.0;
```

```
// 2) inicializar na declaração
```

```
double saldo[5] = {1000.0, 2.0, 3.4, 7.0, 50.0};
```

	0	1	2	3	4
saldo	1000.0	2.0	3.4	7.0	50.0

Como fazemos a inicialização quando o *array* é muito grande?

Arrays ou Vetores

- **Acesso aos elementos** dum *array*:

Colocamos o índice do elemento pretendido entre parênteses retos **[]**

```
double saldoMes;
```

```
saldoMes = saldo[2];
```

Arrays bi-dimensionais

- Exemplo de um *array* bi-dimensional \Leftrightarrow uma matriz:

```
int matriz[10][4];
```

matriz que guarda 10 *arrays*, cada *array* com 4 elementos

- Inicializar um *array* bi-dimensional aquando da declaração:

```
int matriz[4][2] =  
    { {1, 2},  
      {3, 4},  
      {5, 6},  
      {7, 8} };
```

Arrays bi-dimensionais

- Conteúdo do *array* `matriz[4][2]` após inicialização:

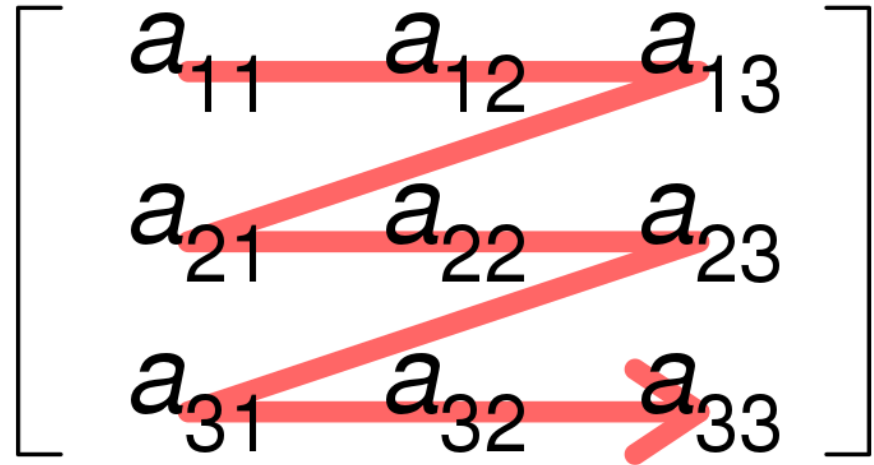
		coluna →			
		0	1	0	1
linha ↓	0	<code>matriz[0][0]</code>	<code>matriz[0][1]</code>	0	1
	1	<code>matriz[1][0]</code>	<code>matriz[1][1]</code>	1	4
	2	<code>matriz[2][0]</code>	<code>matriz[2][1]</code>	2	6
	3	<code>matriz[3][0]</code>	<code>matriz[3][1]</code>	3	8

Como é guardado o *array* `matriz[4][2]` em memória?

Arrays bi-dimensionais em memória

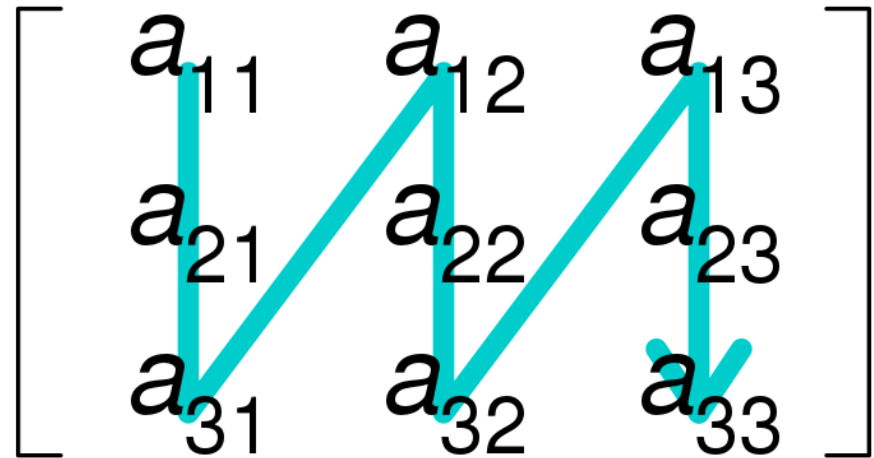
- **Em C: row-major order**

Elementos de cada linha
são armazenados em
posições contíguas na
memória



- **Fortran: column-major order**

Elementos de cada coluna
são armazenados em
posições contíguas na
memória



Arrays bi-dimensionais

- Como em C os elementos de cada linha são armazenados em posições contíguas na memória, a **matriz** é assim guardada:

```
int matriz[4][2] =  
    { {1, 2},  
      {3, 4},  
      {5, 6},  
      {7, 8} };
```

