Многомерные статические массивы.

Многомерные статические массивы

Количество размерностей массива практически не ограничено.

```
int a[3][2];
```

Компилятор Си располагает строки матрицы a в памяти одну за другой вплотную друг к другу.

a[0][0]	a[0][1]	a[1][0]	a[1][1]	a[2][0]	a[2][1]

```
printf("%d", a[2][0]); // не a[2,0]
```

Компоненты многомерного массива

```
int a[2][3][5];
```

a – массив из двух элементов типа “int [3][5]”

```
int (*p)[3][5] = a;
```

a[i] – массив из трех элементов типа “int [5]” (i ∈ [0, 1])

```
int (*q)[5] = a[i];
```

a[i][j] – массив из пяти элементов типа “int” (i ∈ [0, 1], j ∈ [0, 1, 2])

```
int *r = a[i][j];
```

a[i][j][k] – элемент типа “int” (i ∈ [0, 1], j ∈ [0, 1, 2], k ∈ [0, 1, 2, 3, 4])

```
int s = a[i][j][k];
```

Инициализация многомерных массивов

```
int a[3][3] =
{
    {1, 2, 3},
    {4, 5}
};

int d[][2] = { {1, 2} };

int e[][] =
{
    {1, 2},
    {4, 5}
};
// error: array type has incomplete element type

int b[3][3] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
// warning: missing braces around initializer [-Wmissing-braces]

// c99
int c[2][2] = {[0][0] = 1, [1][1] = 1};
```

Указатели и многомерные массивы

Иногда удобно многомерный массив рассматривать как одномерный.

```
#define N 2
#define M 5
...
int a[N][M];
int *p;
...
for (p = &a[0][0]; p <= &a[N-1][M-1]; p++)
    *p = 0;
```

Указатели и многомерные массивы

Обработка строки матрицы (обнуление i-ой строки)

```
// указатель на начало i-ой строки
int *p = &a[i][0];

&a[i][0] => &(*(a[i] + 0)) => &(*a[i]) => a[i]
```

Т.е. выражение a[i] – это адрес начала i-ой строки.

```
// обнуление i-ой строки
for (p = a[i]; p < a[i] + M; p++)
    *p = 0;
```

a[i] можно передать любой функции, обрабатывающей одномерный массив:

```
zero(a[i], M);
```

Указатели и многомерные массивы

Обработка столбца матрицы (обнуление j-го столбца)

```
// указатель на строку (строка – это массив из M элементов)
int (*q)[M];
```

Скобки важны из-за приоритета операций! Без скобок получится массив из M указателей.

Выражение q++ смещает указатель на следующую строку

Выражение (*q)[j] возвращает значение в j-ом столбце строки, на которую указывает q.

```
for (q = a; q < a + N; q++)
    (*q)[j] = 0;
```

Передача многомерных массивов в функцию

Пусть определена матрица

```
int a[N][M];
```

Для ее обработки могут быть использованы функции со следующими прототипами:

```
void f(int a[N][M], int n, int m);
void f(int a[][M], int n, int m);
void f(int (*a)[M], int n, int m);
```

Неверные прототипы:

```
void f(int a[][], int n, int m);
// не указана вторая размерность => компилятор не сможет
// выполнить обращение по индексу
void f(int *a[M], int n, int m);
void f(int **a, int n, int m);
// неверный тип – массив указателей
```

Особенности использования const

```
void print(const int arr[][M], int n, int m)
{
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
            printf("%d ", arr[i][j]);
        printf("\n");
    }

    printf("\n");
}

...

int a[N][M] = {{1, 2, 3}, {4, 5, 6}};

// expected 'const int (*)[5]' but argument is of type 'int(*)[5]'
print(a, 2, 3);
```

Особенности использования const

Формальное объяснение

Согласно C99 6.7.3 #8 и 6.3.2.3.2 выражение T (*p)[N] не преобразуется неявно в T const (*p)[N].

Способы борьбы

- не использовать const;
- использовать явное преобразование типа

```
print((const int (*)[M]) a, 2, 3);
```

Особенности использования const

Почему такое неявное преобразование запретили

```
const char c = 'x';                    /* 1 */
char *p1;                  /* 2 */
// warning: initialization from incompatible pointer type
const char **p2 = &p1;                 /* 3 */
*p2 = &c;                  /* 4 */
*p1 = 'X';                 /* 5 */
```

3: В p2 поместили адрес p1.

4: С помощью p2 изменили p1. p1 теперь указывает на c.

5: С помощью p1 изменили константу.