INSTITUTO FEDERAL
Paraná
Campus Paranaguá

Ministério da Educação

ENSINO MÉDIO INTEGRADO - INFORMÁTICA
Disciplina de Linguagem de Programação
Aula 10: Matriz

Gil Eduardo de Andrade

Conceitos Preliminares

(https://docs.microsoft.com/pt-br/cpp/c-language/?view=msvc-170)

Introdução

Uma matriz pode ser definida, assim como um vetor, como um conjunto de elementos (variáveis) que apresentam um mesmo tipo e são referenciados por um nome comum. As matrizes em C, diferentemente dos vetores que possuem apenas uma dimensão, podem apresentar duas ou mais dimensões.

Matrizes Bidimensionais

Normalmente, quando trabalhamos com matrizes em C, elas são bidimensionais, ou seja, possuem duas dimensões, as quais denominamos linhas e colunas. Os elementos que compõem uma matriz bidimensional são acessados, necessariamente, através da especificação de dois índices inteiros, que correspondem, respectivamente, a linha e a coluna onde esse elemento se encontra dentro da matriz.

A declaração de matrizes bidimensionais é efetuada, de modo geral, da seguinte maneira:

```
tipo nome_matriz[nr_linhas][nr_colunas];
```

- **tipo:** indica qual é o tipo das variáveis que compõem a matriz. Exemplo: ***int***, ***float***, ***double***;
- **nome_matriz:** indica o nome pelo qual a matriz é referenciada;
- **nr_linhas:** indica o número total de linhas que a matriz possui;
- **nr_colunas:** indica o número total de colunas que a matriz possui;

Sintaxe e Utilização

Matrizes Bidimensionais Inteiras

Considerando as informações apresentadas anteriormente, podemos declarar uma matriz do tipo **int** contendo **8 linhas** e **8 colunas**. Teríamos:

```
int tabuleiro[8][8];
```

Ao declararmos a matriz tabuleiro garantimos que o espaço de memória necessário para armazenar todos os seus elementos seja reservado. Contudo, é importante observar que apenas declaramos a matriz "tabuleiro", ou seja, ainda não especificamos os elementos (valores) que serão armazenados nela.
Antes desta especificação, dizemos que a matriz possui ***"lixo de memória"*** armazenado em suas linhas e colunas, que são valores quaisquer gerados por outros programas que utilizaram este mesmo espaço de memória num momento anterior.

Armazenando Valores numa Matriz Bidimensional Inteira

Como mencionado, as matrizes bidimensionais possuem linhas e colunas. Sendo assim, elas podem armazenar um total de ***"linhas x colunas"*** elementos.

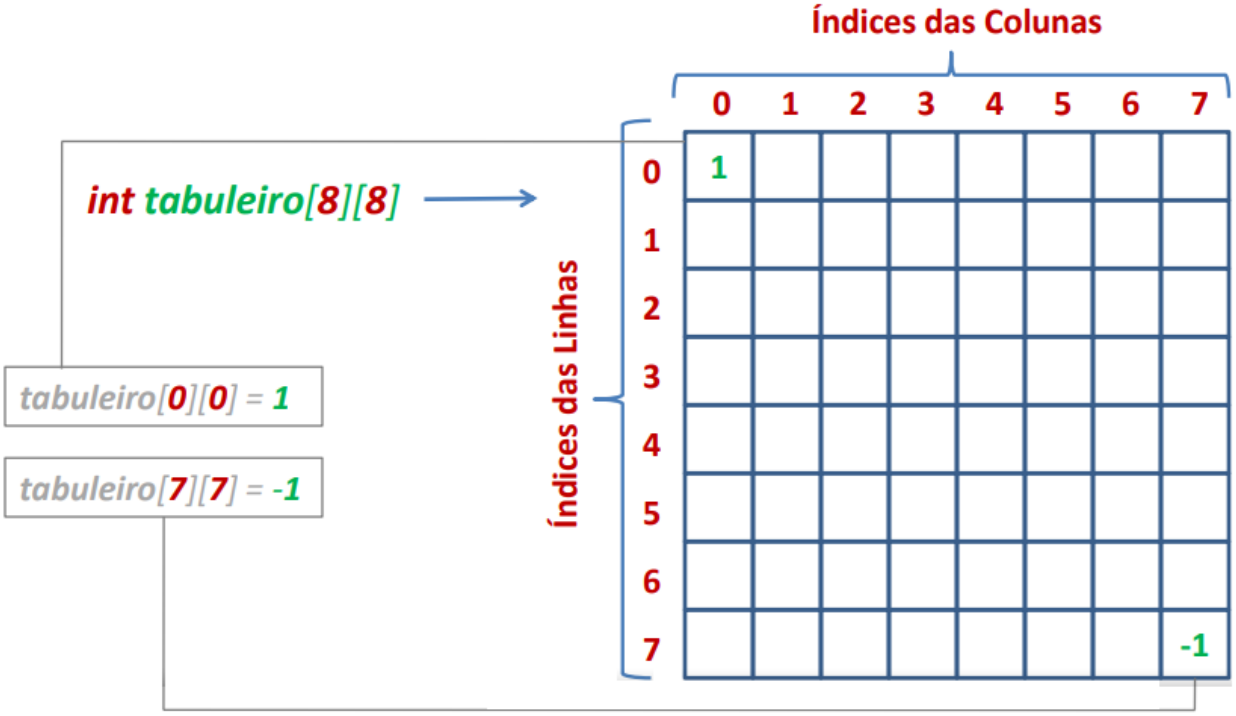
Ex.: uma matriz com ***8 linhas*** por ***8 colunas*** pode armazenar ***64 elementos*** – ***8 x 8***;

Para que seja possível especificar qual elemento da matriz inteira receberá um valor a ser armazenado, utilizamos dois índices (valores inteiros) que indicam a posição (linha, coluna) da matriz que receberá o dado.
Os índices da matriz, tanto em linha como em coluna, variam de ***"0"*** até seu ***"número de linhas - 1"*** ou ***"número de colunas - 1"***. Em outras palavras, a matriz **int tabuleiro[8][8]** possui os índices: **0**, **1**, **2**, **3**, **4**, **5**, **6** e **7**.
Para armazenar dados na matriz **int tabuleiro[8][8]** utilizamos a seguinte sintaxe:

```
tabuleiro[0][0] = 1                    tabuleiro[7][7] = -1
```

Representação Gráfica

Uma matriz, em programação, pode ser representada como uma planilha, já que esta contém linhas e colunas. Cada célula desta planilha é referenciada por dois índices, um indicando a linha e outro a coluna onde encontra-se o elemento que desejamos manipular.



Matrizes x Strings

Assim como visto na aula de vetores (String) é possível trabalharmos com matrizes de caracteres (char). Sendo assim, podemos considerar que uma matriz de caracteres permite armazenar mais de uma String. Torna-se possível trabalharmos com o método ***gets()*** para armazenar strings em linhas individuais de uma matriz do tipo char.

Ao solicitarmos que o usuário digite uma string (por exemplo, um nome) utilizamos a função ***gets()***, passando a ela qual das linhas da matriz deve armazenar a string digitada;

```
char nomes[3][20];
gets(nomes[0]);
```

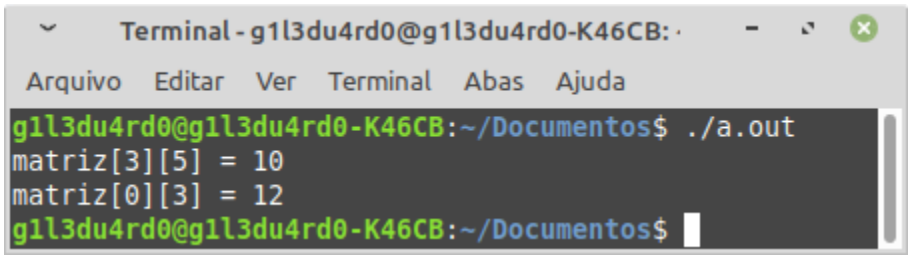
Codificação – Linguagem de Programação C

Declarando e Atribuindo Valores - Matriz Bidimensional Inteira

```c
#include <stdio.h>

#define LIN 5

#define COL 7

int main() {

   // declara uma matriz com 5 linhas e 7 colunas

   int matriz[LIN][COL];

   // armazena o valor 10 a linha 3, coluna 5 da matriz

   matriz[3][5] = 10;

   // armazena o valor 12 na linha 0, coluna 3 da matriz

   matriz[0][3] = 12;

   // exibe os valores armazenados anteriormente

   printf("matriz[3][5] = %i", matriz[3][5]);

   printf("\nmatriz[0][3] = %i", matriz[0][3]);

   printf("\n");

   return 0;

}
```

Atribuindo Valores - Matriz Bidimensional Inteira.

(Arquivo-fonte: 11 - Matriz/atribuindo_valor_inteiro.c)



Resultado da execução do código anterior

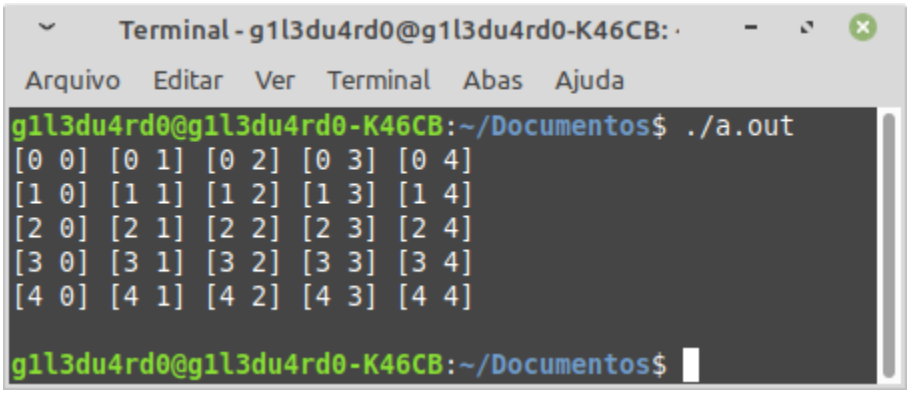
Exibindo os Índices (linha / coluna) da Matriz

```c
#include <stdio.h>

#define LIN 5

#define COL 7

int main() {

   // declara os índices

   int lin, col;

   // declara uma matriz com 5 linhas e 7 colunas

   int matriz[LIN][COL];

   // percorre a matriz

   for(lin=0; lin<LIN; lin++) {

       for(col=0; col<LIN; col++) {

           // apresenta os índices (linha/coluna)

           printf("[%i %i] ", lin, col);

       }

       // pula para a próxima linha

       printf("\n");

   }

   printf("\n");
```

```c
    return 0;
}
```

Exibindo os índices - Matriz.

(Arquivo-fonte: 11 - Matriz/exibindo_indices.c)



Resultado da execução do código anterior

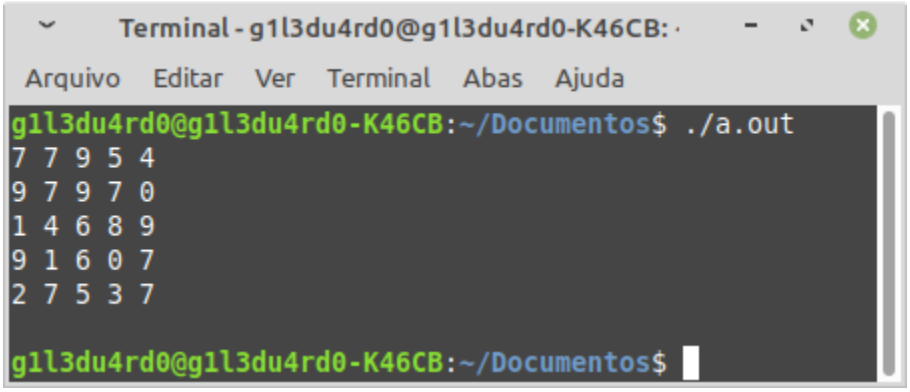
Preenchendo Matriz Inteira - Números Aleatórios

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define LIN 5
#define COL 7
int main() {
   // declara os índices para percorrer a matriz
   int lin, col;
   // declara uma matriz com 5 linhas e 7 colunas
   int matriz[LIN][COL];
   // define semente para geração dos nrs aleatórios
   srand(time(NULL));
```

```c
    // percorre a matriz
    for(lin=0; lin<LIN; lin++) {
        for(col=0; col<LIN; col++) {
            // armazena os valores aleatórios: 0-9
            matriz[lin][col] = rand()%10;
            // apresenta os valores armazenados
            printf("%i ", matriz[lin][col]);
        }
        // pula para a próxima linha
        printf("\n");
    }
    printf("\n");
    return 0;
}
```

Preenchendo Matriz - Números Aleatórios.

(Arquivo-fonte: 11 - Matriz/preenchendo_inteira.c)



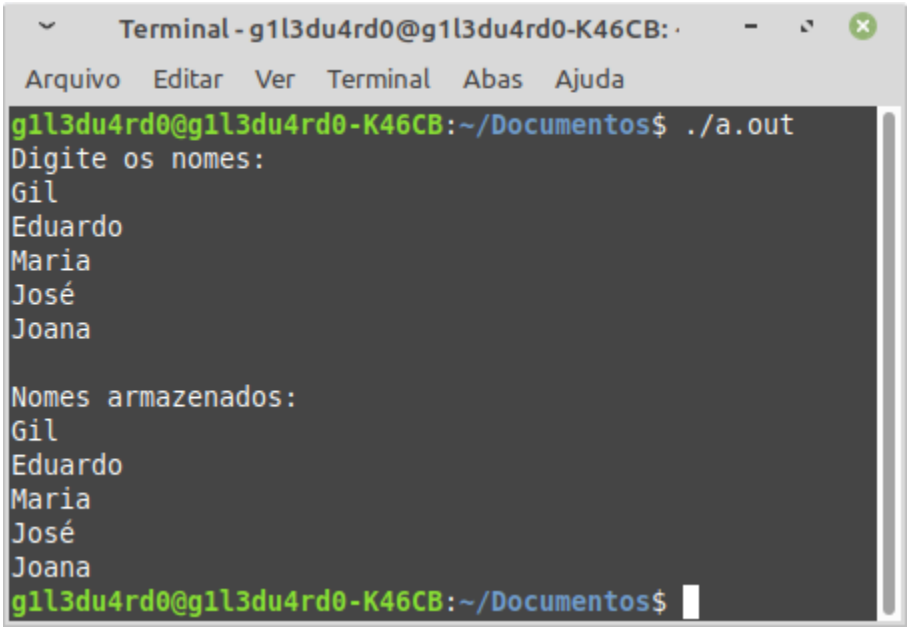
Resultado da execução do código anterior

Armazenando Nomes - Matriz String

```c
#include <stdio.h>
#include <stdio_ext.h>
#include <stdlib.h>
#define LIN 5
#define COL 30
int main() {
   // declara o índice para as linhas
   int lin;
   // declara uma matriz com 5 linhas e 30 colunas
   char nomes[LIN][COL];
   printf("Digite os nomes:\n");
   // percorre as linhas matriz
   for(lin=0; lin<LIN; lin++) {
       // limpa o buffer de entrada
       __fpurge(stdin);
       // captura e armazena o nome na matriz "nomes"
       gets(nomes[lin]);
   }
   printf("Nomes armazenados:");
   // percorre as linhas matriz
   for(lin=0; lin<LIN; lin++) {
       // Exibe os nomes
       printf("\n%s", nomes[lin]);
   }
   printf("\n");
   return 0;
}
```

Armazenando Nomes - Matriz String.

(Arquivo-fonte: 11 - Matriz/armazenando_nomes_string.c)



Resultado da execução do código anterior