Professor
Edson Pedro FERLIN

Programação em C

Matrizes e Strings

Prof. Edson Pedro Ferlin

Professor
Edson Pedro FERLIN

Programação em C

Definições

Definição : uma matriz é uma coleção de variáveis do *mesmo tipo* que são referenciadas por um nome em comum.

Observações :

1. Em C, todas as matrizes consistem em *localizações contíguas de memória*.
2. O *menor endereço* corresponde ao *primeiro elemento* e o *maior ao último elemento*.
3. As matrizes podem ter de *uma* a *N dimensões*.
4. Cada elemento de uma matriz é identificado pelo seu índice, e zero é o índice do primeiro elemento.

Matrizes de uma dimensão

Forma geral:

tipo var_nome[tamanho];
onde *tipo* declara o *tipo base da matriz* e *tamanho* define *quantos elementos a matriz armazenará*.

```
int mat[100];                 /* o índice varia de 0 à 99 */
float A[200] , B[50] , C[6];
char s[80];                   /* o índice varia de 0 à 79 */
```

N_total_Bytes = Tamanho_tipo_base * número_elementos

O compilador C não faz verificação de limites em matrizes; nada impede que seja ultrapassado o fim de uma matriz. Se isso ocorrer durante uma operação de atribuição, estaremos neste caso atribuindo algo a outra variável ou até mesmo para uma parte do código do programa

Matrizes de uma dimensão

Exemplo:

```
main()
{
   int x[10];
   int i;
   for ( i=0 ; i<10 ; i++ )  x[i] = i;
}
```

Matrizes de uma dimensão

Strings (1)

- Consiste em uma matriz de caracteres terminada em zero;
- Um zero é especificado como '\0';
- Declarar as matrizes de caracteres como sendo um caractere maior que a maior string que elas contenham;
- Exemplo: para uma matriz que conterá uma string de 10 caracteres, teremos: char str[11].

Matrizes de uma dimensão

Strings (2)

```
char str[8];
main()
{
   int i;
   for ( i=0 ; i<7 ; i++ )  str [i] = 'A' + i ;
}
```

Matrizes Unidimensionais São Listas

str[0]	str[1]	str[2]	str[3]	str[4]	str[5]	str[6]
A	B	C	D	E	F	G

Funções que usam Strings
gets() – insere string via teclado

```
#include "stdio.h"
main()
{
   char str [80];
   printf("informe uma string : ");
   gets(str);   /* Lê a string do teclado */
   printf("%s", str);
}
```

A função gets( ) continuará lendo caracteres até que seja pressionado o ENTER.

Funções que usam Strings
strcpy() - copia conteúdo

```
#include "stdio.h"
#include "string.h“
main()
{
   char str[80];
   strcpy(str , "Meu primeiro exemplo");
   printf("%s", str);
}
```

A função *strcpy(para, de);* é usada para *copiar* o conteúdo da string *de* para a string *para.*

Funções que usam Strings
strcat() – anexa strings

```
#include "stdio.h"
#include "string.h"
main()
{  char str_1[20] , str_2[15];
   strcpy(str_1 , "Meu ");
   strcpy(str_1 , "primeiro exemplo");
   strcat(str_1 , str_2);
   printf("%s", str_1);
   printf("%s", str_2);
}
```

A função *strcat(str_1 , str_2);* *anexa* str_1 em str_2; str_2 não é modificada.

Funções que usam Strings
strcmp() – compara duas strings

```
#include "stdio.h"
#include "string.h"
senha()
{  char s[80] ;
   printf("Qual a senha ? ");  gets(s);
   if( strcmp(s, "xxyyzz") )
   { printf("Senha inválida\n");
     return 0; }
return 1;    }
```

A função *strcmp(str_1 , str_2);* *compara* duas strings str_1 e str_2 e retorna zero se elas forem iguais. Se str_1 é lexicograficamente maior que str_2, então um número positivo é retornado; se for menor que str_2, um número negativo é retornado.

Funções que usam Strings
strlen() – comprimento da string

```
#include "stdio.h"
#include "string.h"
main()
{  char str[80];
   printf("Digite uma string : ");
   gets(str);
   printf(" A string possui %d caracteres ", strlen(str) );
}
```

A função *strlen(str);* retorna o *tamanho* de str. Todos os caracteres são contados (brancos, ...) exceto o terminador nulo.

Funções que usam Strings
toupper() – converte p/ maiúscula

```
/* Converter uma string para letras maiusculas*/
#include "stdio.h"
#include "string.h"
#include "ctype.h"
main()
{   char str[80];
    int i;
    strcpy(str,“converte para maiuscula");
    for ( i=0 ; str[i] ; i++ )          str[i] = toupper(str[i]);
    printf("%s", str);
}
```

O teste de str[i] no for é realizado porque um valor verdadeiro é qualquer valor diferente de zero. Portanto, o laço é executado até que encontre o zero (terminador nulo).

Matrizes
Bidimensionais

- Uma matriz bidimensional é uma lista de matrizes unidimensionais.

  tipo nome_matriz[*num_linhas*][*num_colunas*];

  int x[10][10], y[6][8];

  char str[100][80];

- As matrizes bidimensionais são armazenadas em uma matriz de linha e coluna, onde o *primeiro índice indica a linha e o segundo, a coluna.* Os índices iniciam com zero.

- No caso de *matrizes strings* o acesso a cada string pode ser feito por exemplo utilizando somente o índice das linhas, isto é: gets(mat_str[2][0]); é equivalente a gets(mat_str[2]);

Matrizes
Multidimensionais

- A forma geral é :

  ***tipo* nome[tam_1][tam_2]...[tam_n];**

- O uso de matrizes de três ou mais dimensões implica diretamente na quantidade de memória requerida para armazená-las, pois são alocadas permanentemente durante a execução do programa.

```
int mat[10][50][10];      /* alocara 10000 Bytes para mat */
double A[10][10][10];     /* alocara 8000 Bytes para A */
```

Matrizes
Inicialização (1)

```
int mat[10] = { 1, 3, 6, 7, 11, 12, 34, 566, 7, -60 };

float X[5] = { 12.98 , -89.9, -90 , 87 , 655.99 } ;

char str [7] = "MATRIZ";
```

equivale a `char str[6] = { 'M', 'A', 'T' , 'R', 'I', 'Z','\0'};`

```
int a[3][4] = { 3 , 8 , 9 , 1 , 3 , 5 , 77 , 6 , 5 , 55 , 2 , 6 };
```

ou

```
int a[3][4] = { 3 , 8 , 9 ,
                1 , 3 , 5 ,
               77 , 6 , 5 ,
               55 , 2 , 6
               };
```

ou

```
int a[ ][4] = { 3 , 8 , 9 ,
                1 , 3 , 5 ,
               77 , 6 , 5 ,
               55 , 2 , 6
               };
```


Matrizes
Inicialização (2)

```
char s1[20] = "informação inválida\n";
```

equivale a : `char s1[ ] = "informação inválida\n";`

Quando não especificado o tamanho da matriz, como em :

s1[ ] e a[ ][4]

neste caso, o dimensionamento da matriz será automático.

Se, em uma *declaração de Inicialização de matriz, o tamanho não é especificado*, o compilador criará uma matriz grande o suficiente para armazenar todos os inicializadores presentes.

A *vantagem dessa declaração* sobre a versão com especificação de tamanho é que a tabela pode ser aumentada ou diminuída sem mudar as dimensões da matriz.

Contato

eferlin@live.com

(BLOG) professorferlin.blogspot.com

(SITE) professorferlin.webnode.com.br

(YOUTUBE) ProfEdsonPedroFerlin