НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ЯДЕРНЫЙ УНИВЕРСИТЕТ «МИФИ»
Кафедра информатики и процессов управления (№17)

Дисциплина «Информатика (основы программирования)»

Методические указания

Тематическое занятие 11

Массивы указателей и многомерные массивы.

Содержание

Создание и использование массива указателей 2
Массив индексов 2
Создание массива указателей 2
Доступ к элементам массива указателей через индексы 3
Доступ к значениям элементов исходных массивов 4

Адресная арифметика для массива указателей 5
Доступ к элементам массива указателей 5
Доступ к адресам элементов исходных массивов 5
Доступ к значениям элементов исходных массивов 5
Применения адресной арифметики для массива указателей 6
Общая схема доступа к элементам исходных массивов 7

Многомерный массив 8
Двухмерный массив 8
Инициализация элементов 8
Описание многомерного массива 8
Использование в качестве параметров функций 9
Указатели и многомерные массивы 9

Работа с матрицами 11
Описание матрицы 11
Обход элементов матрицы 12
Поиск элемента в матрице 12
Определение характеристик матрицы 13

Операции с матрицами 13
Результат операций 13
Умножение матрицы на вектор 13

Многомерный динамический массив 14
Общие обозначения для создания матрицы 14
Непрерывный блок памяти для всех элементов матрицы 14
Отдельные блоки памяти для каждой строки матрицы 15
Непрерывный блок памяти для всех строк матрицы 15

Упражнения 16
Упражнение 11.1 ..... *16*
Упражнение 11.2 ..... *16*
Упражнение 11.3 ..... *16*
Упражнение 11.4 ..... *16*
Упражнение 11.5 ..... *17*
Упражнение 11.6 ..... *17*

Создание и использование массива указателей

Массив индексов

Пусть имеется неупорядоченный массив целых чисел:

```
int a[5]={39, 76, 26, 14, 52};
```

a[0]	a[1]	a[2]	a[3]	a[4]
39	76	26	14	52

и поставлена задача отсортировать по возрастанию числа, находящиеся в массиве `a`, но при этом запрещено переставлять элементы массива `a` и копировать их.

Кажется, что самым простым решением является создание нового массива, в котором будут храниться индексы элементов исходного массива в порядке возрастания значений этих элементов:

```
int ind[5]={3, 2, 0, 4, 1};
```

Используя массив индексов `ind` можно вывести на экран исходный массив `a`, отсортированный по возрастанию:

```
for (i=0; i<5; ++i)
  printf("%d\n",a[ind[i]]);
```

ind[0]	ind[1]	ind[2]	ind[3]	ind[4]
3	2	0	4	1

a[3]	a[2]	a[0]	a[4]	a[1]
14	26	39	52	76

Создание массива указателей

Рассмотрим пример, когда имеются несколько массивов с различным количеством элементов, все элементы этих массивов нужно расположить в общей последовательности, упорядоченной по возрастанию:

```
int a[5]={39, 76, 26, 14, 52};
int b[3]={67, 15, 48};
int c[4]={82, 23, 59, 61};
```

В этом случае при использовании массива индексов необходимо различать, какому из исходных массивов принадлежит данный индекс. Это усложняет задачу сортировки.

В подобных ситуациях часто используют иной подход – создают общий ***массив указателей***, в котором хранятся адреса всех элементов исходных массивов.

В рассматриваемом примере определим количество элементов общего массива указателей: 5+3+4=12. Объявим массив с названием `arp` (от «**ar**ray of **p**ointers»), элементы которого являются указателями на переменные типа `int`:

```
int *arp[12];
```

Заполним этот массив адресами элементов массивов `a`, `b` и `c` в порядке возрастания их значений:

arp[0]	arp[1]	arp[2]	arp[3]	arp[4]	arp[5]	arp[6]	arp[7]	arp[8]	arp[9]	arp[10]	arp[11]
&a[3]	&b[1]	&c[1]	&a[2]	&a[0]	&b[2]	&a[4]	&c[2]	&c[3]	&b[0]	&a[1]	&c[0]
14	15	23	26	39	48	52	59	61	67	76	82

Теперь в массиве указателей адреса упорядочены по возрастанию значений тех элементов массивов, на которые они ссылаются. При этом не нарушен порядок исходных массивов, в них не произошло ни одной перестановки элементов.

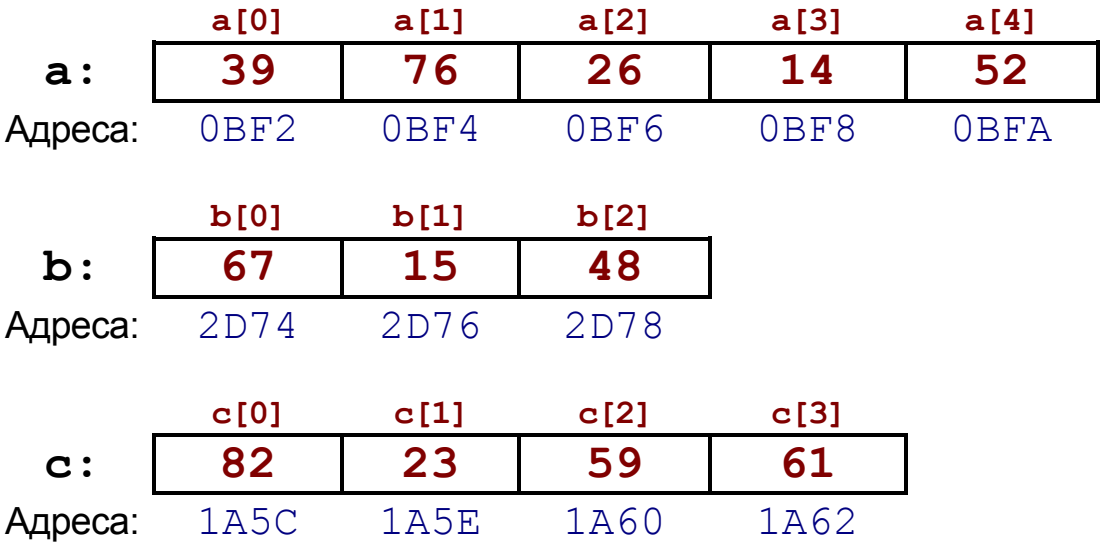
Доступ к элементам массива указателей через индексы

При заполнении массива указателей `arp` обращаться к его элементам можно несколькими способами. Самый простой из них – это доступ через индексы элементов массива `arp`:

```
arp[0] = &a[3];
arp[1] = &b[1];
arp[2] = &c[1];
arp[3] = &a[2];
...
```

Элементами массива `arp` являются указатели, которые принимают значения адресов ячеек элементов массивов `a`, `b` и `c`.

Проиллюстрируем рассматриваемый пример схемой ячеек памяти с их конкретными адресами:



	arp[0]	arp[1]	arp[2]	arp[3]	arp[4]	arp[5]
arp:	0BF8	2D76	1A5E	0BF6	0BF2	2D78
Адреса:	B8E4	B8EC	B8F4	B8FC	B904	B90C

	arp[6]	arp[7]	arp[8]	arp[9]	arp[10]	arp[11]
	0BFA	1A60	1A62	2D74	0BF4	1A5C
Адреса:	B914	B91C	B924	B92C	B934	B93C

Следует заметить, что в данном упрощенном примере для хранения значения переменной типа `int` используется 2 байта, а для хранения указателя (адреса памяти) – 8 байт. Однако реальная программа должна корректно выполняться в разных вычислительных системах, где количество байт для хранения значений переменных и адресов может быть различным.

Доступ к значениям элементов исходных массивов

Для получения доступа к значениям исходных массивов `a`, `b` и `c` (на которые ссылаются элементы массива указателей `arp`) достаточно применить операцию раскрытия ссылки (разыменования) к адресу, хранящемуся в ячейке массива указателей `arp`. При этом получаются следующие тождественные выражения, которые возвращают одинаковые значения:

```
*arp[0] == *(&a[3]) == *&a[3] == a[3] == 14
*arp[1] == *(&b[1]) == *&b[1] == b[1] == 15
*arp[2] == *(&c[1]) == *&c[1] == c[1] == 23
*arp[3] == *(&a[2]) == *&a[2] == a[2] == 26
...
```

Таким образом, через массив указателей `arp` можно изменять значения исходных массивов `a`, `b` и `c`, например:

```
*arp[5] = 44; /* значение b[2] изменится с 48 на 44 */
```

Адресная арифметика для массива указателей

Доступ к элементам массива указателей

Другим способом доступа к элементам массива указателей `arp` является использование операции разыменования и адресной арифметики для указателя `arp`:

```
*arp     = &a[3];
*(arp+1) = &b[1];
*(arp+2) = &c[1];
*(arp+3) = &a[2];
...
```

Здесь имя массива `arp` является синонимом указателя на первый элемент массива, который сам является указателем на четвертый элемент массива `a`. Таким образом, `arp` – синоним указателя на указатель и содержит адрес `&arp[0]` 1-го из 12-и элементов массива. Его разыменование `*arp` даст значение этого адреса (которое само является адресом 4-го элемента массива `a`):

```
*arp == *(&arp[0]) == *&arp[0] == arp[0] == &a[3] == 0BF8
```

Применение адресной арифметики к идентификатору `arp` позволяет перемещаться по массиву указателей `arp`. Например, выражение `arp+1` ссылается на 2-й элемент массива указателей `arp` :

```
arp+1 == &arp[1]
```

Его разыменование даст значение 2-го из 12-и элементов массива `arp`, которое является адресом 2-го элемента массива `b`)

```
*(arp+1) == *(&arp[1]) == *&arp[1] == arp[1] == &b[1] == 2D76
```

Доступ к адресам элементов исходных массивов

Поскольку элементами массива указателей `arp` являются адреса элементов исходных массивов (`a`, `b` или `c`), то к ним также можно применять адресную арифметику. При этом перемещение будет происходить по одному из исходных массивов `a`, `b` или `c`.

Например:

```
*arp+1 == (*arp)+1 == arp[0]+1 == &a[3]+1 == &a[4] == 0BFA
*(arp+1)+1 == (*(arp+1))+1 == arp[1]+1 == &b[1]+1 == &b[2] == 2D78
```

Доступ к значениям элементов исходных массивов

Разыменование этих указателей позволяет получить доступ к элементам исходных массивов `a`, `b` или `c`:

```
**arp == *(*arp) == *arp[0] == *&a[3] == a[3] == 14
*(*(arp+1)) == *(arp[1]) == *arp[1] == *&b[1] == b[1] == 15
*(*arp+1) == *(arp[0]+1) == *(&a[3]+1) == *&a[4] == a[4] == 52
*(*(arp+1)+1) == *(arp[1]+1) == *(&b[1]+1) == *&b[2] == b[2] == 48
```

Общая формула такого двойного разыменования для идентификатора **arp** массива указателей из **N** элементов:

***(*(arp+n)+i) == *(arp[n]+i)**,

где **0≤n<N**,

i – смещение индекса в одном из исходных массивов.

Следует обратить внимание, что в рассматриваемом примере изменение переменной **n** на единицу приводит к смещению в памяти на 8 байт, поскольку происходит переход к соседней ячейке с адресом. А изменение переменной **i** на единицу приводит к смещению на 2 байта, поскольку переход происходит на соседнюю ячейку, хранящую значение переменной типа `int`.

Конечно, величина смещения в байтах может быть различной в зависимости от конкретной вычислительной системы, на которой выполняется рассматриваемый программный код.

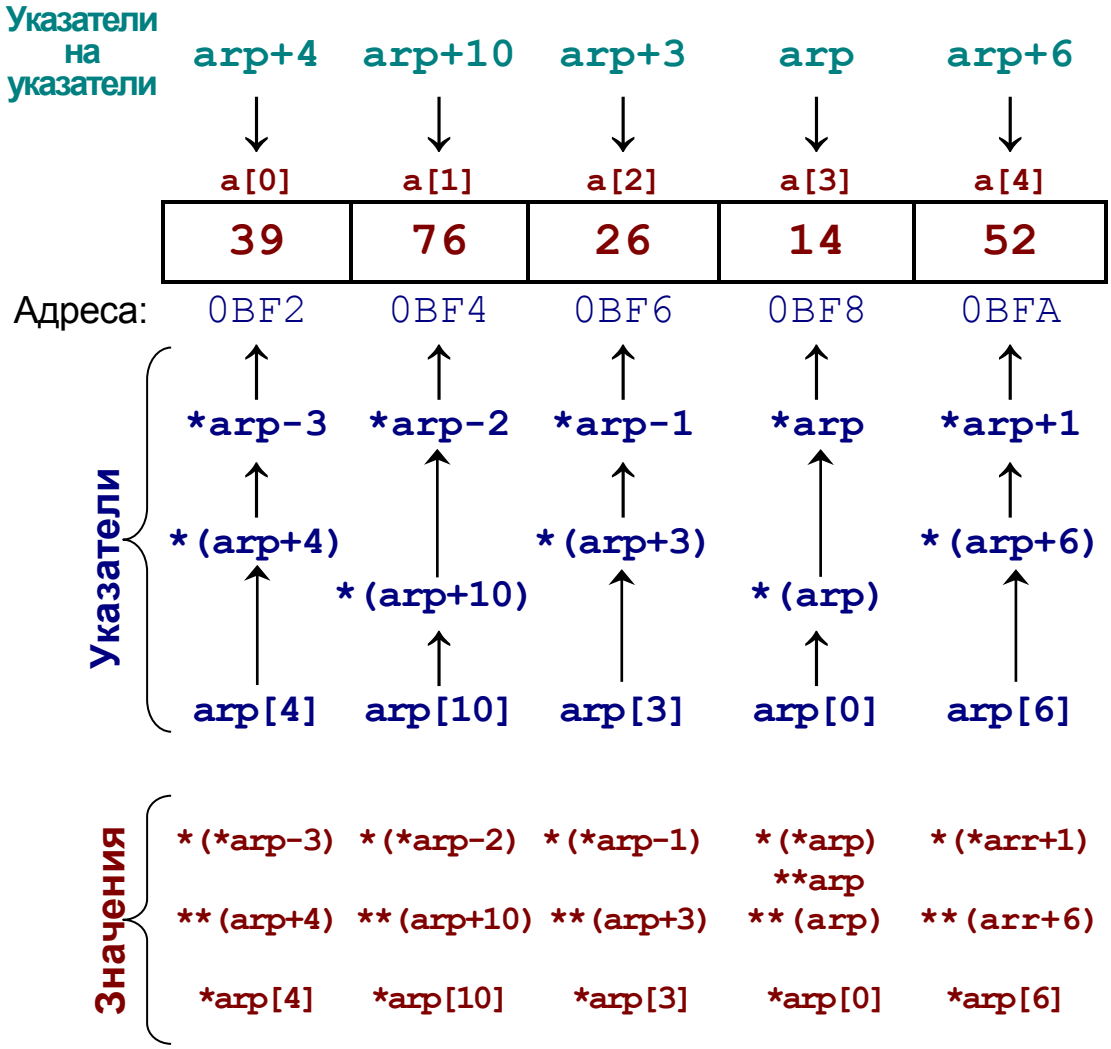
Применения адресной арифметики для массива указателей

Применение операций адресной арифметики к идентификатору массива указателей `arp` позволяет перемещаться как по самому массиву указателей, так и по исходным массивам `a`, `b` и `c`:

Выражение	Значение выражения		
	в массиве указателей	в исходных массивах `a`, `b` и `c`	число-вое
arp	адрес 1-го элемента `&arp[0]`		`B8E4`
arp+2	адрес 3-го элемента `&arp[2]`		`B8F4`
***arp**	значение 1-го элемента `arp[0]`	адрес 4-го элемента `&a[3]` массива `a`	`0BF8`
***(arp+2)**	значение 3-го элемента `arp[2]`	адрес 2-го элемента `&c[1]` массива `c`	`1A5E`
***arp+1**	значение 7-го элемента `arp[6]`	адрес 5-го элемента `&a[4]` массива `a`	`0BFA`
***(arp+2)+1**	значение 8-го элемента `arp[7]`	адрес 3-го элемента `&c[2]` массива `c`	`1A60`
****arp**		значение 4-го элемента `a[3]` массива `a`	**14**
***(*(arp+2))**		значение 2-го элемента `c[1]` массива `c`	**23**
***(*arp+1)**		значение 5-го элемента `a[4]` массива `a`	**52**
***(*(arp+2)+1)**		значение 3-го элемента `c[2]` массива `c`	**59**

Общая схема доступа к элементам исходных массивов

Для наглядности изобразим общую схему различных способов доступа к элементам исходного массива `a`:



На данной схеме в столбцах (для каждой ячейки памяти) одним цветом обозначены тождественные выражения (которые возвращают одинаковые значения). Например, для 1-го элемента массива `a`:

- тождественные выражения для указателей:

```
arp[4]== *(arp+4) == *arp-3 == &a[0] == 0BF2
```

- тождественные выражения для значений (типа `int`):

```
*arp[4] == **(arp+4) == *(*arp-3) == a[0] == 39
```

Подобные схемы можно изобразить для массивов `b` и `c`.

Многомерный массив

Двухмерный массив

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

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

– это объявление двухмерного 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 двухмерным. 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×**3**×**6**×**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

10	**20**
30	**40**
50	**60**

Пусть объявлен двухмерный массив небольшого размера:

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

При таком объявлении `arr` – это имя двухмерного 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 двухмерного 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**:

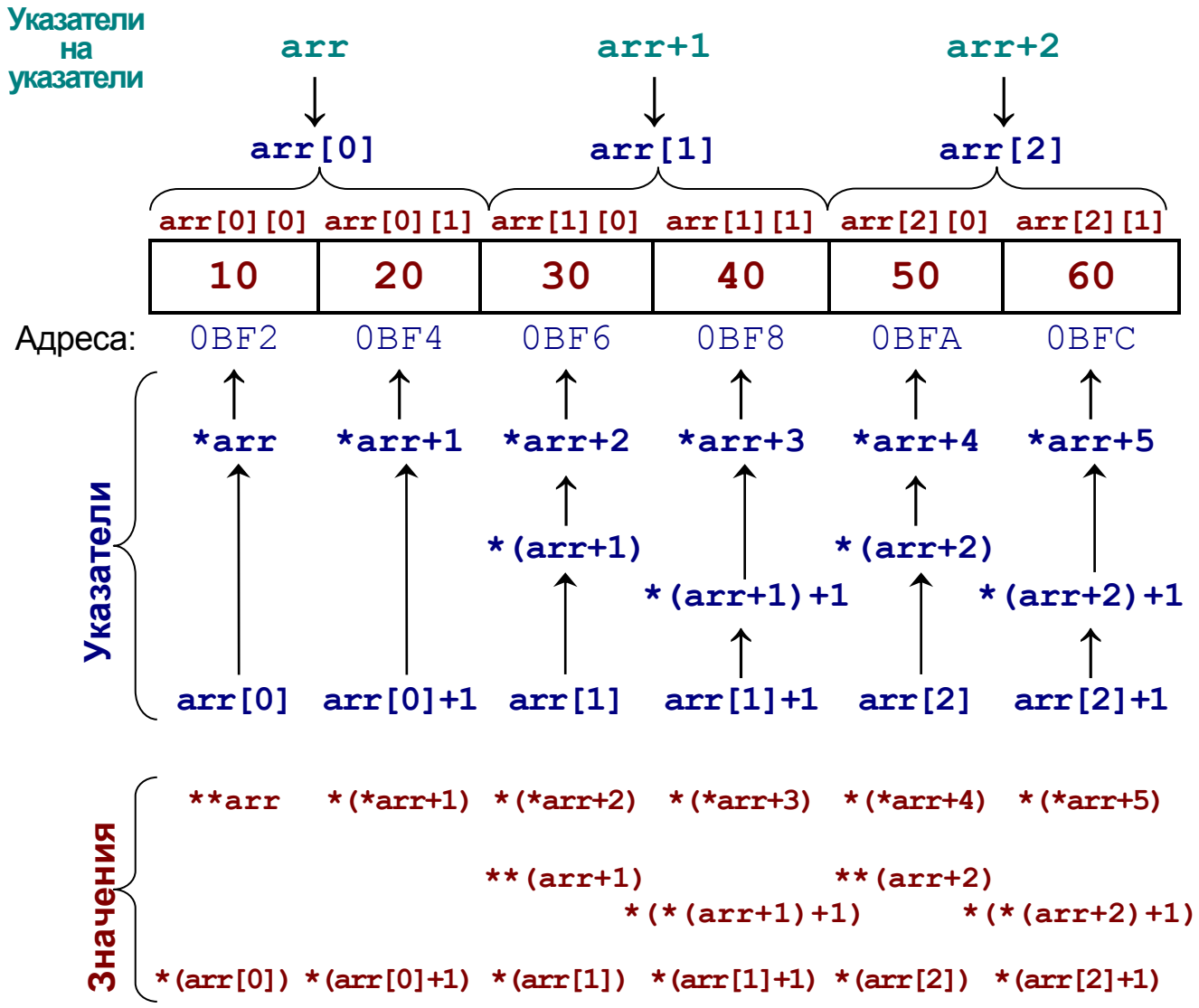
`**arr == *(*arr) == *(arr[0]) == *(&arr[0][0]) == *&arr[0][0] == arr[0][0]`.

Применение адресной арифметики к указателям позволяет перемещаться по массиву

Выражение	Значение выражения		
	в двухмерном массиве	в одномерном массиве	числовое
`arr`	адрес 1-го элемента `&arr[0]`	адрес 1-го элемента `&arr[0][0]` 1-го массива `arr[0]`	`0BF2`
`arr+2`	адрес 3-го элемента `&arr[2]`	адрес 1-го элемента `&arr[2][0]` 3-го массива `arr[2]`	`0BFA`
`*arr`	значение 1-го элемента `arr[0]`	адрес 1-го элемента `&arr[0][0]` 1-го массива `arr[0]`	`0BF2`
`*(arr+2)`	значение 3-го элемента `arr[2]`	адрес 1-го элемента `&arr[2][0]` 3-го массива `arr[2]`	`0BFA`
`*arr+1`		адрес 2-го элемента `&arr[0][1]` 1-го массива `arr[0]`	`0BF4`
`*(arr+2)+1`		адрес 2-го элемента `&arr[2][1]` 3-го массива `arr[2]`	`0BFC`
`**arr`		значение 1-го элемента `arr[0][0]` 1-го массива `arr[0]`	**10**
`*(*(arr+2))`		значение 1-го элемента `arr[2][0]` 3-го массива `arr[0]`	**50**
`*(*arr+1)`		значение 2-го элемента `arr[0][1]` 1-го массива `arr[0]`	**20**
`*(*(arr+2)+1)`		значение 2-го элемента `arr[2][1]` 3-го массива `arr[2]`	**60**

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

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



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
```

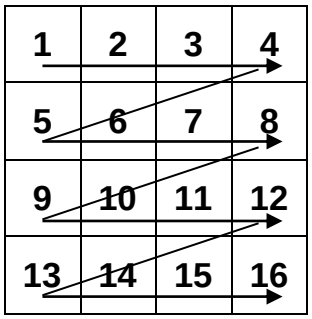
```
int main(void) {
  int matrix[n][m] = { /* объявление и инициализация */
    { 1, 2, 3, 4, 5},
    { 6, 7, 8, 9,10},
    {11,12,13,14,15}
  };
  printf("matrix[%d][%d]=%d\n",2,3,matrix[2][3]);
  return 0;
}
```

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)



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

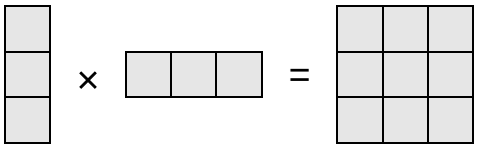
Составим функцию для проверки симметричности квадратной матрицы.

```
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;
}
```

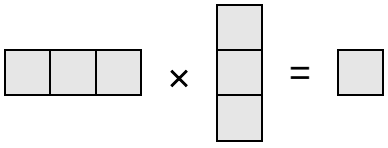
Операции с матрицами

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:



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



Умножение матрицы на вектор

Рассмотрим пример процедуры для вычисления произведения матрицы размерности 3×5 и вектора-столбца (5×1). Результатом такой операции будет вектор-столбец размерности 3×1.

```
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 двухмерного 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)`, я индекс **`j`** – от `0` до `(m-1)`.

Создать массив в динамический 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 их $n \times m$ 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 тритий 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)`

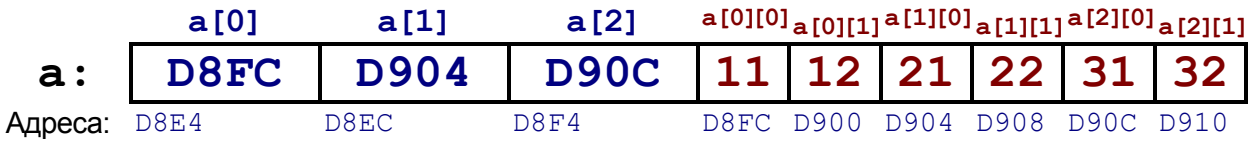
А чтобы освободить всю выделенную динамическую память, достаточно вызвать `free(a)`.

Рассмотрим пример использования такого непрерывного блока.

Пусть на некоторой платформе указатель занимает **8** байт памяти, а целочисленное значение типа `int` – **4** байта. Для матрицы 3×2 (**n**=3 и **m**=2) выделим в динамической памяти блок размером $n\cdot8+n\cdot m\cdot4=3\cdot8+3\cdot2\cdot4=48$ 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:



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 11.1

По аналогии с общей схемой различных способов доступа к элементам массива `a` составить описание различных способов доступа к элементам массива `b`.

Упражнение 11.2

По аналогии с общей схемой различных способов доступа к элементам массива `a` составить описание различных способов доступа к элементам массива `c`.

Упражнение 11.3

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

Упражнение 11.4

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

Упражнение 11.5

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

Упражнение 11.6

Выполнить ***упражнение 11.3*** для матрицы, созданной в динамической области памяти.