Массивы в языке Си. Многомерные массивы. Массивы и указатели. Массивы и функции. Примеры

Массивы в языке Си
Массив – это составной тип данных в основе которого лежит набор однотипных скалярных(имеющих одно значение) или составных типов данных.
Другое определение:
Массив — это последовательность элементов одного типа, расположенных в памяти *подряд*.

Объявление одномерного массива:

```
int arr[10]; // массив из 10 элементов типа int
```

- Индексация начинается с нуля: `arr[0]`, `arr[1]`, `arr[2]`, ..., `arr[9]`.
- Инициализация:

  ```
  int arr[5] = {1, 2, 3, 4, 5}; // Задаем массиву готовые значения
  int arr[] = {1, 2, 3}; // размер массива определяется автоматически
  ```

  Основные операции:
- Чтение:

  ```
  int a = arr[1];
  ```

- Запись:

  ```
  arr[3] = 52;
  ```

Многомерные массивы
Многомерные массивы — массивы *массивов*, по сути - матрица(*Понравилось Шмегуле*).

Объявление и инициализация многомерного массива:

```
int matrix[3][4]; // массив 3x4
int matrix[2][2] = {{1, 2}, {3, 4}}; // массив 2x2 с заданными значениями
```

- Обращение к элементу: `matrix[1][0]` (строка 1, столбец 0). Чтение и запись аналогично одномерному массиву.
- В памяти хранится как один блок данных: строки идут подряд. То есть: `stroka1|stroka2|stroka3`
  Пример:

```
int matrix[2][3] = {{1, 2, 3}, {4, 5, 6}};
printf("%d", matrix[1][2]); // Вывод: 6
```

Массивы и указатели
Массивы жесть как связаны с указателями, прям вторая половинка <3.
Имя массива - адрес его первого элемента.
Пример:

```
int arr[5] = {10, 20, 30, 40, 50};
int *ptr = arr; // ptr указывает на arr[0]
printf("%d", *(ptr + 2)); // Вывод: 30
```

- `arr[i]` то же самое, что и `*(arr + i)`
- Адрес элемента: `&arr[i]`.
  Многомерные массивы и указатели
  Многомерные массивы также можно обрабатывать с использованием указателей. Как уже было сказано, в памяти многомерный массив хранится как последовательность элементов (в порядке "строки за строками").

Связь указателей и многомерных массивов:

- Имя двумерного массива (`matrix`) - это указатель на первую строку (массив под индексом `[0][0]`).
- `matrix[i]` - указатель на строку `i`.
- Элемент массива можно получить как `*(*(matrix + i) + j)`.
  Разбор этой шняги:

1. `*(matrix + i)` - получаем позицию в памяти первого элемента определенной строки, путем прибавления индекса строки к позиции первого элемента первой строки. *Вот как то так. (если не поняли, внизу файла сноска с кратким объяснением от гпт)*
2. `*( *(matrix + i) + j)` - к строке прибавляем индекс нужного элемента и получаем значение.
   Пример:

```c
int matrix[2][3] = {{1, 2, 3}, {4, 5, 6}};

int *row = matrix[0]; // Указывает на первую строку
printf("%d\n", *(row + 1)); // Вывод: 2 (второй элемент первой строки)

int value = *(*(matrix + 1) + 2); // Элемент из строки 1, столбца 2
printf("%d\n", value); // Вывод: 6
```

Прочее:

- Обращение к строкам - `matrix[i]` то же самое что и `*(matrix + i)`.
- Обращение к элементам - `matrix[i][j]` эквивалентен `*(*(matrix + i) + j)`.

Примечание от Мастера Си-си: Ребятки, ку! Постараюсь помочь вам с полным пониманием указателей на массив и что за страшная конструкция в виде `*(*(matrix + i) + j)`. Начнем с того, что указатель на массив - это указатель на его первый элемент. То есть буквально так:

```c
{
  int arr[3] = {34, 12, 2};
  int *p = arr; // *p = &arr[0]. Сейчас в 'p' хранится значение '34'
}
```

Теперь разберемся, что мы получаем, когда заходим в цикл и проходимся с помощью указателей по всему массиву. У нас есть цикл, который проходится по каждому элементу массива, где итератор - это i. Теперь мы получаем, что конструкция примет следующий вид при выводе элемента в поток вывода `*(matrix + i)`. Здесь мы вспоминаем, что указатель на массив - это указатель на первый элемент. Так давайте переведем это в более наглядный вид и будем считать, что идет вторая итерация цикла, то есть $i = 1$: `*(matrix[0] + 1)`. Здесь мы отлично видим, что к первому элементу добавляется сдвиг

на один байт, то есть теперь мы имеем $matrix[1] = 12$. **Отсюда мы можем сделать вывод, что конструкция** `*(matrix + i)` **это ведь наше привычное** `matrix[i]`.

Во второй части хотел бы разобрать конкретно конструкцию `*(*(matrix + i) + j)`, когда у нас двумерный массив и два вложенных цикла. А отсюда мы какой вывод можем сделать девочки?))) Конечно, это то же самое, что и `matrix[i][j]`. Пример для лучшего понимания:

```
#include <stdio.h>

int main(void) {
    int arr[2][2] = {
        {23, 15},
        {0, 1}
    };

    int *p = arr[0]; // *p = &arr[0][0]

    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2; j++) {
            printf("arr[%d][%d]. Значение: %d\n", i, j, *(*(arr + i) + j));
        }
    }
}
```

Вывод:

```
arr[0][0]. Значение: 23
arr[0][1]. Значение: 15
arr[1][0]. Значение: 0
arr[1][1]. Значение: 1
```

Небольшое пояснение: почему я указываю именно на $arr[0]$, а не на arr. Если я указываю на arr, то я указываю на `&arr[0]`, а у нас двумерный массив, поэтому мы должны указывать на первый элемент с указанием номера строки и столбца. Отсюда мы понимаем, что если наш указатель уже указывает на `&arr[0]` (строка), то на что нам еще нужно указать? Ну, конечно, на столбец. Поэтому запись принимает вид $*p = matrix[0]$. Если это для вас кажется сложным или вам просто тяжеловатенько запомнить это, то вы можете использовать другую запись: `int *p = &arr[0][0]` <=> `int *p = matrix[0]`

Массивы и функции

Массивы передаются в функции **по адресу**, а не по значению.

Пример на передачу массива в функцию:

```
void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
}
int main() {
    int arr[3] = {1, 2, 3};
    printArray(arr, 3);
}
```

- `arr` - указатель на первый элемент массива.

  Многомерные массивы в функции

  При передаче многомерного массива нужно указать размеры всех измерений, кроме первого. также необходимо явно указать тип.

  Пример:

```
void printMatrix(int matrix[][3], int rows) {
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < 3; j++) {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }
}
```

Также можно передавать массив так:

```
void printMatrix(int *matrix, int rows, int cols) {
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            printf("%d ", *(matrix + i * cols + j));
        }
        printf("\n");
    }
}
```

В таком случае, элемент массива получать нужно так:

```
matrix[i][j] = *(matrix + i * cols + j)
```

Примеры

Нахождение суммы элементов массива (явно не использовав указатель, т.к имя массива - указатель на его первый элемент):

```
int sumArray(int arr[], int size) {
    int sum = 0;
    for (int i = 0; i < size; i++) {
        sum += arr[i];
    }
    return sum;
}
```

Вывод матрицы:

```
void processMatrix(int (*matrix)[3], int rows) {
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < 3; j++) {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }
}
```

Транспонирование матрицы через указатели:

```
void transposeMatrix(int (*matrix)[3], int (*transposed)[2], int rows, int
cols) {
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            transposed[j][i] = matrix[i][j];
        }
    }
}
```

Напоминаю, `transposed[j][i] = matrix[i][j]` будет равно `*(*(transposed + j) + i) = *(*(matrix + i) + j)`

База

- Имя массива - указатель на его первый элемент.
- Передача массивов в функции осуществляется **по указателю**.
- Индексы массива `i`, `j` не должны быть равны или больше его размерности. если массив `arr[10][5]`, то максимальные `i` и `j` это 9 и 4.

Типо сноска

- `matrix + i` перемещает указатель к строке `i`.
- `*(matrix + i)` разыменовывает этот указатель, чтобы получить массив (строку) `i`.
- `(*(matrix + i) + j)` смещает указатель внутри строки `i` на элемент `j`.
- `*(*(matrix + i) + j)` возвращает значение этого элемента.