TI0044 – Técnicas de Programação para Engenharia I (TPE I)
by prof. Alexandre M Moraes
DETI / CT / UFC
ASSUNTO: Arrays

1. Arrays

1.1 Definição: Coleção de elementos, sendo que <u>todos</u> devem ser do mesmo tipo.

1.2 Características:

1.2.1 Tipo de cada elemento: números inteiros, números em ponto flutuante, estruturas, ponteiros etc.

1.2.2 Dimensão: está relacionada com a quantidade de elementos.

1.2.3 Número de dimensões

1.2.4 Índices: valores inteiros que representam a posição do elemento no *array*.

1.3 Classificação quanto ao número de dimensões:

1.3.1 *Array* Unidimensional (uma dimensão – vetor)

$$ARRAY1D_2 = [10 \quad 20]$$

1.3.1.1 Declaração de variável(eis):

```
tipo identificador[dimensão];
```

1.3.1.2 Exemplo:

```
int array1d[2];
```

OBS.: Não é possível declarar um vetor tendo como índice uma variável recém-declarada não inicializada. Exemplo:

```
int dimension, array1d [dimension]; /* ERRADO */
```

1.3.1.3 Índices: os índices só deverão variar entre `0` e `dimensão-1`. Se algum índice estiver fora desta faixa, seu programa fatalmente irá usar outras regiões de memória talvez já reservadas para outras variáveis.

1.3.1.4 Inicialização (que só pode ser feita junto com a declaração). Exemplo:

```
int array1d [2] = {10, 20};
```

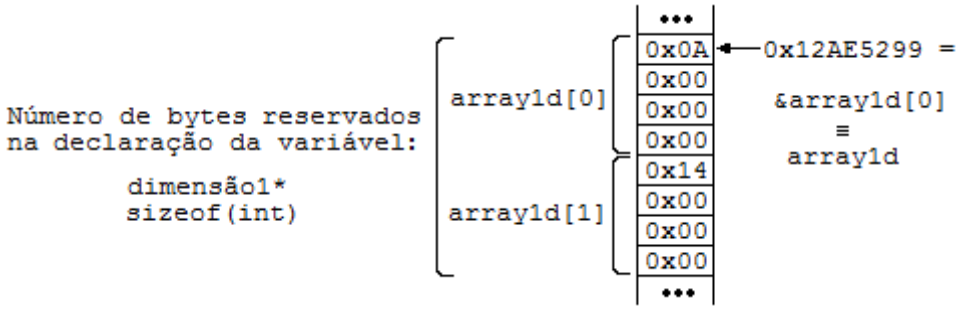
ou

```
int array1d [] = {10, 20}; /* CERTO TAMBÉM */
```

OBS.: Não é possível declarar um vetor sem dimensão, sem inicializá-lo. Exemplo:

```
int array1d []; /* ERRADO */
```

1.3.1.5 Modelo na memória. Exemplo:



OBS.1: Notar que os elementos que são do tipo `int` (4 *bytes*), e são representados na memória na notação *little endian* – o *byte* menos significativo (LSB) do `int` é colocado no MENOR endereço da memória reservada para os 4 bytes. Se fossem representados na notação *big endian*, o *byte* mais significativo (MSB) do `int` seria colocado no MENOR endereço da memória reservada para os 4 bytes.

OBS.2: Quando o identificador do *array* unidimensional (no exemplo, `vet`) aparecer no código fonte (após ter sido declarado) sem nenhuma referência a algum de seus elementos (ou seja, sem os colchetes e um índice), este identificador representa então um valor constante igual ao endereço do primeiro *byte* do primeiro elemento do *array* unidimensional (no exemplo, `array1d = 0x12AE5299`, pode ser considerado como "o endereço do primeiro elemento", ou simplesmente, "o endereço do vetor").

1.3.1.6 Só acredito vendo...

```
#include <stdio.h>
#include <stdlib.h>
```

```
#define DIMENSION1 2

int main() {
  int array1d[DIMENSION1] = {10, 20};
  int i;

  printf("sizeof(array1d) = %d bytes\n\n", sizeof(array1d));

  printf("Endereco de cada elemento:\n");
  for(i = 0; i < DIMENSION1; i++)
    printf("&array1d[%d] = %p\n", i, &array1d[i]);

  printf("\nEndereco especial:\n");
  printf("array1d     = %p\n", array1d);

  return 0;
}
```

1.3.2 *Array* Bidimensional (duas dimensões – matriz)

$$ARRAY2D_{2x2} = \begin{bmatrix} 10 & 20 \\ 30 & 40 \end{bmatrix}$$

1.3.2.1 Declaração de variável(eis):

```
tipo identificador[dimensão1][domensão2];
```

OBS.: Para que a matriz (que representa uma tabela) possa ser também representada no modelo de memória adotado (ou seja, as linhas da tabela são colocadas uma depois da outra na memória, começando pela primeira linha). A dimensão 1 representa então o número de linhas e a dimensão 2 representa o número de colunas por linha.

1.3.2.2 Índices: os índices só deverão variar entre `0` e `dimensão1-1` para as linhas e `0` e `dimensão2-1` para as colunas. Valem aqui também a mesma observação caso algum dos índices fiquem fora das faixas previstas pelas dimensões.

1.3.2.3 Exemplo:

```
short array2d[2][2];
```

OBS.: Não é possível declarar uma matriz tendo como índices variáveis recém-declaradas não inicializadas. Exemplo:

```
short row, col, array2d [row][col]; /* ERRADO */
```

1.3.2.4 Inicialização (que só pode ser feita junto com a declaração). Exemplo:

```
short array2d [2][2] = {{10, 20}, {30, 40}};
```

ou

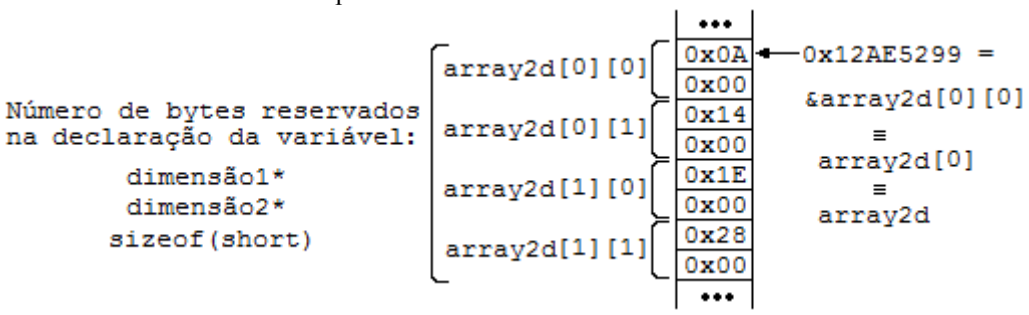
```
short array2d [][2] = {{10, 20}, {30, 40}}; /* CERTO TAMBÉM */
```

OBS1.: Notar que mesmo durante a declaração da variável é possível inicializar a matriz, mesmo que falte a dimensão da quantidade de linhas. Porém, jamais poderia faltar a dimensão da quantidade de colunas.

OBS2.: Não é possível declarar uma matriz sem as duas dimensões, sem inicializá-la. Exemplo:

```
short array2d [][]; /* ERRADO */
```

1.3.2.5 Modelo na memória. Exemplo:



OBS.1: Notar que um *array* bidimensional (matriz) poder ser visto com um *array* unidimensional (vetor) de *arrays* unidimensionais (vetores), ou seja, como um "vetor de vetores", onde cada elemento do vetor também é um vetor. Complicado???

OBS.2: Quando o identificador do *array* bidimensional (no exemplo, `mat`) aparecer no código fonte (após ter sido declarado) sem nenhuma referência a algum de seus elementos (ou seja, sem os colchetes e os dois índices), este identificador representa então um valor constante igual ao endereço do primeiro *byte* da primeira coluna da primeira linha do *array* bidimensional (no exemplo, `array2d = 0x12AE5299`, pode ser considerado como "o endereço do primeiro elemento", "o endereço da primeira linha" ou simplesmente, "o endereço da matriz").

OBS.3: Quando o identificador do *array* bidimensional aparecer no código com apenas uma das suas dimensões entre colchetes, esta forma representa então um valor constante igual ao endereço da <u>linha</u> indicada pelo índice. Do exemplo:

`array2d[0]` ≡ endereço da primeira linha (linha 0)

`array2d[1]` ≡ endereço da segunda linha (linha 1)

1.3.2.6 Só acredito vendo...

```
#include <stdio.h>
#include <stdlib.h>

#define DIMENSION1 2
#define DIMENSION2 2

int main() {
  short array2d[DIMENSION1][DIMENSION2] = {{10, 20}, {30, 40}};
  int i, j;

  printf("sizeof(array2d) = %d bytes\n\n", sizeof(array2d));

  printf("Endereco de cada elemento:\n");
  for(i = 0; i < DIMENSION1; i++)
    for(j = 0; j < DIMENSION2; j++)
      printf("&array2d[%d][%d] = %p\n", i, j, &array2d[i][j]);

  printf("\nEnderecos especiais:\n");
  printf("array2d        = %p\n", array2d);
  printf("array2d[0]     = %p\n", array2d[0]);
  printf("array2d[1]     = %p\n", array2d[1]);

  return 0;
}
```

1.3.3 *Array* N-Dimensional (n > 2, sem um nome específico)

$$ARRAY3D_{2x2x2} = \begin{bmatrix}\begin{bmatrix}'A' & 'B'\\ 'C' & 'D'\end{bmatrix}\begin{bmatrix}'E' & 'F'\\ 'G' & 'H'\end{bmatrix}\end{bmatrix}$$

1.3.3.1 Declaração de variável(eis) – Exemplo com dimensão = 3:

```
tipo identificador[dimensão1][domensão2][dimensão3];
```

OBS.: Para que o *array* possa ser também representado no modelo de memória adotado, a `dimensão1` representa então o número de planos, a `dimensão2` representa então o número de linhas por plano e a `dimensão3` representa o número de colunas por linha.

1.3.3.2 Índices: os índices só deverão variar entre `0` e `dimensão1-1` para os planos, `0` e `dimensão2-1` para as linhas e `0` e `dimensão3-1` para as colunas. Valem aqui também a mesma observação caso algum dos índices fiquem fora das faixas previstas pelas dimensões.

1.3.3.3 Exemplo:

```
char array3d[2][2][2];
```

OBS.: Não é possível declarar uma matriz tendo como índices variáveis recém-declaradas não inicializadas. Exemplo:

```
char pla, row, col, array3d[pla][row][col]; /* ERRADO */
```

1.3.3.4 Inicialização (que só pode ser feita junto com a declaração). Exemplo:

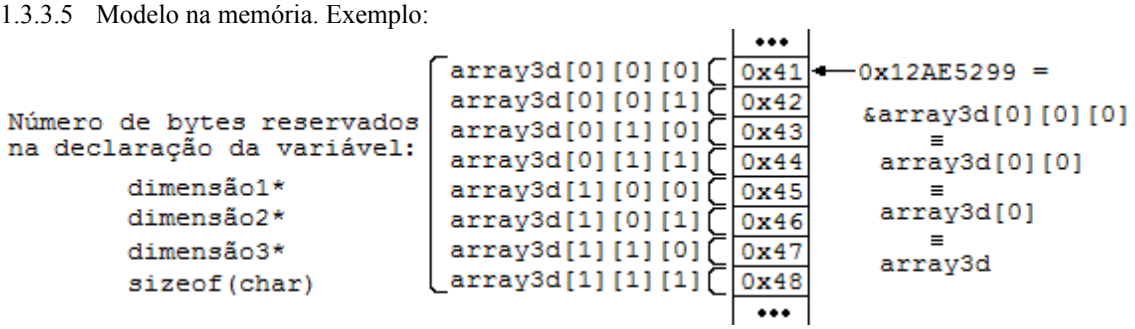
```
char array3d[2][2][2] = {{{'A','B'},{'C','D'}},{{'E','F'},{'G','H'}}};
                                ou
char array3d[][2][2] = {{{'A','B'},{'C','D'}},{{'E','F'},{'G','H'}}};
                          /* CERTO TAMBÉM */
```

OBS1.: Notar que mesmo durante a declaração da variável é possível inicializar o *array*, mesmo que falte a dimensão da quantidade de planos. Porém, jamais poderia faltar a dimensão da quantidade de colunas e da quantidade de linhas.

OBS2.: Não é possível declarar um *array* sem as três dimensões, sem inicializá-lo. Exemplo:

```
char array3d[][][]; /* ERRADO */
```

1.3.3.5 Modelo na memória. Exemplo:



1.3.3.6 Só acredito vendo...

```
#include <stdio.h>
#include <stdlib.h>

#define DIMENSION1 2
#define DIMENSION2 2
#define DIMENSION3 2

int main() {
  char array3d[DIMENSION1][DIMENSION2][DIMENSION3] =
        {{{'A','B'},{'C','D'}},{{'E','F'},{'G','H'}}};
  int i, j, k;

  printf("sizeof(array3d) = %d bytes\n\n", sizeof(array3d));

  printf("Endereco de cada elemento:\n");
  for(i = 0; i < DIMENSION1; i++)
    for(j = 0; j < DIMENSION2; j++)
      for(k = 0; k < DIMENSION3; k++)
      printf("&array3d[%d][%d][%d] = %p\n", i, j, k, &array3d[i][j][k]);

  printf("\nEnderecos especiais:\n");
  printf("array3d           = %p\n", array3d);
  printf("array3d[0]        = %p\n", array3d[0]);
  printf("array3d[0][0]     = %p\n", array3d[0][0]);
  printf("array3d[0][1]     = %p\n", array3d[0][1]);
  printf("array3d[1]        = %p\n", array3d[1]);
  printf("array3d[1][0]     = %p\n", array3d[1][0]);
  printf("array3d[1][1]     = %p\n", array3d[1][1]);

  return 0;
}
```

OBS.: Como pode se notar, a ideia pode ser estendida para dimensões n > 3, é só entender como os *arrays* uni, bi e tridimensionais são representados pela linguagem. E a forma como eles devem ser representados no modelo de memória.

EXERCÍCIOS: Utilizando como base os códigos fontes apresentados, crie novos programas variando os tipos, e as dimensões dos novos *arrays* e tente verificar se os resultados apresentados batem com aqueles que você previa, antes mesmo de rodar os programas. Só assim você vai aos poucos obtendo confiança para manipular *arrays* com tranquilidade.