

Matrizes bidimensionaisMatrizes de stringsMatrizes multidimensionaisInicializaçãoInicialização sem especificação de tamanhoMatrizes bidimensionais

Já vimos como declarar matrizes unidimensionais (vectors). Vamos tratar agora de matrizes bidimensionais. A forma geral da declaração de uma matriz bidimensional é muito parecida com a declaração de um vector:

***tipo\_da\_variável nome\_da\_variável [linha][coluna];***

É muito importante ressaltar que, nas matrizes, o índice da esquerda indexa as **linhas** e o da direita indexa as **colunas**. Quando vamos preencher ou ler uma matriz no C o índice da direita varia mais rapidamente que o índice da esquerda. Mais uma vez é bom lembrar que, na linguagem C, os índices variam de zero ao valor declarado, menos um; mas o C não vai verificar isto por nós, logo, manter os índices na faixa permitida é tarefa do programador.

1- Abaixo um exemplo do uso de uma matriz:

```
#include<stdio.h>
#include<conio.h>
main()
{
int t, i, num[3][4];
clrscr();
for(t=0; t<3; t++)
{
for(i=0;i<4;i++)
{
num[t][i]=(t*4)+i+1;
printf("num[%d][%d]= %d\n",t,i,num[t][i]);
}
}
scanf("%d");
}
```

No exemplo acima, a matriz **mtrx** é preenchida, sequencialmente por linhas, com os números de 1 a 12.

Tendo em conta a fórmula  $(t*4)+i+1$  a nossa matriz ficaria algo do género:

		<b>i</b>	<b>i</b>	<b>i</b>	<b>i</b>
		<b>0</b>	<b>1</b>	<b>2</b>	<b>3</b>
<b>t</b>	<b>0</b>	$(0*4)+0+1$ <b>1</b>	$(0*4)+1+1$ <b>2</b>	$(0*4)+2+1$ <b>3</b>	$(0*4)+3+1$ <b>4</b>
<b>t</b>	<b>1</b>	$(1*4)+0+1$ <b>5</b>	$(1*4)+1+1$ <b>6</b>	$(1*4)+2+1$ <b>7</b>	$(1*4)+3+1$ <b>8</b>
<b>t</b>	<b>2</b>	$(2*4)+0+1$ <b>9</b>	$(2*4)+1+1$ <b>10</b>	$(2*4)+2+1$ <b>11</b>	$(2*4)+3+1$ <b>12</b>

2- Este algoritmo faz o mesmo que o anterior mas com um processo diferente.

```
#include <stdio.h>
#include<conio.h>
main ()
{
int mtrx [3][4];
int i,j,count;
count=1;
clrscr();
for (i=0;i<3;i++)
    {
        for (j=0;j<4;j++)
            {
                mtrx[i][j]=count;
                count++;
                printf("mtrx[%d][%d]=%d\n ",i,j,mtrx[i][j]);
            }
    }
scanf("%d");
}
```

No exemplo acima, a matriz **mtrx** é preenchida, sequencialmente por linhas, com os números de 1 a 12.

3 **Exercício** - Utilizando uma matriz, elabore um algoritmo que tenha como output apenas os números pares de 2 a 24.

```
#include <stdio.h>
#include<conio.h>
main ()
{
int mtrx [3][4];
int i,j,count;
count=2;
clrscr();
for (i=0;i<3;i++)
    {
        for (j=0;j<4;j++)
            {
                mtrx[i][j]=count;
                count=count+2;
                printf("mtrx[%d][%d]=%d\n ",i,j,mtrx[i][j]);
            }
    }
scanf("%d");
}
```

No exemplo acima, a matriz **mtrx** é preenchida, sequencialmente por linhas, com os números de 1 a 12.

## Matrizes de strings

Matrizes de strings são matrizes bidimensionais. Imagine uma string. Que é um vector. Se fizermos um vector de strings estaremos fazer uma lista de vectores. Esta estrutura é uma matriz bidimensional de chars. Podemos ver a forma geral de uma matriz de strings como sendo:

```
char nome_da_variável [num_de_strings][compr_das_strings];
```

Aí surge a pergunta: como aceder a uma string individual? É só usar apenas o primeiro índice. Então, para aceder a uma determinada string fazemos:

```
nome_da_variável [índice]
```

4 - Aqui está um exemplo de um programa que lê 5 strings e as exibe no ecrã:

```
#include <stdio.h>
main ()
{
char strings [5][50];
int i;
for (i=0;i<5;i++)
    {
        printf ("\n\nDigite a string da linha %d ",i);
        gets (strings[i]);
    }
printf ("\n\nAs strings que digitou foram:\n\n");
for (i=0;i<5;i++)
    {
        printf ("%s\n",strings[i]);
    }
    scanf("%d");
}
```

O programa acima não é mais que uma lista de vectores formando assim uma matriz com 5 linhas e 50 colunas.

5 - Este exemplo tem exactamente a mesma função do exemplo anterior mas utiliza códigos diferentes além de que está mais completo.

```
/* Leitura e escrita de 10 nomes */
#include <stdio.h>
main()
{
    int I;
    char NOME[10][40];
    printf("\n\nListagem de nomes\n\n");

    /* Entrada dos dados */

    for (I = 0; I <= 9; I++)
        {
            printf("Digite o %2do. nome: ", I+1);
            fflush(stdin); fgets(NOME[I], 40, stdin);
        }

    /* Apresentacao dos nomes */

    for (I = 0; I <= 9; I++)
        printf("Nome: %2d --> %s", I+1, NOME[I]);
    scanf("%d");
}
```

A função `fflush()` caracteriza-se por ser um comando de limpeza do buffer (stream), em que `stdin` (standard input) é o stream a ser limpo, ou seja, o stream de entrada padrão, no caso representado pelo teclado.

Quando trabalhamos com números não necessitamos do `fflush`, mas quando trabalhamos com strings é conveniente utilizar o `fflush` o que não quer dizer que não se possa utilizar o `fflush` com dados numéricos.

**`fgets(NOME[I], 40, stdin);`** este código serve para ler os dados introduzidos pelo utilizador.

### Matrizes multidimensionais

O uso de matrizes multidimensionais na linguagem C é simples. A sua forma geral é:

***tipo\_da\_variável nome\_da\_variável [tam1][tam2] ... [tamN];***

Uma matriz N-dimensional funciona basicamente como outros tipos de matrizes. Basta lembrar que o índice que varia mais rapidamente é o índice mais à direita.

## Inicialização

Podemos inicializar matrizes, assim como podemos [inicializar variáveis](#). A forma geral de uma matriz como inicialização é:

```
tipo_da_variável nome_da_variável [tam1][tam2] ... [tamN]
= {lista_de_valores};
```

A lista de valores é composta por valores (do mesmo tipo da variável) separados por vírgula. Os valores devem ser dados na ordem em que serão colocados na matriz. Abaixo vemos alguns exemplos de inicializações de matrizes:

```
float vect [6] = { 1.3, 4.5, 2.7, 4.1, 0.0, 100.1 }
int matrnx [3][4] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
char str [10] = { 'J', 'o', 'a', 'o', '\0' };
char str [10] = "Joao";
char str_vect [3][10] = { "Joao", "Maria", "Jose" };
```

O primeiro demonstra inicialização de vetores. O segundo exemplo demonstra a inicialização de matrizes multidimensionais, onde **matrnx** está sendo inicializada com 1, 2, 3 e 4 em sua primeira linha, 5, 6, 7 e 8 na segunda linha e 9, 10, 11 e 12 na última linha. No terceiro exemplo vemos como inicializar uma string e, no quarto exemplo, um modo mais compacto de inicializar uma string. O quinto exemplo combina as duas técnicas para inicializar um vector de strings. Repare que devemos incluir o ; no final da inicialização.

## Inicialização sem especificação de tamanho

Podemos, em alguns casos, inicializar matrizes das quais não sabemos o tamanho *a priori*. O compilador C vai, neste caso verificar o tamanho do que você declarou e considerar como sendo o tamanho da matriz. Isto ocorre na hora da compilação e não poderá mais ser mudado durante o programa, sendo muito útil, por exemplo, quando vamos inicializar uma string e não queremos contar quantos caracteres serão necessários. Alguns exemplos:

```
char mess [] = "Linguagem C: flexibilidade e poder.";
```

```
int matrnx [][][2] = { 1,2,2,4,3,6,4,8,5,10 };
```

No primeiro exemplo, a string `mess` terá tamanho 36. Repare que o artifício para realizar a inicialização sem especificação de tamanho é não especificar o tamanho! No segundo exemplo o valor não especificado será 5.