Указатели

Определение указателей

При объявлении переменных, компилятор выделяет для них память, размер которой определяется указанным типом и инициализирует их значениями (если они присутствуют). Далее все обращения к этим переменным заменяются на адрес участка памяти, в котором хранятся их значения.

Разработчик может сам определять переменные для хранения адресов памяти, т.е. *указатели*.

Указатель – это переменная, которая может содержать адрес некоторого объекта. Форма объявления указателя:

***Тип * Имя_Указателя*;**

Например: int *a; double *f; char *w;

Звездочка относится непосредственно к ***Имени_Указателя***, поэтому для объявления несколько указателей, ее нужно записывать перед каждым.

Например:

```
int  *a,  *b,  c;
```

определены два указателя на участки памяти для целочисленных данных и целочисленная переменная *c*.

Значение указателя равно первому байту участка памяти, на который он ссылается. Под указатель любого типа выделяется 4 байта.

В языке Си имеются три вида указателей

– указатели на объект известного типа;

– указатель типа ***void***;

– указатель на функцию.

Указатель может быть константой или переменной, а также указывать на константу или переменную.

С ***указателями-переменными*** связаны две унарные операции **&** и *.

Операция **&** означает «***взять адрес***» операнда. Операция * имеет смысл – «***значение, расположенное по указанному адресу***» (операция разадресации).

Обращение к объектам любого типа в языке Си может производиться:

– по имени (идентификатору);

– по указателю (*косвенная адресация*):

Имя_Указателя = &*Имя_Объекта*;

***Имя_Указателя** – косвенная адресация.

Операция разадресации используется как для получения значения величины, адрес которой хранится в указателе, так и для ее изменения (не константы).

Унарная операция **&** применима только к адресуемым выражениям (***L-значениям***), т.е. к переменным для которых выделена память и можно определить ее адрес.

Получить адрес скалярного выражения, самоопределенной константы или регистровой переменной (***register***) ***нельзя***.

Пример:

int x, *y; Переменная ***int*** и указатель на объект типа ***int***

y = &x; Адрес переменной ***x*** присвоим указателю ***y***

(установим указатель ***y*** на переменную ***x***)

*y = 1; По указанному адресу записать значение 1, т.е.

*y = *x* = 1

Выражение ***Имя_Указателя*** можно использовать в левой части оператора присваивания, т.к. оно является ***L-значением***, т.е. определяет адрес участка памяти.

Имя_Указателя считают именем переменной, на которую ссылается указатель. С ней допустимы все действия, определенные для величин соответствующего типа (если указатель инициализирован).

Инициализация указателей

Опасная ошибка – использование неинициализированных указателей, поэтому желательно присвоить указателю начальное значение уже при объявлении.

Инициализатор записывается после ***Имени Указателя*** либо после знака равенства, либо в круглых скобках. Рассмотрим способы инициализации указателей.

1. Присваивание указателю адреса известного объекта:

а) используя операцию **&** (получение адреса):

```
int  a = 5;
int *p = &a;    Указателю p присвоили адрес объекта a
int *p (&a);    То же самое другим способом
```

б) с помощью ранее определенного указателя (***p***):

```
int *g = p;
```

в) с помощью имени массива или функции, которые трактуются как адрес начала участка памяти, в котором размещается указанный объект.

Следует знать, что ***имена массивов*** и ***функций*** являются ***константными указателями***, которые можно присвоить переменной-указателю, но ***нельзя*** изменять, например:

```
int  x[10],  *y;
y = x;          Присваивание константы переменной
x = y;      Ошибка, т.к. в левой части константа.
```

2. Присваивание пустого значения:

```
int  *x1 = NULL;
int  *x2 = 0;
```

Константа ***NULL*** – указатель, равный нулю (можно использовать просто цифру **0**), т.е. отсутствие адреса.

Так как объекта с нулевым адресом не существует, то пустой указатель обычно используют для проверки, ссылается указатель на некоторый объект или нет.

3. Присваивание указателю адреса выделенного участка динамической памяти (стандартные функции *calloc*, *malloc* использовать не будем*)* с помощью операции C++ ***new*** :

```
int  *n = new  int;
```

Результат этой операции – адрес начала выделенной (захваченной) памяти, при возникновении ошибки – ***NULL***.

Операция sizeof (размер …)

Формат

sizeof (***Параметр***);

Параметр – тип или имя объекта (не имя ***функции***).

Операция определяет размер указанного *Параметра* в байтах (тип результата *int*).

Если указано имя сложного объекта (массив, структура), то результатом будет размер всего объекта. Например:

sizeof (int) Результат 4 байта

double b[5];

sizeof (b) Результат 8 байт * 5 = 40 байт

Операции над указателями

Помимо уже рассмотренных операций, с указателями можно выполнять арифметические операции сложения, инкремента (++), вычитания, декремента (--) и операции сравнения.

Арифметические операции с указателями автоматически учитывают ***размер типа*** величин, адресуемых указателями.

Например, инкремент увеличивает (перемещает в право) указатель типа ***int*** на 4 байта, а инкремент указателя типа ***double*** – на 8 байт, и т.п.

Эти операции применимы к указателям одного типа и имеют смысл в основном при работе с данными, последовательно размещенными в памяти, например, с массивами.

При работе с массивами, инкремент перемещает указатель к следующему элементу массива, декремент – к предыдущему.

Указатель, таким образом, может использоваться в выражениях вида

p* # *iv ; ## ***p*** ; ***p*** ## ; ***p* # = *iv*** ;

p – указатель (***pointer***), ***iv*** – целочисленное выражение (*int value*), **#** – символ операции '+' или '–'.

Результат таких выражений – увеличенное или уменьшенное значение указателя на величину

***iv* * *sizeof* (**p*)**

С указателями могут использоваться любые операции сравнения, но чаще используются отношения равенства или неравенства. Другие отношения имеют смысл только для указателей на последовательно размещенные объекты (элементы одного массива).

В применении к массивам разность двух указателей равна числу объектов в соответствующем диапазоне, т.е. разность указателей, например, на третий и шестой элементы равна 3.

Суммирование двух указателей не допускается.

Значение указателя в шестнадцатеричном виде можно вывести на экран с помощью функции *printf*, используя спецификацию %***p*** (*pointer*), или с помощью *cout*.

Связь указателей и массивов

Работа с массивами тесно связана с применением указателей. Рассмотрим эту связь на примере одномерного массива.

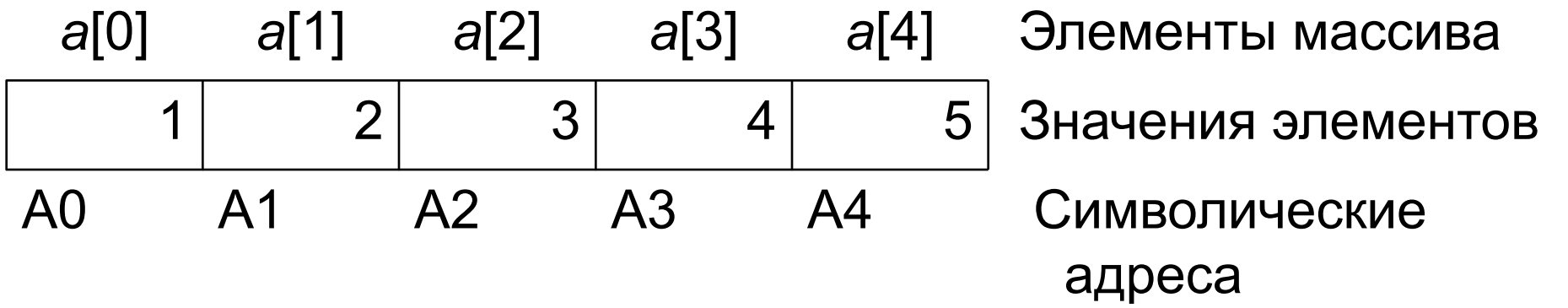
Пусть объявлены одномерный массив ***a*** и указатель ***p*** :

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

Имя массива ***a*** – константный указатель на его начало, т.е

a = &*a*[0] Адрес первого элемента

Элементы массива ***a*** в выделенной памяти располагаются следующим образом (по 4 байта каждый):



Указатель ***a*** – адрес начала массива (A0).

Тогда адрес первого элемента

A1 = A0 + sizeof(int) = A0 + 4;

адрес второго

A2 = A1 + sizeof (int) = A0 + 2 * 4 = A0 + 8 … и т.д.

Если установить указатель ***p*** на объект ***a*** :

p = *a*;

что эквивалентно ***p* = &*a*[0];** то получим, что и ***p* = A0**.

Идентификаторы ***a*** и ***p*** – указатели, очевидно что с учетом адресной арифметики обращение к ***i***-му элементу массива ***a*** может быть записано следующими выражениями

$$a[i] \sim *(a+i) \sim *(p+i) \sim p[i]$$

приводящими к одинаковому результату.

Очевидна и эквивалентность выражений:

– Адрес начала массива в памяти:

$\&a[0] \leftrightarrow \&(*a) \leftrightarrow a$

– Обращение к первому элементу массива:

$*a \leftrightarrow a[0]$

Указатели на указатели

Указатели, как и переменные любого другого типа, могут объединяться в массивы.

Объявление ***массива указателей p*** на целые числа:

```
int  *p[10], y;
```

Теперь каждому из элементов массива указателей ***p*** можно присвоить адрес переменной ***y***, например: *p*[1] = &*y*;

Чтобы найти значение переменной ***y*** через данный элемент массива ***p***, необходимо записать **p*[1].

В языке Си можно описать переменную типа «***указатель на указатель***». Это ячейка памяти (переменная), в которой будет храниться адрес указателя на некоторую переменную. Признак такого типа данных – повторение символа «*» перед именем переменной. Количество звездочек определяет уровень вложенности указателей друг в друга. При объявлении указателей на указатели возможна их инициализация.

Например:

```
int a = 5;
int *p = &a;
int **pp = &p;
int ***ppp = &pp;
```

Если присвоить переменной ***a*** новое значение: a=10; то следующие величины будут иметь такие же значения 10:

*p **pp ***ppp

Для доступа к памяти, отведенной под переменную *a* можно использовать и индексы, т.е. следующие выражения - эквивалентны :

*p ~ p[0] ;

**pp ~ pp[0][0] ;

***ppp ~ ppp[0][0][0] .

Фактически, используя указатели на указатели, мы имеем дело с многомерными массивами.

Многомерные массивы

Декларация многомерного массива:

Тип Имя_Масси*в*а [***Размер*1**][***Размер*2**]…[***РазмерN***] ;

Наиболее быстро изменяется последний индекс, т.к. многомерные массивы размещаются в памяти компьютера построчно друг за другом.

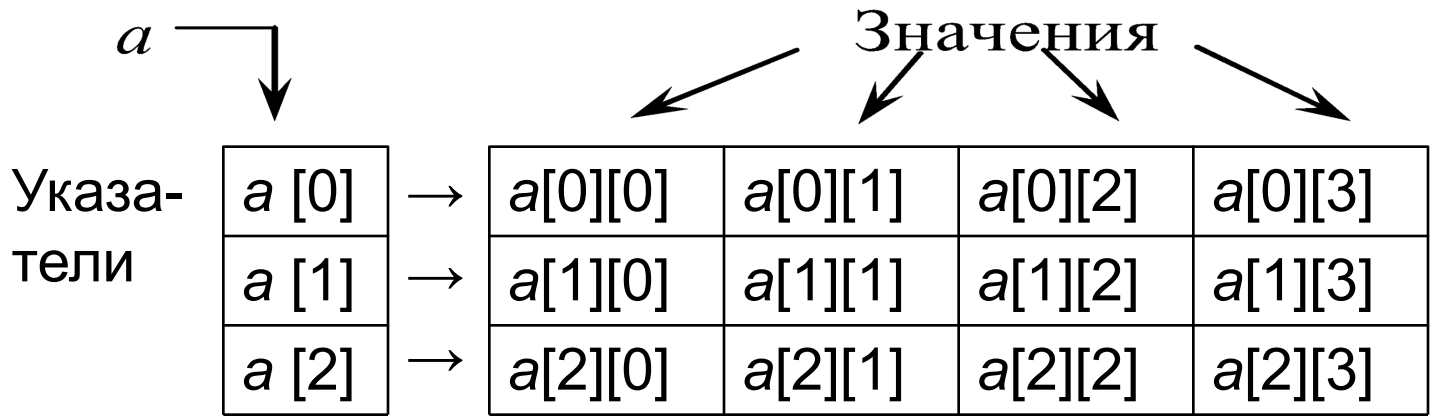
Рассмотрим особенности работы с многомерными массивами на примере двухмерного массива.

Пусть приведена декларация двухмерного массива:

***int a*[3][4];**

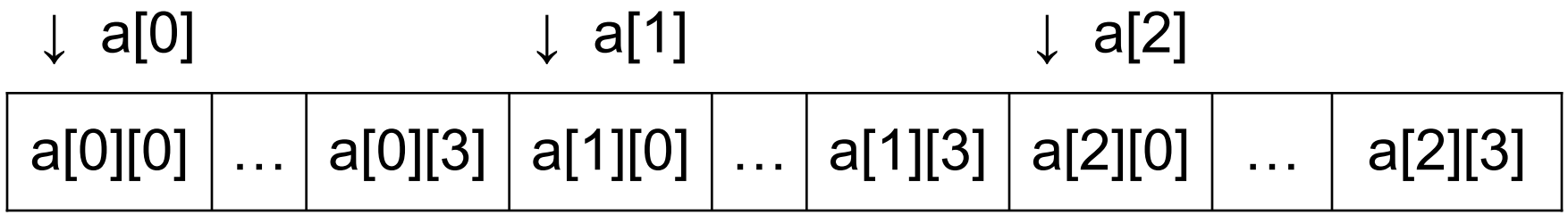
Двухмерный массив *a*[3][4] компилятор рассматривает как массив трех указателей, каждый из которых установлен на начало одномерного массива размером 4.

Схема размещения массива a в памяти:



Причем в данном случае, указатель *a*[1] имеет адрес равный *a*[0] + 4**sizeof* (*int*), т.е. каждый первый элемент следующей строки располагается за последним элементом предыдущей строки.

Т.е. массив ***a*** в памяти занимает последовательно размещенный участок:



Обращению к элементам массива при помощи операции индексации *a*[*i*][*j*] соответствует эквивалентное выражение, использующее адресную арифметику – *(*(*a* + *i*) + *j*).

Аналогичным образом можно установить соответствие между указателями и массивами с произвольным числом измерений.

Динамические массивы

Работа с динамическими массивами связана с операциями их создания и уничтожения по запросу программы, при котором выделение (захват) и освобождение памяти производится не на этапе обработки программы, как для статических массивов, а в процессе ее выполнения.

Для объявления динамических массивов используются указатели.

В языке Си захват и освобождение памяти выполняются при помощи библиотечных функций (*calloc*, *mallok*, *free*).

В языке C++ для захвата и освобождения памяти используется более простой механизм – операции *new* и *delete*.

Рассмотрим эти операции на простых примерах:

1) *type *p = **new** type* (*Значение*);

– захват участка памяти размером *sizeof*(*type*), путем установки на него указателя, и запись в эту область указанного *Значения*; например:

int *p = new int(5); соответствует int *p = new int;
*p = 5;

Тогда для освобождения захваченной памяти

***delete** p*;

2) *type *p = **new** type* [*n*];

– захват памяти на ***n*** последовательно размещенных объектов, возвращает указатель на начало участка ОП размером *n*sizeof*(*type*); используется для создания массива;

В этом случае освобождение всей захваченной памяти

delete [ ] *p*;

Результат операции ***new*** – адрес начала выделенного участка памяти для размещения данных, указанного типа и количества, при возникновении ошибки (например, при нехватке памяти) результат равен ***NULL***.

Причем операция ***delete*** (удалить) не уничтожает значения, находящиеся по указанным адресам, а разрешает компилятору использовать ранее занятую память. Но попытка использовать эти значения может привести к непредсказуемым результатам.

Квадратные скобки в операции *delete* при освобождении памяти, занятой массивом, обязательны. Их отсутствие также может привести к непредсказуемым результатам.

Создание одномерного динамического массива

Кусочек кода, необходимый для работы с динамическим одномерным массивом вещественных чисел ***x*** (размером ***n***) с проверкой достаточно ли памяти для его размещения:

```
…
double *x;                  - Объявление указателя для массива
int   i, n;
cout << " Size = : ";       - Определение размера на этапе
cin >> n;                     выполнения программы
x = new double [n] ;        - Создание массива
if (x == NULL) {            - Проверка на ошибку
     cout << " Error ! “ << endl;        (Ошибка)
     return;
}
…                           - Обработка массива
delete [ ]x;                - Освобождение занятой памяти
…
```

Создание двухмерного динамического массива

Создание двухмерного динамического массива выполняется в два этапа:

***Этап* 1**: выделяется память под указатели, расположенные последовательно друг за другом (по количеству строк);

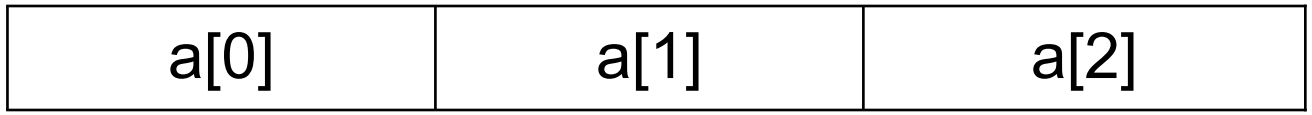
***Этап* 2**: каждому указателю выделяется участок памяти под элементы (по количеству столбцов).

```
. . .
int  **a, n, m, i, j;        - Объявление указателя на указатель
                               для двухмерного массива a
cout << " n, m : ";          - Определение размеров массива на
                               этапе выполнения программы
cin >> n >> m;                 для n  строк  и  m  столбцов
a = new int* [n];            - 1. Захват памяти для n  указателей
for (i=0; i<n; i++)          - 2. Захват памяти для m  элементов
    a[i] = new int [m];        каждой строки
```

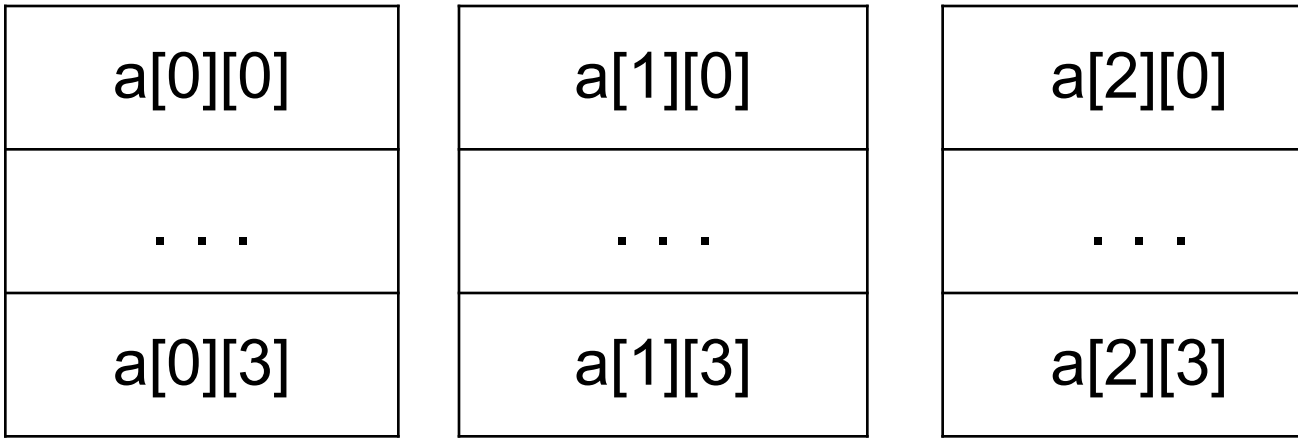
```
. . .                                  - Обработка массива
for ( i=0; i<n; i++)                   - Освобождение памяти:
      delete []a[i];                     сначала, занятую под элементы,
delete []a;                              затем, занятую под указатели
. . .
```

Схема выделения памяти под массив ***a*** для ***n*** = **3**, ***m*** = **4**:

1) выделяем память под 3 указателя на строки (по 4 б.), т.е. создаем одномерный массив из указателей:



2) каждый указатель строки устанавливаем на участок выделенной памяти под элементы



Рассмотрим некоторые необходимые участки программ для выполнения лабораторной работы № 6 (реализация ввода-вывода для консольных приложений).

Имеем следующее объявление

int ****a*, n, m, i, j, *и другие* …;

****a*** – указатель на указатель для создания динамического двухмерного массива, ***i*** , ***j*** – текущие индексы для строк и столбцов, ***n*** – количество строк, ***m*** – столбцов (размеры вводим с клавиатуры).

1) Ввод элементов массива:

```
    for (i = 0; i < n; i++)
for (j = 0; j < m; j++)
{
     cout << " a[ " << i+1 << " ] [ " << j+1 << " ] =  " ;
     cin >> a[i][j];
}
```

2) Заполнение массива ***a*** случайными числами в диапазоне [-10, 10] и вывод их на экран в виде матрицы

```
for (i = 0; i < n; i++) {
    for (j = 0; j < m; j++) {
        a[i][j] = random(21) – 10;
        cout << setw(5) << a[i][j] ;
    }
    cout << endl ;
}
```

3) Поиск минимального элемента массива ***a*** (в объявление добавим ***int*** переменные для индексов минимального элемента ***i_min*** – строка, ***j_min*** – столбец):

```
i_min = j_min = 0;
for (i = 0; i < n; i++)
    for (j = 0; j < m; j++)
        if ( a[i][j] < a[i_min][j_min] ) {
            i_min = i;
            j_min = j;
        }
cout << " Min = " << a[i_min][j_min]
    << " Row = " << i_min
     << " Col = " << j_min << endl ;
```

4) Поиск минимальных элементов в строках массива ***a*** (в объявление добавили указатель ***int *min*** для массива минимальных элементов):

```
min = new int [n];                    - Захват памяти
for (i = 0; i < n; i++) {
    min[i] = a[i][0];
    for (j = 1; j < m; j++)
        if ( a[i][j] < min [ i ] )
            min[i] = a[i][j];
}
```

5) Сортировка строк массива ***a*** по **???** минимальных элементов в строке (в объявление добавили указатель ***int *pr*** для перестановки строк и переменную ***int r*** для перестановки значений минимальных элементов):

```
for (i = 0; i < n – 1; i++)
for (j = i+1; j < n; j++)
    if ( min[i] > min[j] ) {
     r = min[i]; min[i] = min[j]; min[j] = r;
```

- Перестановка ***i****-го* и ***j****-го* ***значений*** массива ***min***

```
     pr = a[i]; a[i] = a[j]; a[j] = pr;
```

- Перестановка ***i****-й* (a[i]) и ***j****-й* (a[j]) строк массива ***a*** с помощью ***указателей***

```
     }
```

6) Сортировка строк массива ***a*** по **???** первых элементов строк (указатель ***int *pr*** используем для перестановки строк):

```
for (i = 0; i < n – 1; i++)
for (j = i+1; j < n; j++)
    if ( a[i][0] < a[j][0] ){
     pr = a[i];
     a[i] = a[j];
     a[j] = pr;
     }
```

7) Найти количество различных элементов массива ***a*** (повторяющиеся элементы считать только один раз).

Такого типа задачи проще решаются с использованием одномерных массивов, поэтому из 2-х мерного массива создаем одномерный (в объявление добавим переменные ***int nm, *b, k, kol***):

Создаем одномерный массив ***b*** размером ***n*m*** (***nm***):

nm = n*m; - Размер одномерного массива ***b***

b = new int [nm];

for (k = i = 0; i < n; i++)

for (j = 0; j < m; j++)

b [k++] = a[i][j]; - Постфиксный инкремент (***k++***) выполнится после операции присваивания

Рассмотрим два варианта решения этой задачи.

7.1) Посчитаем ***количество*** таких элементов, используя массив ***b*** отсортированный по возрастанию, после этого сравниваем стоящие рядом элементы:

```
for (i = 0; i < nm – 1; i++)
for (j = i+1; j < nm; j++)
    if ( b[i] < b[j] ){
     r = b[i];     b[i] = b[j];  b[j] = r;
    }
kol = 1;
for (i = 0; i < nm – 1; i++)
if ( b[i] != b[i + 1] )
     kol++;
cout << “\n\t Kol = “ << kol << endl;
```

7.2) В массиве ***b*** удалим (со сдвигом) все повторяющиеся элементы, после чего размер ***nm*** полученного массива ***b*** будет равен количеству искомых элементов, которые выведем на экран:

```
for (i = 0; i < nm – 1; i++)
for (j = i+1; j < nm; j++)
    if ( b[i] == b[j] ){
     for (k = j; k < nm – 1 ; k++)
         b[k] = b[k+1];
    nm--;
    j--;
     }
for (i = 0; i < nm; i++)
cout << setw(5) << b[i];
```

8) В массиве ***a*** найти минимальный элемент, лежащий выше побочной диагонали. Решение задач, где работа связана с диагональю, количество строк и столбцов рекомендуется задавать равными. Пример массива ***n*** = ***m*** = 4:

5	***4***	***3***	***2***
7	***1***	***2***	0
-1	***5***	2	-7
1	-8	3	0

Должны получить значение -1.

```
int  min = a[0][0];
for (i = 0; i < n – 1; i++)
    for (j = 0; j < n – 1 – i; j++)
        if ( a[i][j] < min )
            min = a[i][j];
cout << “ Min = “ << min << endl;
```

9) Создать массив ***b*** (одномерный), ***k***-й элемент которого равен **-1**, если все элементы ***k***-го столбца массива ***a*** меньше либо равны 0, иначе ***k***-й элемент равен **1** (объявляем указатель ***int *b*** для формируемого массива размером ***m***):

```
b = new  int [m];            - Создаем массив
for (j = 0; j < m ; j++) {       - Внешний цикл по столбцам
b[ j ] = -1;             - Предположим, что ВСЕ <=0
for (i = 0; i < n; i++)      - Внутренний цикл по строкам
    if ( a[i][j] > 0 ) {     - Находим хотя бы один, и
     b[ j ] = 1;          этого достаточно, чтобы
     break;               прекратить проверку
     }
cout << b[ j ] << endl;   - Вывод j-го элемента
}
```

Пример массива ***n*** = 3, ***m*** = 4:

-5	***4***	***-3***	**0**
-7	***-1***	***-2***	***2***
-1	**-5**	**-2**	**-7**

Должны получить значение вектора:

-1 1 -1 1

Во всех приведенных примерах не забываем освободить захваченную (с помощью операции ***new***) **память.**

Адресная функция (дополнительная информация)

При работе с массивами каждому массиву выделяется непрерывный участок памяти указанного размера.

При этом элементы, например, двухмерного массива x размером $n \times m$ размещаются в памяти по строкам, т.е. в следующей последовательности:

$x(0,0)$, $x(0,1)$,..., $x(0, m-1)$,..., $x(1,0)$, $x(1,1)$,..., $x(1,m-1)$,...,

$x(n-1,0)$, $x(n-1,1)$, $x(n-1,2)$,..., $x(n-1, m-1)$

Адресация элементов массива определяется некоторой адресной функцией, связывающей адрес и индексы элемента.

Адресная функция двухмерного массива x:

$$N1 = K(i, j) = m*i + j;$$

где $i = 0,1,2,\ldots, (n-1)$; $j = 0,1,2,\ldots, (m-1)$; j – изменяется в первую очередь.

Тогда справедливо следующее:

$$\boldsymbol{a}(i, j) \leftrightarrow \boldsymbol{b}(K(i, j)) = \boldsymbol{b}(N1),$$

b – одномерный массив с размером $N1 = n*m$.

Например, для двухмерного массива ***a***(2*3) имеем:

0,0	0,1	0,2	1,0	1,1	1,2
0	1	2	3	4	5

– Индексы массива ***a***

– Индексы массива ***b***

Проведем расчеты:

$i = 0, j = 0\ N1 = 3*0+0 = 0 \quad b(0)$

$i = 0, j = 1\ N1 = 3*0+1 = 1 \quad b(1)$

$i = 0, j = 2\ N1 = 3*0+2 = 2 \quad b(2)$

$i = 1, j = 0\ N1 = 3*1+0 = 3 \quad b(3)$

$i = 1, j = 1\ N1 = 3*1+1 = 4 \quad b(4)$

$i = 1, j = 2\ N1 = 3*1+2 = 5 \quad b(5)$

Аналогично получаем адресную функцию для трехмерного массива ***x***(*n*1, *n*2, *n*3):

$$K(i, j, k) = n3 * n2 * i + n2 * j + k ,$$

где $i = 0, 1, 2, \ldots , (n1 - 1)$; $j = 0, 1, 2, \ldots , (n2 - 1)$; $k = 0, 1, 2, \ldots , (n3 - 1)$; значение *k* – изменяется в первую очередь.

Для размещения такого массива потребуется участок памяти размером (*n*1 * *n*2 * *n*3) * *sizeof*(*type*).

Рассматривая такую область как одномерный массив ***y***(0, 1,..., *n*1**n*2**n*3), можно установить соответствие между элементом трехмерного массива ***x*** и элементом одномерного массива ***y*** :

$$\boldsymbol{x}(i, j, k) \leftrightarrow \boldsymbol{y}(K(i, j, k)) .$$

Необходимость введения адресных функций возникает лишь в случаях, когда требуется изменить способ отображения массива с учетом особенностей конкретной задачи.