УДК 373.167.1:004
ББК 32.97я72
К96

Одобрено Научно-редакционным советом корпорации «Российский учебник» под председательством академиков Российской академии наук В. А. Тишкова и В. А. Черешнева

Кушниренко, А. Г.

К96 Информатика : 9 класс : учебник / А. Г. Кушниренко, А. Г. Леонов, Я. Н. Зайдельман, В. В. Тарасова. — М. : Дрофа, 2018. — 238, [2] с. : ил. — (Российский учебник).

ISBN 978-5-358-18533-3

Учебник соответствует Федеральному государственному образовательному стандарту основного общего образования и входит в завершенную предметную линию учебников по информатике для 7—9 классов.

УДК 373.167.1:004
ББК 32.97я72

ISBN 978-5-358-18533-3

§ 1 Табличные величины и работа с ними

Секрет могущества компьютера — высокая скорость и большая память. В 7 и 8 классах вы уже научились использовать скорость работы компьютера: команды повторения (циклы) позволяют составлять короткие алгоритмы, при выполнении которых компьютер быстро совершает очень длинные последовательности действий. До сих пор, однако, во всех наших алгоритмах объем информации, хранимый в памяти компьютера в процессе выполнения алгоритма, был очень невелик.

А как быть, если в процессе научного исследования или, например, работы банковской системы нужно обработать сотню, тысячу или сотни тысяч чисел?

Понятно, что невозможно описывать в алгоритме такое количество отдельных величин. А ведь надо еще описать обработку этой информации, и если все числа обрабатываются одинаково, то хотелось бы иметь возможность задать обработку всех чисел единым алгоритмом, а не писать алгоритм отдельно для каждого числа.

Для этих целей применяется специальный способ организации информации — ***табличные величины*** (или просто ***таблицы***).

1.1. Простые и составные величины

- Вспомните, с какими типами величин вы встречались в 8 классе.

Величины в алгоритмическом языке делятся на ***простые*** и ***составные***. Целый и вещественный типы относятся к простым. Величины этого типа занимают одну ячейку памяти и в каждый момент времени имеют единственное значение.

Табличные величины относятся к ***составным***. Таблица — это набор данных простого типа, одинакового для всех величин, входящих в таблицу. Этот общий тип называется ***базовым типом*** таблицы.

Чаще всего используются числовые таблицы, состоящие из целых или вещественных величин.

Величины, составляющие таблицу, называются ее ***элементами***. Количество элементов называется ***размером*** таблицы.

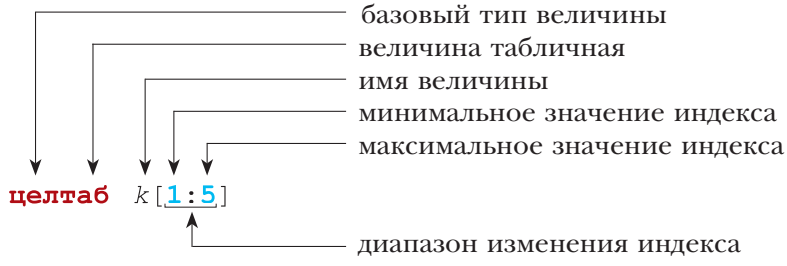
Например, длина таблицы для регистрации результатов прыжков в длину 8 спортсменов будет равна 8, т. е. в таблице будет 8 элементов. Значениями этих элементов будут дробные числа (результаты спортсменов в метрах), поэтому базовым типом таблицы будет **вещ**.

1.2. Описание таблицы и ее размещение в памяти компьютера

В памяти компьютера таблица занимает несколько ячеек памяти — по одной для каждого элемента. Все элементы таблицы нумеруются. Номера элементов называются ***индексами***. Нумерация чаще всего начинается с 1, но может начинаться с нуля или любого другого числа. При задании таблицы указывается ***диапазон изменения индекса*** — минимальное и максимальное значения индекса.

При составлении алгоритма табличные величины, как и любые другие величины, необходимо описывать.

Пример описания табличной величины:



Таким образом, описание таблицы включает следующую информацию: базовый тип; указание, что величина табличная; имя величины; минимальное значение индекса, максимальное значение индекса.

Зная диапазон изменения индекса, можно найти число элементов таблицы. Если минимальное значение индекса равно 1, то число элементов совпадает с максимальным значением индекса. Например, для диапазона 1 : 5 число элементов равно 5 и для таблицы **целтаб** *k*[1:5] выделяется 5 ячеек памяти базового типа **цел** (рис. 1).

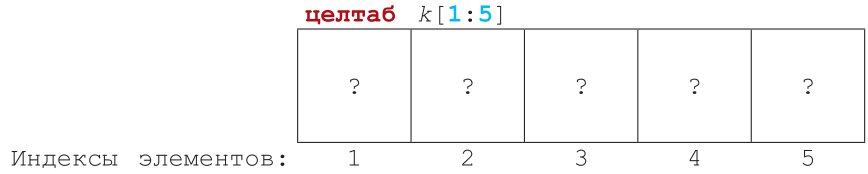


Рис. 1 Вновь созданная таблица из 5 элементов

В момент создания ни один элемент таблицы не содержит информации. Говорят, что элементы таблицы не определены. Таблица заполняется информацией в процессе работы алгоритма. Например, таблица k может в какой-то момент выглядеть так, как на рисунке 2.

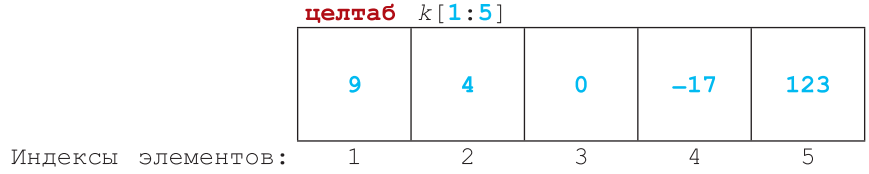


Рис. 2 Заполненная таблица из 5 элементов

На рисунке 2 показаны значения элементов таблицы k — пять целых чисел: {9, 4, 0, –17, 123}.

При описании таблицы можно задать начальные значения ее элементов, перечислив их в фигурных скобках через запятую. Например, описание

```
целтаб k[1:5]={11,22,33,44,55}
```

создает таблицу, в которой все элементы определены и имеют перечисленные значения (рис. 3).

Рис. 3 Таблица с начальными значениями элементов

1.3. Работа с элементами таблиц

Для обращения к конкретному элементу надо указать имя таблицы и в квадратных скобках после него — индекс нужного элемента. Например, если для таблицы, представленной на рисунке 3, выполнить команду

$$k[3] := k[2] + k[4],$$

то компьютер подставит вместо $k[2]$ и $k[4]$ значения второго и четвертого элементов таблицы k, т. е. числа 22 и 44, сложит их и присвоит третьему элементу полученное новое значение 66, стерев старое значение этого элемента 33 (рис. 4).

Рис. 4 Таблица после выполнения команды присваивания

При обращении к элементу таблицы индекс может быть не только числом, но и переменной или выражением. Это дает возможность использовать *цикл* для выполнения одинаковых действий со всеми элементами. Например, можно с помощью короткого алгоритма записать нулевые значения во все элементы большой таблицы (А1):

```
алг обнуление (рез вещтаб A[1:100])                    |A1|
надо | все элементы таблицы A равны 0
нач цел i
. i := 1                  | *
. нц пока i<=100          | *
. . A[i] := 0             | *
. . i := i+1              | *
. кц                      | *
кон
```

Алг. А1

Аналогично можно заполнить целочисленную таблицу так, чтобы значение каждого элемента было равно его индексу (А2):

```
алг заполнение индексом (рез целтаб A[1:100])          |A2|
надо | каждый элемент таблицы A равен своему индексу
нач цел i
. i := 1                  | *
. нц пока i<=100          | *
. . A[i] := i             | *
. . i := i+1              | *
. кц                      | *
кон
```

Алг. **A2**

1.4. Цикл для

Обратите внимание на строчки, отмеченные звездочками в алгоритмах (А1) и (А2). Они обеспечивают перебор всех элементов таблицы *A*. Величина *i* последовательно принимает значения 1, 2, ... и т. д. до 100, при каждом выполнении цикла обрабатывается элемент таблицы *A* с индексом *i*.

Такая схема поэлементной обработки очень часто встречается в различных задачах. Для упрощения составления новых и понимания уже составленных алгоритмов, содержащих подобную обработку, в алгоритмическом языке существует специальный вид цикла — цикл **для**. С использованием цикла **для** алгоритм (А1) можно переписать так:

```
алг обнуление (рез вещтаб A[1:100])                    |A3|
надо | все элементы таблицы A равны 0
нач цел i
. нц для i от 1 до 100
. . A[i] := 0
. кц
кон
```

Алг. **A3**

В общем случае цикл **для** обычно выглядит так:

```
нц для i от i1 до i2
. | тело цикла (последовательность команд)
кц
```

Здесь *i* — имя величины целого типа (она называется ***параметром цикла***), *i1* и *i2* — произвольные целые числа или выражения с целыми значениями. Тело цикла последовательно выполняется для *i* = *i1*, *i* = *i1* + 1, *i* = *i1* + 2, ..., *i* = *i2*.

По правилам алгоритмического языка *i1* и *i2* могут принимать любые целые значения. В частности, *i2* может быть равно *i1* — в этом случае тело цикла будет выполнено один раз. Значение *i2* может оказаться меньше *i1*. Этот случай не считается ошибочным: просто тело цикла не будет выполнено ни разу, а компьютер сразу перейдет к выполнению команд, записанных после **кц**.

В полном варианте цикла **для** можно задать не только начальное и конечное значения параметра цикла, но и шаг, с которым он будет изменяться. Это позволяет перебирать элементы таблицы более сложным образом. Можно, например, организовать перебор в обратном порядке:

```
нц для i от 10 до 1 шаг -1
. | тело цикла (последовательность команд)
кц
```

(тело цикла последовательно выполняется для *i* = 10, *i* = 9, *i* = 8, ..., *i* = 1) или перебрать элементы через один:

```
нц для i от 1 до 10 шаг 2
. | тело цикла (последовательность команд)
кц
```

(тело цикла последовательно выполняется для *i* = 1, *i* = 3, *i* = 5, *i* = 7, *i* = 9).

Полные правила записи и выполнения цикла **для** можно найти в документации к системе КуМир.

1.5. Таблицы вычисляемого размера

В алгоритмах (А2) и (А3) размер таблицы задан в заголовке алгоритма. Это не очень удобно: если потребуется заполнить таблицу с другим количеством элементов, придется составлять новый алгоритм.

С подобной проблемой мы встречались в 8 классе, когда составляли алгоритмы управления *Чертежником*. Тогда, чтобы можно было с помощью одного алгоритма строить изображения разного размера, мы стали использовать алгоритмы с аргументами.

Аналогичный прием применяется при обработке таблиц. В заголовке алгоритма описывается таблица ***вычисляемого размера***, например A[1:n]. При этом значение *n* должно быть уже известно (или вычислено) в момент создания такой таблицы. Чаще всего размер таблицы задается как аргумент алгоритма.

Например, алгоритм **обнуление** с использованием таблицы вычисляемого размера можно переписать так (А4):

```
алг обнуление (арг цел n, рез вещтаб A[1:n])          |A4|
надо | все элементы таблицы A равны 0
нач цел i
. нц для i от 1 до n
. . A[i] := 0
. кц
кон
```

Алг. **А4**

Обратите внимание: в заголовке алгоритма описание размера таблицы должно предшествовать описанию самой таблицы. При вызове алгоритма компьютер обрабатывает аргументы и результаты в том порядке, в котором они заданы в заголовке. Поэтому он сначала получит значение аргумента *n*, а потом использует это значение для создания таблицы нужного размера.

В дальнейшем мы будем использовать только таблицы вычисляемого размера. Для упрощения записи слово **арг** в начале заголовка алгоритма можно опускать, так как в алгоритмическом языке действует правило:

все параметры, описанные в заголовке до первого слова **рез** или **аргрез**, считаются аргументами.

Например, заголовок алгоритма **обнуление** можно записать так:

```
алг обнуление (цел n, рез вещтаб A[1:n])          |A4|
```

1.6. Виды задач по обработке таблиц

При решении на компьютере реальных задач обычно приходится обрабатывать большие объемы однородной информации. Чаще всего эта информация представляется в форме таблиц. Для обработки табличной информации применяются различные алгоритмы, в том числе довольно сложные.

Однако чаще всего обработка табличных величин сводится к применению нескольких сравнительно простых алгоритмов. Поэтому, если мы научимся составлять алгоритмы для решения некоторых наиболее часто встречающихся задач, то сможем успешно применять их и в более сложных задачах.

Чаще всего встречаются следующие виды задач:

1. ***Задачи заполнения.*** В этих задачах требуется заполнить таблицу информацией в соответствии с заданным правилом.

2. ***Задачи анализа.*** В этих задачах дана уже заполненная таблица и требуется найти какую-то ее характеристику (например, сумму всех элементов).

3. ***Задачи поиска.*** В этих задачах надо найти, есть ли в заполненной таблице элемент с заданными свойствами и если есть, то где он находится.

4. ***Задачи перестановки или перевычисления.*** В этих задачах требуется переставить местами элементы таблицы в соответствии с заданным правилом или вычислить новые значения элементов таблицы, заменив ими старые значения.

1.7. Задачи заполнения

В задачах заполнения таблица выступает как ***результат*** алгоритма. Обычно в этих задачах нужно перебрать все элементы (для этого удобно использовать цикл **для**) и записать в каждый из них нужную информацию.

Пример 1. Самый простой пример — заполнение таблицы нулями — был разобран ранее (алгоритм А4).

Пример 2. Заполним таблицу **целтаб** *A*[1:*n*] квадратами натуральных чисел. Надо получить A[1] = 1, A[2] = 4, A[3] = 9 и т. д.

```
алг квадраты (цел n, рез целтаб A[1:n])                    |A5|
надо | каждый элемент таблицы A равен
     | квадрату индекса этого элемента
нач цел i
. нц для i от 1 до n
. . A[i] := i**2
. кц
кон
```

Алг. **A5**

Пример 3. В математике хорошо известна ***числовая последовательность Фибоначчи***. Она начинается так: 1, 1, 2, 3, 5, 8, 13, 21... Первые два члена последовательности равны 1, а каждый следующий равен сумме двух предыдущих. Составим алгоритм, заполняющий таблицу числами Фибоначчи (А6).

```
алг Фибоначчи (цел n, рез целтаб A[1:n])                |А6|
дано n>1
надо | каждый элемент таблицы A с индексом i равен
     | члену последовательности Фибоначчи с номером i
нач цел i
. A[1] := 1; A[2] := 1
. нц для i от 3 до n
. . A[i] := A[i-2]+A[i-1]
. кц
кон
```

Алг. **А6**

Обратите внимание, что этот алгоритм работает правильно и при $n = 2$. В этом случае выполняются два присваивания до цикла, а цикл **для** не выполняется ни разу.

1.8. Задачи анализа. Однопроходные алгоритмы

В задачах анализа нужно найти какую-либо характеристику заданной таблицы. Например, найти сумму всех элементов таблицы

```
целтаб k[1:5]={2,4,3,17,12}
```

При этом таблица выступает как ***аргумент*** алгоритма (значения элементов таблицы исследуются, но не меняются), а результатом будет найденная (вычисленная) характеристика — в данном примере сумма элементов.

Обычно в подобных задачах нужно перебрать все элементы (для этого удобно использовать цикл **для**) и выполнить с каждым из них определенные действия.

При решении многих задач анализа можно применить специальный метод, который называется ***построением однопроходных алгоритмов***. При выполнении таких алгоритмов последовательно, за один проход, анализируются все элементы таблицы.

Суть этого метода такова. Мы рассматриваем фрагменты таблицы, состоящие из первого элемента, из первых двух элементов, из первых трех элементов и так далее до фрагмента, который охватывает всю таблицу, и при этом находим требуемую характеристику для первого фрагмента, потом для второго и т. д., пока, наконец, не охватим всю таблицу.

Найти требуемую характеристику для первого элемента обычно бывает нетрудно. Затем надо понять, как изменяется эта характеристика при добавлении к начальному фрагменту таблицы еще одного элемента, и выполнить эти изменения для всех элементов, начиная со второго.

Рассмотрим применение этого метода на примере.

Пример 1. Найти сумму всех элементов таблицы

```
целтаб k[1:5]={2,4,3,17,12}
```

Решение.

Фрагмент из 1 элемента: {2}	Сумма: 2
Фрагмент из 2 элементов: {2, 4}	Сумма: 2 + 4 = 6
Фрагмент из 3 элементов: {2, 4, 3}	Сумма: 6 + 3 = 9
Фрагмент из 4 элементов: {2, 4, 3, 17}	Сумма: 9 + 17 = 26
Фрагмент из 5 элементов: {2, 4, 3, 17, 12}	Сумма: 26 + 12 = 38

Ясно, что для первого элемента таблицы сумма равна этому элементу. Ясно также, что при добавлении к произвольному фрагменту таблицы еще одного элемента (выше он выделен красным цветом) сумма увеличивается на значение добавленного элемента. В общем случае получаем такой алгоритм[1]:

```
алг сумма (цел n, целтаб A[1:n], рез цел S)          |A7|
надо | S = сумма элементов таблицы A
нач цел i
. S := A[1]
. нц для i от 2 до n
. . S := S+A[i]
. кц
кон
```

Алг. A7

При выполнении этого алгоритма S последовательно принимает значения:

[1] Напомним, что по правилам алгоритмического языка первые два параметра алгоритма являются аргументами, хотя слово **арг** перед ними опущено.

```
A[1]
A[1]+A[2]
A[1]+A[2]+A[3]
. . .
A[1]+A[2]+ A[3]+ . . . +A[n]
```

Пример 2. Найти максимальное значение, встречающееся в числовой таблице **целтаб** *k*[1:4]={2,4,17,3}.

Решение.

Фрагмент из 1 элемента: {2}	Максимум: 2
Фрагмент из 2 элементов: {2, 4}	Максимум: max (2, 4) = 4
Фрагмент из 3 элементов: {2, 4, 17}	Максимум: max (4, 17) = 17
Фрагмент из 4 элементов: {2, 4, 17, 3}	Максимум: max (17, 3) = 17

Ясно, что для одного элемента максимум равен этому элементу. Ясно также, что при добавлении к произвольному фрагменту таблицы еще одного элемента нужно действовать так:

если добавленный элемент (выше он выделен красным цветом) больше ранее найденного максимума, этот элемент становится новым максимумом, в противном случае максимум не изменяется.

В общем случае получаем такой алгоритм:

```
алг максимум (цел n, целтаб A[1:n], рез цел макс) |A8|
надо | макс = максимальное число в таблице A
нач цел i
. макс := A[1]
. нц для i от 2 до n
. . если A[i]>макс
. . . то макс := A[i]
. . все
. кц
кон
```

Алг. **A8**

Обратите внимание, что этот алгоритм работает правильно и при $n = 1$. В этом случае выполняется одно присваивание до цикла, а цикл **для** не выполняется ни разу.

Операции нахождения максимума и минимума двух чисел встречаются в программировании очень часто, и потому для этих операций

во многих языках программирования предусматриваются стандартные функции. В алгоритмическом языке есть четыре функции:

imax, ***imin***, ***max***, ***min***.

Первые две функции позволяют находить максимум и минимум двух целых чисел, а последние две — максимум и минимум двух вещественных чисел. Используя функцию ***imax***, можно заменить в алгоритме А8 конструкцию

```
если A[i]>макс
. то макс := A[i]
все
```

на более короткую конструкцию

```
макс := imax(макс, A[i])
```

Кроме того, эти функции позволяют составить короткие алгоритмы нахождения максимума и минимума нескольких чисел, например:

```
алг цел imax3(цел a,b,c)
нач знач:=imax(imax(a,b),c)
кон

алг цел imin4(цел a,b,c,d)
нач знач:=imin(imin(a,b),imin(c,d))
кон
```

При составлении однопроходного алгоритма иногда бывает удобно начинать не с одноэлементного фрагмента таблицы, а с пустого фрагмента, не содержащего элементов вообще.

Например, для рассмотренной выше задачи о нахождении суммы элементов при таком подходе получается следующий алгоритм:

```
алг сумма (цел n, целтаб A[1:n], рез цел S)          |A9|
надо | S = сумма элементов таблицы A
нач цел i
. S := 0
. нц для i от 1 до n
. . S := S+A[i]
. кц
кон
```

Алг. **А9**

При выполнении этого алгоритма S последовательно принимает такие значения:

```
0
A[1]
A[1]+A[2]
A[1]+A[2]+A[3]
. . .
A[1]+A[2]+A[3]+ . . . +A[n]
```

Пример 3. Найти количество положительных чисел в таблице.

Решение. В этой задаче удобно начать с пустого фрагмента. В нем нет никаких чисел, в том числе и положительных, их количество равно нулю. Когда мы добавляем к очередному фрагменту еще один элемент, количество положительных чисел увеличивается на 1, если новый элемент положителен, в противном случае оно не изменяется.

```
алг число положительных(цел n, вещтаб A[1:n], рез цел k) |A10|
надо | k = число положительных элементов таблицы A
нач цел i
. k := 0
. нц для i от 1 до n
. . если A[i]>0
. . . то k := k+1
. . все
. кц
кон
```

Алг. **A10**

Метод однопроходных алгоритмов позволяет решать и более сложные задачи, алгоритм для которых не всегда удается найти другим способом.

Пример 4. Сколько раз встречается в таблице ее максимальный элемент?

Решение. Эту задачу можно легко решить в два прохода: сначала найти значение максимального элемента, а затем еще раз просмотреть всю таблицу и подсчитать количество элементов, равных максимальному. Но, рассуждая по правилам построения однопроходных алгоритмов, можно найти решение, позволяющее обойтись одним просмотром таблицы.

Во время этого просмотра мы будем для каждого фрагмента таблицы вычислять два числа:

— значение максимального элемента в этом фрагменте (величина **макс**);

— количество максимальных чисел (величина **чис**).

Начинаем с рассмотрения фрагмента таблицы из одного элемента. Единственный элемент фрагмента максимален, других максимальных элементов во фрагменте нет, значит, количество максимальных элементов во фрагменте равно 1.

При добавлении к уже рассмотренному фрагменту таблицы нового элемента (ниже он выделен красным цветом) возможны следующие случаи:

а) новый элемент равен имеющемуся максимуму — число максимумов увеличивается на 1;

б) новый элемент больше ранее найденного максимума — устанавливается новый максимум, а количество максимумов «сбрасывается» (становится равным 1);

в) случаи а) и б) не имеют места, т. е. новый элемент меньше ранее найденного максимума, — число максимумов не изменяется.

Найдем этим методом количество максимальных элементов в целочисленной таблице:

```
целтаб A[1:8]={3,2,3,1,4,4,1,4}
```

Фрагмент из 1 элемента: {3}	макс = 3 чис = 1
Фрагмент из 2 элементов: {3, 2}	макс = 3 чис = 1
Фрагмент из 3 элементов: {3, 2, 3}	макс = 3 чис = 2
Фрагмент из 4 элементов: {3, 2, 3, 1}	макс = 3 чис = 2
Фрагмент из 5 элементов: {3, 2, 3, 1, 4}	макс = 4 чис = 1
Фрагмент из 6 элементов: {3, 2, 3, 1, 4, 4}	макс = 4 чис = 2
Фрагмент из 7 элементов: {3, 2, 3, 1, 4, 4, 1}	макс = 4 чис = 2
Фрагмент из 8 элементов: {3, 2, 3, 1, 4, 4, 1, 4}	макс = 4 чис = 3

В общем виде эти рассуждения можно оформить в виде такого алгоритма:

```
алг число максимумов(цел n, целтаб A[1:n], рез цел чис) |A11|
надо | чис = число максимальных элементов в таблице A
нач цел i, цел макс
. макс := A[1]; чис := 1
. нц для i от 2 до n
. . выбор
. . . при A[i]=макс: чис := чис+1
. . . при A[i]>макс: чис := 1; макс := A[i]
. . все
. кц
кон
```

Алг. A11