

## Методические указания

### Тематическое занятие 15

## **Многомерные массивы. Матрицы.**

### Содержание

<b>Многомерный массив</b> .....	2
<i>Двухмерный массив</i> .....	2
<i>Инициализация элементов</i> .....	2
<i>Описание многомерного массива</i> .....	2
<i>Использование в качестве параметров функций</i> .....	3
<i>Указатели и многомерные массивы</i> .....	3
<b>Работа с матрицами</b> .....	5
<i>Описание матрицы</i> .....	5
<i>Обход элементов матрицы</i> .....	6
<i>Поиск элемента в матрице</i> .....	6
<i>Определение характеристик матрицы</i> .....	7
<b>Операции с матрицами</b> .....	7
<i>Результат операций</i> .....	7
<i>Умножение матрицы на вектор</i> .....	7
<b>Многомерный динамический массив</b> .....	8
<i>Общие обозначения для создания матрицы</i> .....	8
<i>Непрерывный блок памяти для всех элементов матрицы</i> .....	8
<i>Отдельные блоки памяти для каждой строки матрицы</i> .....	9
<i>Непрерывный блок памяти для всех строк матрицы</i> .....	9
<b>Упражнения</b> .....	10
<i>Упражнение 15.1</i> .....	10
<i>Упражнение 15.2</i> .....	10
<i>Упражнение 15.3</i> .....	10
<i>Упражнение 15.4</i> .....	10

# Многомерный массив

## Двухмерный массив

В языке C двухмерный массив описывается как **массив одномерных массивов**, например:

```
int m[10][20];
```

– это объявление двумерного массива целых чисел (типа `int`), размерностью **10×20**. Тогда первая часть этого объявления `int m[10][20];` означает, что `m` – массив из 10-и элементов, а вторая `int m[10][20];` описывает массив из 20-и элементов типа `int`.

Обращение к элементам данного массива:

```
m[0][0]=100; /* первый элемент первой строки */  
m[9][19]=200; /* последний элемент последней строки */
```

## Инициализация элементов

Пример инициализации элементов массива:

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

1	2	3
4	5	6

Тоже самое получится, если:

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

При частичной инициализации недостающие элементы массива будут заполнены нулями:

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

1	2	0
3	4	0

но при этом:

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

1	2	3
4	0	0

## Описание многомерного массива

Массив, размерность (ранг, количество индексов) которого больше единицы ( $N > 1$ ) называется ***N*-мерным массивом**.

Многомерные массивы описываются аналогично двумерным. Общая форма объявления многомерного массива:

```
типЭлем имяМассива[разм1][разм2]...[размN];
```

Здесь `типЭлем` – тип элементов массива (например, один из стандартных типов), а `размN` – количество элементов в  $N$ -м измерении.

Пример объявления четырехмерного массива из  $4 \times 3 \times 6 \times 5 = 360$  вещественных чисел:

```
double a[4][3][6][5];
```

Обращение к элементам данного массива:

```
a[1][0][5][3]=2.718;
```

```
a[3][2][5][4]=3.14; /* последний элемент */
```

## Использование в качестве параметров функций

При использовании имени массива в качестве параметра функции необходимо указать **количество элементов в каждом измерении, кроме самого левого** (первого).

Например, для объявленного выше четырехмерного массива `a` функция, в которую передается этот массив, должна выглядеть так:

```
void func(int a[][3][6][5])
{
    ...
}
```

Конечно, можно включить в объявление и размер первого измерения, но это излишне.

## Указатели и многомерные массивы

Пусть объявлен двумерный массив небольшого размера:

```
int arr[3][2] = {{10, 20},
                 {30, 40},
                 {50, 60}};
```

10	20
30	40
50	60

При таком объявлении `arr` – это имя двумерного массива, т.е. синоним указателя, который содержит адрес первого из его трех элементов `&arr[0]`. Разыменование данного указателя `*arr` дает этот адрес:

```
*arr == *(&arr[0]) == *&arr[0] == arr[0].
```

Первый элемент объявленного двумерного массива `arr[0]` сам является одномерным массивом из двух элементов, а значит, содержит адрес своего первого элемента `&arr[0][0]`. Разыменование `*(arr[0])` позволяет получить значение элемента одномерного массива, хранящееся в `arr[0][0]`, то есть число 10 типа `int`.

Поэтому для получения значения (типа `int`) первого элемента первого одномерного массива к имени двумерного массива `arr` **операцию разыменования необходимо применить дважды**:

```
**arr == *(*arr) == *(arr[0]) == *(&arr[0][0]) == *&arr[0][0] == arr[0][0].
```

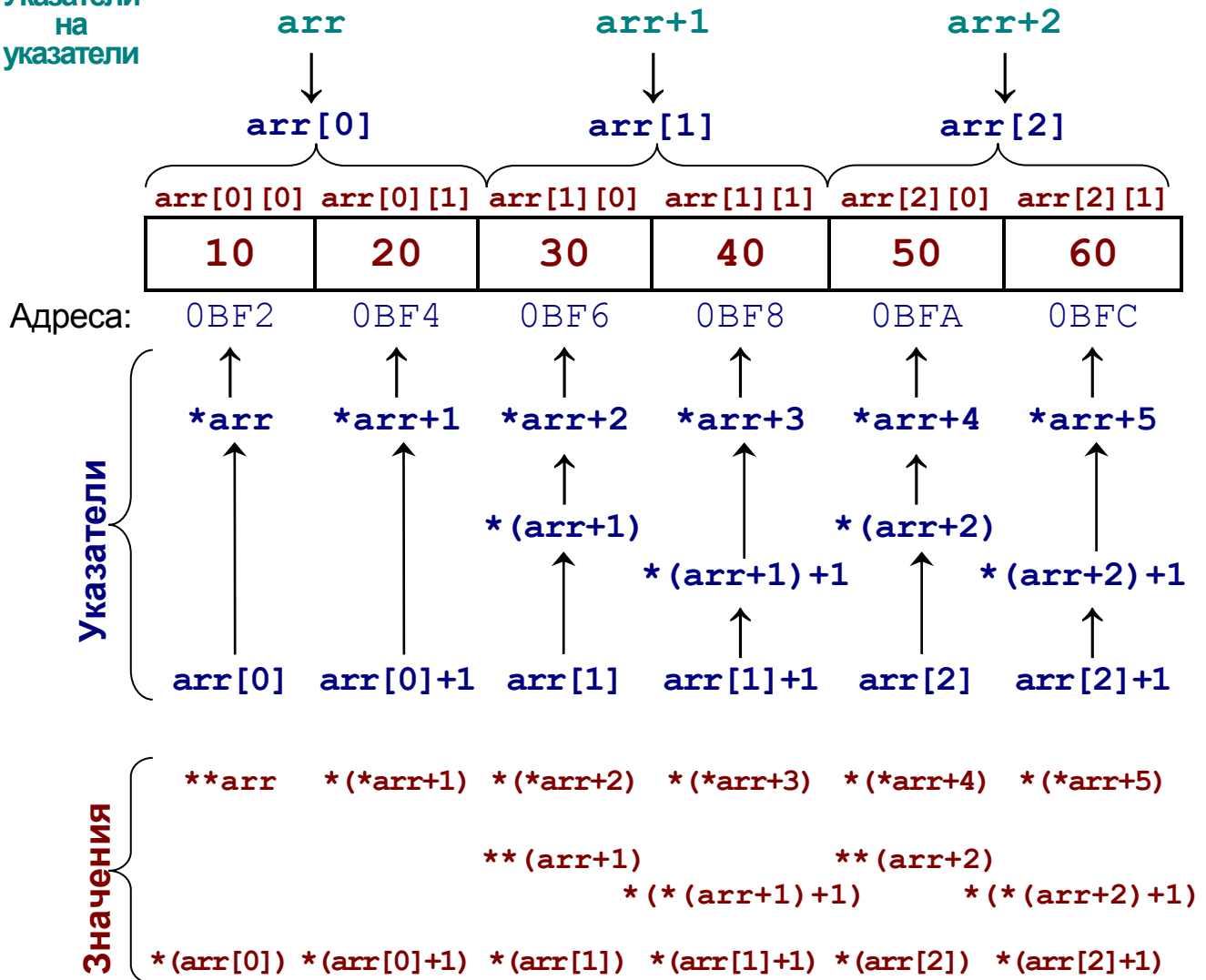
Применение адресной арифметики к указателям позволяет перемещаться по массиву

Выражение	Значение выражения		
	в двумерном массиве	в одномерном массиве	числовое
<b>arr</b>	адрес 1-го элемента <code>&amp;arr[0]</code>	адрес 1-го элемента <code>&amp;arr[0][0]</code> 1-го массива <code>arr[0]</code>	0BF2
<b>arr+2</b>	адрес 3-го элемента <code>&amp;arr[2]</code>	адрес 1-го элемента <code>&amp;arr[2][0]</code> 3-го массива <code>arr[2]</code>	0BFA
<b>*arr</b>	значение 1-го элемента <code>arr[0]</code>	адрес 1-го элемента <code>&amp;arr[0][0]</code> 1-го массива <code>arr[0]</code>	0BF2
<b>*(arr+2)</b>	значение 3-го элемента <code>arr[2]</code>	адрес 1-го элемента <code>&amp;arr[2][0]</code> 3-го массива <code>arr[2]</code>	0BFA
<b>*arr+1</b>		адрес 2-го элемента <code>&amp;arr[0][1]</code> 1-го массива <code>arr[0]</code>	0BF4
<b>*(arr+2)+1</b>		адрес 2-го элемента <code>&amp;arr[2][1]</code> 3-го массива <code>arr[2]</code>	0BFC
<b>**arr</b>		значение 1-го элемента <code>arr[0][0]</code> 1-го массива <code>arr[0]</code>	10
<b>*(*(arr+2))</b>		значение 1-го элемента <code>arr[2][0]</code> 3-го массива <code>arr[0]</code>	50
<b>*(*arr+1)</b>		значение 2-го элемента <code>arr[0][1]</code> 1-го массива <code>arr[0]</code>	20
<b>*(*(arr+2)+1)</b>		значение 2-го элемента <code>arr[2][1]</code> 3-го массива <code>arr[2]</code>	60

Общая формула двойного разыменования для массива с объявлением `arr[N][M]` :

**`*(*(arr+n)+m) == arr[n][m]`**, где  $0 \leq n < N$  и  $0 \leq m < M$ .

Указатели  
на  
указатели



## Работа с матрицами

### Описание матрицы

При обращении к многомерным массивам компьютер много времени затрачивает на вычисление адреса, так как при этом приходится учитывать значение каждого индекса. Поэтому **доступ к элементам многомерного массива происходит значительно медленнее**, чем к элементам одномерного.

Поэтому на практике массивы размерности более двух ( $N > 2$ ) используются довольно редко. Логическая структура двухмерного массива может быть представлена прямоугольной **матрицей**.

Описание и инициализация матрицы **3×5** (строк × столбцов):

```
#include <stdio.h>
#define n 3
#define m 5
```

```

int main(void) {
    int matrix[n][m] = { /* объявление и инициализация */
        { 1, 2, 3, 4, 5},
        { 6, 7, 8, 9,10},
        {11,12,13,14,15}
    };
    printf("matrix[%d][%d]=%d\n",2,3,matrix[2][3]);
    return 0;
}

```

В результате на экран будет выведено: matrix[2][3]=14

## Обход элементов матрицы

В предыдущем примере матрица заполнена при инициализации, теперь рассмотрим примеры заполнения матрицы при обходе ее элементов.

Обход элементов массива в привычном порядке (по строкам сверху вниз и слева на право)

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

реализован в следующем фрагменте программы, заполняющей матрицу значениями таблицы умножения:

```

int i; /* счетчик по строкам от 1 до n */
int j; /* счетчик по столбцам от 1 до m */
for (i=0; i<n; i++)
    for (j=0; j<m; j++)
        matrix[i][j]=(i+1)*(j+1);

```

## Поиск элемента в матрице

Составим функцию для поиска максимального элемента матрицы и его индексов (координат).

```

void SearchMax(int a[][m],
               int *amax, /* значение искомого элемента */
               int *imax, int *jmax) { /* его координаты */
    int i; /* счетчик по строкам от 1 до n */
    int j; /* счетчик по столбцам от 1 до m */
    *amax=a[0][0];
    *imax=0; *jmax=0;
    for (i=0; i<n; i++)
        for (j=0; j<m; j++)
            if (a[i][j] > *amax) {
                *amax=a[i][j];
                *imax=i; *jmax=j;
            }
}

```

Вызов этой функции для матрицы `matrix`, описанной выше:

```
int am, im, jm;  
SearchMax(matrix, &am, &im, &jm);  
printf("MAX: matrix[%d][%d]=%d\n", im, jm, am);
```

Если в матрице содержится несколько элементов с максимальным значением, то данный алгоритм найдет первый из них.

## Определение характеристик матрицы

Составим функцию для проверки симметричности квадратной матрицы.

```
int Symmetric(int mtr[n][n]) {  
    int i; /* счетчик по строкам */  
    int j; /* счетчик по столбцам */  
    for (i=0; i<n-1; i++)  
        for (j=i+1; j<n; j++)  
            if (mtr[i][j]!=mtr[j][i])  
                return 0;  
    return 1;  
}
```

## Операции с матрицами

### Результат операций

В общем случае матричные операции **некоммутативны**. То есть результат операции зависит от порядка следования операндов. Например, при умножении вектора-столбца на вектор-строку получается квадратная матрица:

The diagram illustrates the multiplication of a column vector (3x1) and a row vector (1x3). The column vector is represented by three stacked squares, the row vector by three squares in a single row, and the resulting square matrix by a 3x3 grid of squares.

а при умножении вектора-строки на вектор-столбец – скалярная величина:

The diagram illustrates the multiplication of a row vector (1x3) and a column vector (3x1). The row vector is represented by three squares in a single row, the column vector by three stacked squares, and the resulting scalar value by a single square.

### Умножение матрицы на вектор

Рассмотрим пример процедуры для вычисления произведения матрицы размерности 3x5 и вектора-столбца (5x1). Результатом такой операции будет вектор-столбец размерности 3x1.

```
void Mult(int mtr[n][m], /* матрица 3x5 */  
          int vm[m], /* вектор-множитель 5x1 */  
          int vn[n]) { /* вектор-результат 3x1 */  
    int i; /* счетчик по строкам от 1 до n */  
    int j; /* счетчик по столбцам от 1 до m */
```

```

for (i=0; i<n; i++)
    vn[i]=0; /* обнуление вектора-результата */
for (i=0; i<n; i++)
    for (j=0; j<m; j++)
        vn[i] = vn[i] + mtr[i][j] * vm[j];
}

```

Данная функция «возвращает» одномерный массив `vn[n]` (вектор-результат), точнее она изменяет значения его элементов. Но использование операции разыменования (\*) здесь не требуется, поскольку имена массивов сами являются указателями на их первые элементы.

По этой же причине при вызове данной функции в нее передается имя возвращаемого массива без операции получения адреса &:

```

int vector[m]={1,2,3,4,5}; /* вектор-множитель */
int result[n]; /* вектор-результат */
Mult(matrix,vector,result);

```

## Многомерный динамический массив

### Общие обозначения для создания матрицы

Рассмотрим способы создания многомерного динамического массива на примере двумерного массива (матрицы) размером  $n \times m$  ( $n$  строк,  $m$  столбцов). Зададим символические константы:

```

#define n 10 /* n - количество строк */
#define m 15 /* m - количество столбцов */

```

Объявим переменные `i` и `j` для обозначения индексов текущей строки и текущего столбца матрицы соответственно:

```

int i, j; /* i - номер строки, j - номер столбца */

```

Целочисленный индекс `i` изменяется от 0 до  $(n-1)$ , а индекс `j` – от 0 до  $(m-1)$ .

Создать массив в динамической памяти можно несколькими способами.

### Непрерывный блок памяти для всех элементов матрицы

Самый простой способ – это выделение непрерывного блока динамической памяти для всех  $n \times m$  элементов матрицы. Объявление динамического массива:

```

int *a;
a = (int*)malloc(n*m*sizeof(int));

```

Фактически здесь объявлен одномерный массив из  $n \times m$  элементов типа `int`.

Поэтому и обращаться к значениям элементов такой матрицы придётся как к элементам одномерного массива:

`a[i*m+j]`      или      `*(a+i*m+j)`

Недостаток такого способа – невозможно получить доступ к значениям элементов через двойную индексацию `a[i][j]`, поскольку двойное разыменование – некорректная операция.



## Отдельные блоки памяти для каждой строки матрицы

Другой способ – это выделение отдельных (непрерывных) блоков динамической памяти для каждой строки матрицы (или для каждого столбца матрицы). Указатели на каждый такой блок необходимо хранить в отдельном массиве указателей, для которого так же выделяется динамическая память:

```
int **a;    /* Указатель на массив из n указателей на строки. */
a = (int**)malloc(n*sizeof(int*)); /* Выделение памяти для */
for (i=0; i<n; ++i) /* массива указателей. */
    a[i] = (int*)malloc(m*sizeof(int)); /* Выделение памяти */
                                         /* для строк матрицы. */
```

Здесь `a` – одномерный динамический массив указателей на строки матрицы, который содержит `n` элементов типа `(int*)`. Элементы этого массива `a[0] ... a[n-1]` являются указателями на строки матрицы, каждая из которых сама является одномерным динамическим массивом из `m` элементов типа `int`.

Тогда обращаться к значениям элементов матрицы можно через двойную индексацию:

`a[i][j]`      или      `*(*(a+i)+j)`

Где первое разыменование `a[i]` возвращает указатель на `i`-ю строку матрицы, а второе `a[i][j]` дает значение элемента `j`-го столбца этой строки матрицы.

При данном способе создания динамического многомерного массива:

- 1) матрица занимает больше памяти за счет хранения отдельного массива указателей на строки матрицы, общий объем занимаемой памяти складывается из  $n \times m$  ячеек с элементами матрицы типа `int` и `n` ячеек с указателями типа `(int*)`;
- 2) строки матрицы хранятся в разных фрагментах памяти, поэтому невозможно работать со всеми элементами матрицы через непрерывное пространство адресов;
- 3) при освобождении памяти необходимо вызвать функцию `free()` сначала для каждой строки матрицы `a[i]`, и только затем для массива указателей `a`;
- 4) время работы программы увеличивается за счет многократного выполнения функций `malloc()` и `free()`.

В зависимости от способа организации управления памятью на конкретном компьютере, выполнение функций `malloc()` и `free()` может занимать довольно значительное время.

## Непрерывный блок памяти для всех строк матрицы

Объединить два предыдущих подхода позволяет третий способ создания двухмерного массива – выделение непрерывного блока памяти для хранения всех элементов и всех указателей на строки матрицы.

```
int **a;    /* Указатель на массив из n указателей на строки. */
a = (int**)malloc(n*sizeof(int*) + n*m*sizeof(int)); /* Общее
выделение памяти для всех указателей и строк матрицы. */
for (i=0; i<n; ++i)
    a[i] = (int*)(a+n) + i*m;
/* Запись адресов строк в указатели на строки матрицы. */
```

Так же, как и в предыдущем случае, обращаться к значениям элементов матрицы можно через двойную индексацию:

`a[i][j]`      или      `*(*(a+i)+j)`

А чтобы освободить всю выделенную динамическую память, достаточно вызвать `free(a)`.

Рассмотрим пример использования такого непрерывного блока.

Пусть на некоторой платформе указатель занимает 8 байт памяти, а целочисленное значение типа `int` – 4 байта. Для матрицы  $3 \times 2$  ( $n=3$  и  $m=2$ ) выделим в динамической памяти блок размером  $n \cdot 8 + n \cdot m \cdot 4 = 3 \cdot 8 + 3 \cdot 2 \cdot 4 = 48$  байт. И заполним элементы массива значениями в соответствии с формулой:

`a[i][j] = 10*(i+1)+(j+1);`

Такому блоку может соответствовать следующая схема ячеек памяти с их конкретными адресами:

	<b>a[0]</b>	<b>a[1]</b>	<b>a[2]</b>	<b>a[0][0]</b>	<b>a[0][1]</b>	<b>a[1][0]</b>	<b>a[1][1]</b>	<b>a[2][0]</b>	<b>a[2][1]</b>
<b>a:</b>	<b>D8FC</b>	<b>D904</b>	<b>D90C</b>	<b>11</b>	<b>12</b>	<b>21</b>	<b>22</b>	<b>31</b>	<b>32</b>
Адреса:	D8E4	D8EC	D8F4	D8FC	D900	D904	D908	D90C	D910

Для хранения матрицы выделена непрерывная область памяти, которая логически разделяется на две части: в первой части хранятся указатели на строки матрицы (на первый элемент каждой строки), во второй части хранятся сами элементы матрицы (по строкам). Объем первой части –  $n$  ячеек типа `(int*)`, второй части –  $n \times m$  ячеек типа `int`.

## Упражнения

### Упражнение 15.1

Составить программу, которая создает матрицу  $4 \times 4$  и заполняет ее случайными целыми числами. Вывести созданную матрицу на экран (построчно, в виде прямоугольника).

### Упражнение 15.2

Составить функцию, которая вычисляет сумму элементов  $k$ -го столбца матрицы. Номер столбца (число  $k$ ) передается в функцию в качестве параметра.

### Упражнение 15.3

Составить функцию, которая находит минимальный элемент матрицы и выводит его координаты на экран. Если таких элементов несколько, найти координаты каждого из них.

### Упражнение 15.4

Выполнить **упражнение 15.1** для матрицы, созданной в динамической области памяти.