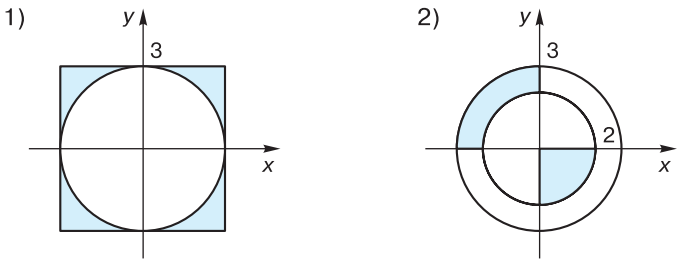
15. Разработайте программу, которая выводит сообщение «Да», если точка с координатами (x, y) принадлежит закрашенной области, и «Нет» в противном случае.



16. Шифр кодового замка является двузначным числом. Буратино забыл код, но помнит, что сумма цифр этого числа, сложенная с их произведением, равна самому числу. Напишите все возможные варианты кода, чтобы Буратино смог быстрее открыть замок. Решите задачу методом перебора.

§ 8 Структурированные типы данных. Массивы

Мы повторили основные приёмы работы с простыми типами данных. Из элементов простых типов в языке Pascal можно образовывать составные типы данных (структуры данных). Примером таких структур являются одномерные массивы.

Массив — это поименованная совокупность однотипных элементов, упорядоченных по индексам, определяющим положение элемента в массиве.

8.1. Общие сведения об одномерных массивах

Массив в языке Pascal — это набор однотипных данных, причём количество этих данных фиксировано и определяется при описании массива. Все переменные, входящие в массив, имеют одно и то же имя — имя массива, а различаются они по индексу — номеру (месту) в массиве.

Описание массива выглядит так:

```
array [<тип индекса>] of <тип компонент>
```

Здесь:

- **array** и **of** — служебные слова («массив» и «из»);
- <тип индекса> — описание индексации компонент (элементов) массива;
- <тип компонент> — тип величин, составляющих массив.

Например:

- **var** *day*: **array** [1..365] **of** integer — 365 целочисленных элементов пронумерованы от 1 до 365;
- **var** *tem*: **array** [1..12] **of** real — 12 вещественных элементов пронумерованы от 1 до 12;
- **var** *ocenka*: **array** [2..5] **of** integer — 4 целочисленных элемента пронумерованы от 2 до 5;
- **const** *n* = 10; **var** *slovo*: array [1..*n*] **of** string — *n* строковых величин пронумерованы от 1 до *n*.

Вспомним основные приёмы работы с массивами.

Пример 1. Имеются сведения о количестве ежедневных осадков в течение июня месяца в некотором регионе. Требуется найти среднее количество осадков и вывести таблицу, в которой для каждого дня месяца указать количество осадков в этот день и его отклонение от среднемесячного значения.

Для решения этой задачи данные о количестве ежедневных осадков в течение месяца будут просмотрены дважды:

1) при поиске среднего значения;
2) при расчёте отклонения.

Для решения задачи нам понадобится массив из 30 вещественных чисел. Назовём его *osad*. В программе будет два цикла. В первом цикле мы будем вводить значения элементов массива и сразу же подсчитывать их сумму — по завершении цикла мы получим сумму осадков, выпавших в течение месяца. Во втором цикле мы будем выводить строки таблицы и вычислять значения отклонений.

При работе с элементами массива будем пользоваться переменной *osad*[*i*]; значение индекса *i* при этом будет изменяться от 1 до 30 с шагом 1. Для вычисления среднего значения задействуем вещественную переменную *sred*, присвоив ей начальное значение 0 и последовательно накапливая в ней сумму осадков, выпавших в течение месяца. Разделив по завершении цикла

полученное значение на 30, вычислим требуемое среднемесячное количество осадков и присвоим результат этой же переменной.

```
Program osadki;
var osad: array [1..30] of real;
    sred: real; i: integer;
begin
  sred:=0;
  writeln('Введите количество осадков по дням');
  for i:=1 to 30 do
    begin
      write(i, ' июня: ');
      readln(osad[i]);
      sred:=sred + osad[i]
    end;
  sred:=sred/30;
  writeln('День Количество осадков Отклонение');
  for i:=1 to 30 do
    writeln(i:3, osad[i]:15:3, osad[i] - sred:12:2)
end.
```

Найдите в Интернете информацию о количестве ежедневных осадков, выпавших в течение месяца, в вашем регионе. Используя эти данные, выполните программу в среде программирования Pascal.

Выполните аналогичные расчёты с помощью электронных таблиц.

Чаще всего массив обрабатывается в цикле **for**. Но при работе с массивами можно использовать и другие циклы.

Пример 2. Имеется массив символов. Требуется вывести на экран элементы данного массива в обратном порядке.

Элементами массива символов могут быть любые символы, имеющиеся на клавиатуре, причём каждому элементу соответствует именно один символ. Если в качестве элементов нашего массива рассматривать последовательности букв, образующие некоторое слово или фразу на естественном языке, то, решив поставленную задачу, мы научимся строить «перевёртыши» слов.

Будем рассматривать слова и фразы не более чем из 20 символов, задав соответствующую размерность массива:

```
simbol: array [1..20] of char;
```

Если какое-то слово или фраза будут короче, то часть массива окажется не занятой, но это не повлияет на работу программы. Договоримся признаком конца слова считать точку — ввод символов продолжается, пока не введена точка; после ввода точки ввод символов прекращается.

```
program slova;
var simbol: array [1..20] of char;
    i, n: integer;
begin
  writeln('Введите слово - цепочку символов —
           с точкой в конце');
  i:=0;
  repeat
    i:=i+1;
    read(simbol[i]);
  until simbol[i]='.';
  n:=i-1;
  writeln('Перевёрнутое слово: ');
  for i:=n downto 1 do
    write(simbol[i]);
end.
```

Запустите программу в среде программирования Pascal.

Модифицируйте программу так, чтобы в начале её работы пользователю задавался вопрос о количестве символов, которые он будет вводить. Какой цикл при этом лучше использовать?

Как изменить программу, чтобы она выводила на экран в обратном порядке элементы целочисленного массива?

К типовым задачам обработки одномерных массивов, решаемым в процессе их однократного просмотра, относятся:

- задачи поиска элементов с заданными свойствами, в том числе максимумов и минимумов;
- проверка соответствия элементов массива некоторому условию (подсчёт количества или суммы элементов, удовлетворяющих некоторому условию; проверка соответствия всех элементов массива некоторому условию; проверка массива на упорядоченность и др.);
- задачи на удаление и вставку элементов массива;
- задачи на перестановку всех элементов массива в обратном порядке и т. д.

8.2. Задачи поиска элемента с заданными свойствами

Очень часто в реальной жизни нам приходится сталкиваться с задачей поиска информации в большом массиве данных. Например, поиск нужного слова в словаре, поиск времени отправления нужного поезда в расписании, поиск нужного товара в интернет-магазине и т. д.

В программировании поиск — одна из наиболее часто встречающихся задач невычислительного характера.

В алгоритмах поиска существует два возможных варианта окончания их работы: поиск может оказаться удачным — заданный элемент найден в массиве и определено его месторасположение, либо поиск может оказаться неудачным — необходимого элемента в данном объёме информации нет.

Рассмотрим несколько типовых задач поиска, первое знакомство с которыми у вас состоялось ещё в основной школе.

Пример 3. Последовательный поиск в неупорядоченном массиве.

Имеется массив $a[1..n]$; требуется найти элемент массива, равный p.

Алгоритм последовательного поиска в неупорядоченном массиве может быть следующим.

1. Установить $i = 1$.
2. Если $a[i] = p$, алгоритм завершил работу успешно.
3. Увеличить i на 1.
4. Если $i \le n$, то перейти к шагу 2. В противном случае алгоритм завершил работу безуспешно.

Возможная программа, реализующая этот алгоритм на языке Pascal, имеет вид:

```
const n=10;
var a: array [1..n] of integer; i, p: integer;
begin
  writeln('Ввод значений элементов массива:');
  for i:=1 to n do
    read(a[i]);
  write('Ввод p: ');
  readln(p);
  i:=1;
  while (i<=n) and (a[i]<>p) do i:=i+1;
```

```
  if i=n+1
    then writeln('Искомого элемента в массиве нет')
    else writeln('Искомый элемент a[', i, '] = ', a[i])
end.
```

Внимательно рассмотрите условие продолжения цикла. В каком случае выполнение цикла продолжается? В каких случаях осуществляется выход из цикла?

Запустите программу на выполнение в среде программирования Pascal.

Как иначе можно решить эту задачу, например, с использованием цикла **for**? Напишите соответствующую программу.

Оценим сложность рассмотренного алгоритма последовательного поиска, непосредственно зависящую от числа сравнений с искомым элементом. В худшем случае искомый элемент окажется на последнем месте или не будет найден вообще. В таком случае необходимо будет проделать n сравнений, т. е. сложность алгоритма будет равна $O(n)$.

Пример 4. Поиск максимумов и минимумов.

Имеется массив $a[1..n]$; требуется найти значение наибольшего (наименьшего) элемента массива.

Алгоритм поиска значения наибольшего (максимального) элемента в неупорядоченном массиве может быть следующим.

1. Установить значение текущего максимума равным первому исследуемому элементу ($max := a[1]$).
2. Установить счётчик равным 2 ($i := 2$).
3. Если исследованы ещё не все элементы ($i <= n$), то перейти к шагу 4, иначе алгоритм окончен (максимальный элемент равен max).
4. Если рассматриваемый элемент больше, чем текущий максимум ($a[i] > max$), то max присвоить значение $a[i]$.
5. Перейти к следующему элементу (увеличить i на единицу).
6. Перейти к шагу 3.

Возможная программа, реализующая этот алгоритм на языке Pascal, имеет вид:

```
const n=10;
var a: array [1..n] of integer;
    i, max: integer;
begin
  writeln('Ввод значений элементов массива:');
```

```
  for i:=1 to n do
    read(a[i]);
  max:=a[1];
  i:=2;
  while (i<=n) do
    begin
      if a[i] > max then max:=a[i];
      i:=i+1
    end;
  writeln('Max=', max)
end.
```

Запустите программу на выполнение в среде программирования Pascal.

Как иначе можно решить эту задачу, например, с использованием цикла **for**? Напишите соответствующую программу.

Преобразуйте программу так, чтобы с её помощью можно было находить минимальный элемент массива.

Какие изменения надо внести в программу для поиска индекса максимального (минимального) элемента массива?

Самостоятельно оцените сложность рассмотренного алгоритма.

8.3. Проверка соответствия элементов массива некоторому условию

Пример 5. Подсчёт количества элементов, удовлетворяющих некоторому условию.

Зачастую бывает важно выяснить, сколько элементов, обладающих определённым свойством, содержится в массиве.

Для решения этой задачи следует:

1) присвоить нулевое значение переменной, введённой для подсчёта количества элементов, удовлетворяющих заданному условию ($k := 0$);
2) организовать просмотр всех элементов массива: если просматриваемый элемент удовлетворяет заданному условию, значение переменной k увеличивать на 1.

Фрагмент программы подсчёта количества элементов массива, например бóльших некоторого числа p, имеет вид:

```
k:=0;
for i:=1 to n do
    if a[i]>p then k:=k+1;
```

Запишите полный текст программы и выполните её на компьютере для рассматриваемого в примере 8 массива *a*, состоящего из семи элементов, и числа *p* = 15.

Как модифицировать программу, чтобы можно было вычислить сумму элементов массива, бóльших некоторого числа *p*?

Пример 6. Проверка соответствия всех элементов массива некоторому условию.

Для того, чтобы установить факт соответствия всех элементов массива некоторому условию достаточно:

1) подсчитать количество элементов массива, соответствующих заданному условию;
2) сравнить найденное количество с общим числом элементов массива и вывести соответствующий результат.

Самостоятельно разработайте программу, позволяющую определить, все ли элементы массива являются двузначными числами. Выполните её на компьютере для рассматриваемого в примере 8 массива *a*, состоящего из семи элементов.

Пример 7. Проверка массива на упорядоченность.

Рассмотрим алгоритм, позволяющий определить, упорядочены ли элементы массива $a[1..n]$ по неубыванию, т. е. каждый элемент массива с 1-го по $(n-1)$-й не больше последующего.

Самый простой путь решения этой задачи — проверить, есть ли в массиве такие пары элементов, что $a[i] > a[i+1]$. Если подобные пары элементов есть, то массив не упорядочен по неубыванию, а если таких пар нет, то упорядочен.

В программе будем использовать логическую переменную *flag*:

- если *flag* = true, то массив упорядочен;
- если *flag* = false, то массив неупорядочен.

Ниже представлен фрагмент программы, реализующей этот алгоритм:

```
flag:=true;
for i:=1 to n-1 do
    if a[i]>a[i+1] then flag:=false;
```

Запишите полный текст программы и выполните её на компьютере для рассматриваемого в примере 8 массива *a*, состоящего из семи элементов.

Как можно решить эту же задачу путём подсчёта количества пар элементов массива, таких что *a*[*i*] > *a*[*i* + 1] (*a*[*i*] <= *a*[*i* + 1])?

8.4. Удаление и вставка элементов массива

Пример 8. Удаление из массива элемента с индексом *k*.

Имеется одномерный целочисленный массив из семи элементов:

i	1	2	3	4	5	6	7
a[*i*]	10	12	5	8	4	15	20

Удалим из массива элемент с индексом $k = 4$, а все элементы, расположенные справа от него, сдвинем на одну позицию влево. Получим следующий целочисленный массив из шести элементов:

i	1	2	3	4	5	6
a[*i*]	10	12	5	4	15	20

При удалении из массива любого из элементов размерность массива уменьшается на 1.

Мы видим, что элементы с индексами от 1 до $k - 1$ не изменились. На место элемента с индексом k (4) переместился элемент, имевший индекс $k + 1$ (5), на место элемента с индексом $k + 1$ (5) переместился элемент, имевший индекс $k + 2$ (6) и т. д.

В общем случае, фрагмент программы удаления из массива $a[1..n]$ элемента с индексом k и последующим сдвигом всех расположенных справа от него элементов на одну позицию влево имеет вид:

```
for i:=k to n-1 do
  a[i]:=a[i+1];
```

Запишите полный текст программы и выполните её на компьютере для рассмотренного выше массива *a*.

Пример 9. Вставка в массив элемента на место с индексом *k*.

Будем работать с тем же массивом из семи элементов. Но теперь наша задача будет состоять в том, чтобы вставить в массив на место с индексом $k = 4$ (т. е. после элемента с индексом $k - 1$) ещё один элемент, имеющий значение 11.

Получим следующий целочисленный массив из восьми элементов:

i	1	2	3	4	5	6	7	8
$a[i]$	10	12	5	11	8	4	15	20

При вставке в массив ещё одного элемента размерность массива увеличивается на 1. Это надо учесть при описании массива.

Итак, $a[4] := 11$. Элементу $a[5]$ следует присвоить то значение, которое было у $a[4]$, элементу $a[6]$ — значение, которое было у $a[5]$ и т. д. В общем случае, элементу $a[k + 1]$ следует присвоить то значение, которое было у $a[k]$.

Подумайте, что получится в результате выполнения следующих групп операторов присваивания:

1) `a[4]:=11; a[5]:=a[4]; a[6]:=a[5]; a[7]:=a[6]; a[8]:=a[7];`
2) `a[8]:=a[7]; a[7]:=a[6]; a[6]:=a[5]; a[5]:=a[4]; a[4]:=11;`

В общем случае, фрагмент программы вставки в массив $a[1..n - 1]$ элемента на место с индексом k и сдвигом k-го, $(k + 1)$-го, ..., $(n - 1)$-го элементов на одну позицию вправо имеет вид:

```
for i:=n downto k+1 do
  a[i]:=a[i-1];
a[k]:=<значение элемента>;
```

Запишите полный текст программы и выполните её на компьютере для рассмотренного выше массива a. Помните, что при описании массива надо учесть размерность массива, получающегося в результате работы программы.

8.5. Перестановка всех элементов массива в обратном порядке

Пример 10. Перестановка всех элементов массива $a[1..n]$ в обратном порядке сводится к тому, что меняются местами первый и последний элементы, второй и предпоследний элементы и т. д.

Перестановка нашего массива из семи элементов даст такой результат:

i	1	2	3	4	5	6	7
$a[i]$	20	15	4	8	5	12	10

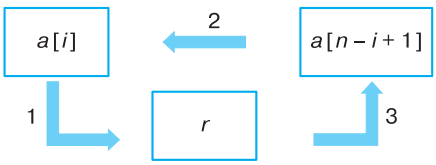
В общем случае, меняются местами элементы $a[i]$ и $a[n - i + 1]$.

Вспомним, как можно произвести обмен значений между двумя переменными. Выполнение операторов:

```
a[1]:=a[n]; a[n]:=a[1];
```

к желаемому результату не приводит. Самый простой вариант — использование вспомогательной переменной:

```
r:=a[i];
a[i]:=a[n-i+1];
a[n-i+1]:=r;
```



Выясним, сколько всего операций обмена следует произвести.

Если произведена перестановка, например, первого и последнего элементов, то одновременно произведена и перестановка последнего и первого элементов. Таким образом, если число элементов массива чётное, то достаточно произвести $n/2$ операций обмена.

Но что происходит, если массив содержит нечётное число элементов? Например, в нашем массиве из семи элементов выполнялось три обмена, а четвёртый элемент, занимающий центральную позицию, оставался на своём месте. В общем случае число операций обмена при перестановке в обратном порядке всех n элементов массива определяется как n div 2.

В общем случае, фрагмент программы по перестановке в обратном порядке всех элементов массива $a[1..n]$ имеет вид:

```
for i:=1 to n div 2 do
  begin
    r:=a[i];
    a[i]:=a[n-i+1];
    a[n-i+1]:=r
  end
```

Запишите полный текст программы и выполните её на компьютере для рассмотренного выше массива *a*, состоящего из семи и из шести элементов.

8.6. Сортировка массива

Сортировка — один из наиболее распространённых процессов современной обработки данных.

Сортировка — это распределение элементов массива в соответствии с определёнными правилами.

Под сортировкой (упорядочением) массива понимают перераспределение значений его элементов в некотором определённом порядке.

Порядок, при котором в массиве первый элемент имеет самое маленькое значение, а значение каждого следующего элемента не меньше значения предыдущего элемента, называют неубывающим.

Порядок, при котором в массиве первый элемент имеет самое большое значение, а значение каждого следующего элемента не больше значения предыдущего элемента, называют невозрастающим.

Цель сортировки — ускорить последующий поиск элементов, т. к. нужный элемент легче искать в упорядоченном массиве.

Рассмотрим и проанализируем несколько алгоритмов сортировки для решения следующей задачи. Дан одномерный массив целых чисел. Требуется отсортировать его так, чтобы все элементы были расположены в порядке неубывания: $a[i] <= a[i + 1]$.

Обменная сортировка методом «пузырька»

Своё название алгоритм получил благодаря следующей ассоциации: если сортировать этим алгоритмом массив по неубыванию, то максимальный элемент «тонет», а «лёгкие» элементы поднимаются на одну позицию к началу массива на каждом шаге алгоритма.

Пусть n — количество элементов в неупорядоченном массиве.

1. Поместим на место n-го элемента ($a[n]$) наибольший элемент массива. Для этого:
   1) положим $i = 1$;
   2) пока не обработана последняя пара элементов, т. е. $(n - 1)$-й и n-й элементы:
      - сравниваем i-й и $(i + 1)$-й элементы массива;
      - если $a[i] > a[i + 1]$ (элементы расположены не по порядку), то меняем элементы местами;
      - переходим к следующей паре элементов, сдвинувшись на один элемент вправо.
2. Повторяем пункт 1, каждый раз уменьшая размерность неупорядоченного массива на 1, до тех пор, пока не будет обработан массив из одной пары элементов (таким образом, на k-м просмотре будут сравниваться первые $(n - k)$ элементов со своими соседями справа).

Пример 11. Есть массив: 5 4 3 2 1. На примере этого массива подсчитаем количество элементарных действий в вычислительном процессе алгоритма сортировки методом «пузырька»:

- 1-я итерация: 4 3 2 1 5 (4 сравнения, 4 обмена);
- 2-я итерация: 3 2 1 4 5 (3 сравнения, 3 обмена);
- 3-я итерация: 2 1 3 4 5 (2 сравнения, 2 обмена);
- 4-я итерация: 1 2 3 4 5 (1 сравнение, 1 обмен).

Алгоритм закончил работу. Было сделано 10 сравнений и 10 обменов (4 + 3 + 2 + 1).

Этот алгоритм легко запоминается, но на практике он используется достаточно редко из-за квадратичной сложности, означающей, что в общем случае количество выполненных сравнений и обменов сопоставимо с n^2, где n — количество элементов массива.

Попытайтесь самостоятельно запрограммировать алгоритм сортировки методом «пузырька».

Сортировка выбором

Сортировка выбором (в порядке неубывания) осуществляется следующим образом:

1) в массиве выбирается минимальный элемент;
2) минимальный и первый элементы меняются местами (первый элемент считается отсортированным);
3) в неотсортированной части массива снова выбирается минимальный элемент и меняется местами с первым неотсортированным элементом массива;
4) действия, описанные в пункте 3, повторяются с неотсортированными элементами массива до тех пор, пока не останется один неотсортированный элемент (его значение будет максимальным).

Пример 12. Есть массив: 5 4 3 2 1.
1-я итерация: 1 4 3 2 5 (4 сравнения, 1 обмен).
2-я итерация: 1 2 3 4 5 (3 сравнения, 1 обмен).
3-я итерация: 1 2 3 4 5 (2 сравнения, 0 обменов).
4-я итерация: 1 2 3 4 5 (1 сравнение, 0 обменов).

В общем случае алгоритм сортировки выбором имеет квадратичную сложность относительно операций сравнения и линейную сложность относительно операций обменов. Этот алгоритм целесообразно применять, когда операция обмена над элементами массива особенно трудоёмка (например, если элементом массива является запись с большим числом полей).

Приведём фрагмент программы, реализующей описанный выше алгоритм:

```
for i:=1 to n-1 do
  begin
    imin:=i;
    for j:=i+1 to n do
      if a[j]<a[imin] then imin:=j;
    per:=a[i];
    a[i]:=a[imin];
    a[imin]:=per;
  end;
```

Запишите полный текст программы и выполните её на компьютере для рассмотренного выше массива.

САМОЕ ГЛАВНОЕ

Из элементов простых типов в языке Pascal можно образовывать составные типы данных (структуры данных). Примером таких структур являются одномерные массивы.

Массив в языке Pascal — это набор однотипных данных, причём количество этих данных фиксировано и определяется при описании массива. Все переменные, входящие в массив, имеют одно и то же имя — имя массива, а различаются они по индексу — номеру (месту) в массиве.

Перед использованием в программе массив должен быть описан, т. е. должно быть указано имя массива, количество элементов массива и их тип. Это необходимо для того, чтобы выделить в памяти под массив блок ячеек нужного типа.

Чаще всего массив обрабатывается в цикле **for**. Но при работе с массивами можно использовать и другие циклы.

К типовым задачам обработки одномерных массивов, решаемым в процессе их однократного просмотра, относятся:

- задачи поиска элемента с заданными свойствами, в том числе максимумов и минимумов;
- проверка соответствия элементов массива некоторому условию (подсчёт количества или суммы элементов, удовлетворяющих некоторому условию; проверка соответствия всех элементов массива некоторому условию; проверка массива на упорядоченность и др.);
- задачи на удаление и вставку элементов массива;
- задачи на перестановку всех элементов массива в обратном порядке и т. д.

Сортировка — один из наиболее распространённых процессов современной обработки данных. Под сортировкой (упорядочением) массива понимают перераспределение значений его элементов в некотором определённом порядке.

Вопросы и задания

1. Приведите примеры задач поиска информации в больших массивах данных.
2. Почему важно уметь решать задачи, связанные с обработкой массивов, путём однократного просмотра массива?
3. Программист написал программу суммирования элементов массива, но допустил в ней ошибку.

```
Program summa;
const n=10;
var a: array [1..n] of integer; s, i: integer;
begin
  s:=0;
  for i:=1 to n do
    begin
      readln(a[i]);
      s:=s+i
    end;
  writeln('s=', s)
end.
```

1) Что получится в результате выполнения этой программы, если в качестве элементов массива ввести числа: 1, –2, 3, –4, 5, –6, 7, –8, 9, –10?
2) Придумайте пример такого массива, обработка которого с помощью этой программы приводила бы к правильному результату.
3) Найдите ошибку, допущенную программистом.

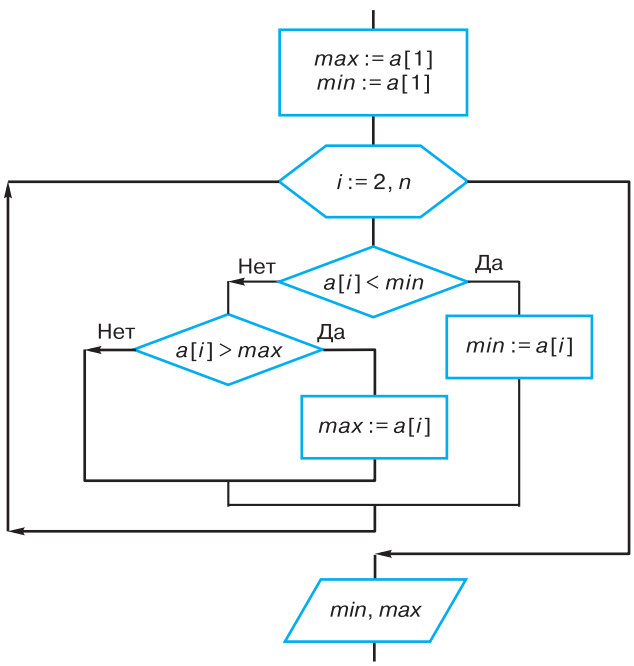
4. Программист написал программу нахождения произведения элементов массива, но допустил в ней ошибку.

```
Program proizv;
const n=10;
var a: array [1..n] of integer; p, i: integer;
begin
  p:=0;
  for i:=1 to n do
```

```
    begin
      readln(a[i]);
      p:=p*a[i]
    end;
  writeln('p=', p)'
end.
```

1) Что получится в результате выполнения этой программы, если в качестве элементов массива ввести числа: 1, –2, 3, –4, 5, –6, 7, –8, 9, –10?
2) Придумайте пример такого массива, обработка которого с помощью этой программы приводила бы к правильному результату.
3) Найдите ошибку, допущенную программистом.

5. На блок-схеме представлен алгоритм одновременного поиска максимального и минимального значений элементов массива:



Реализуйте этот алгоритм на языке программирования и выполните программу для массива из задания 6.

6. Имеется одномерный целочисленный массив из семи элементов:

i	1	2	3	4	5	6	7
a[*i*]	10	12	5	8	4	15	20

Каким будет результат преобразования массива по следующему алгоритму?

```
for i:=k+1 to n do
  a[i-1]:=a[i];
```

7. Имеется ли разница между операциями вставки в массив элемента на место с индексом k и замены значения элемента массива с индексом k? Обоснуйте свой ответ.

8. Имеется одномерный целочисленный массив из семи элементов:

i	1	2	3	4	5	6	7
a[*i*]	10	12	5	8	4	15	20

Каким будет результат преобразования массива по следующему алгоритму?

```
for i:=1 to n div 2 do
  begin
    r:=a[i];
    a[i]:=a[n-i+1];
    a[n-i+1]:=r
  end;
```

9. Дана программа:

```
const n=5;
const a: array[1..n] of integer=(1,2,6,4,6);
var i, max1, max2: integer;
begin
  max1:=a[1];
  max2:=a[2];
  for i:=2 to n do
    if a[i]>max1
    then begin max2:=max1; max1:=a[i]; end
    else if a[i]>max2 then max2:=a[i];
  writeln('max1=', max1, ', max2=', max2);
end.
```

Что получится в результате выполнения этой программы? Какую задачу решает эта программа?

10. Дано натуральное десятичное число $n <= 32\,000$. Напишите программу, в которой:
    1) из цифр данного числа формируется одномерный целочисленный массив;
    2) определяются наибольшая и наименьшая цифры данного числа;
    3) находятся сумма и произведение цифр, образующих данное число.
11. Требуется упорядочить по весу в порядке неубывания n непрозрачных банок с чаем, имея в своём распоряжении только чашечные весы без гирь. Опишите возможный алгоритм решения этой задачи.

§ 9 Структурное программирование

9.1. Общее представление о структурном программировании

Программирование как род занятий и сфера деятельности интенсивно развивается со второй половины прошлого века. За это время сложились определённые технологии, способствующие повышению производительности труда программистов, в том числе сокращению числа ошибок, упрощению отладки, модификации и сопровождения программного обеспечения. Особенно это важно при разработке больших и сложных программных комплексов, осуществляемой усилиями целых коллективов программистов.

Одна из таких технологий — структурное программирование — была разработана ещё в начале 70-х годов прошлого века и связана с именем выдающегося нидерландского ученого Эдсгера Дейкстры (1930–2002).

Структурное программирование — технология разработки программного обеспечения, в основе которой лежит представление программы в виде иерархической структуры логически целостных фрагментов (блоков).

Перечислим некоторые принципы структурного программирования.

1. Любая программа строится из трёх базовых управляющих конструкций: последовательность, ветвление, цикл.
2. В программе базовые управляющие конструкции могут быть вложены друг в друга произвольным образом.
3. Повторяющиеся фрагменты программы можно оформить в виде подпрограмм (процедур и функций). В виде подпрограмм можно оформить логически целостные фрагменты программы, даже если они не повторяются.
4. Все перечисленные конструкции должны иметь один вход и один выход.
5. Разработка программы ведётся пошагово, методом «сверху вниз».

О методе разработки алгоритма «сверху вниз» вы получили представление в курсе информатики основной школы. Напомним его ключевые моменты на примере разработки некоторой программы.

Сначала пишется короткий текст основной программы. В ней вместо каждого логически целостного фрагмента вставляется вызов подпрограммы, которая будет выполнять этот фрагмент. Вместо настоящих, работающих, подпрограмм в программу вставляются так называемые заглушки. Как правило, они удовлетворяют требованиям интерфейса заменяемого фрагмента, но не выполняют его функций.

На следующем шаге следует убедиться, что подпрограммы вызываются в правильной последовательности, т. е. верна общая структура программы.

После этого подпрограммы-заглушки последовательно заменяются на полнофункциональные, причём разработка каждой подпрограммы ведётся тем же методом, что и основной программы. На каждом этапе проверяется, что уже созданная программа правильно работает по отношению к подпрограммам более низкого уровня.

Разработка заканчивается тогда, когда ни на одном уровне не останется ни одной заглушки. Полученная программа проверяется и отлаживается.

Такая последовательность гарантирует, что на каждом этапе разработки программист будет иметь дело с обозримым и понятным ему множеством фрагментов, осознавая, что общая структура всех более высоких уровней программы верна.

9.2. Вспомогательный алгоритм

Пример 1. Применим метод «сверху вниз» для разработки алгоритма нахождения периметра треугольника, заданного координатами своих вершин.

Пусть XA, XB, YA, YB, XC, YC — координаты вершин треугольника ABC. Его периметр — сумма длин отрезков AB, BC и AC.

Из курса геометрии вам известна формула для вычисления длины отрезка AB по координатам его концов (рис. 2.11):

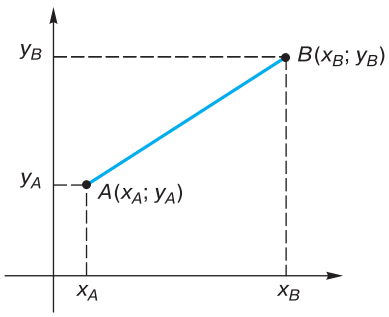


Рис. 2.11. Отрезок AB

$$d = \sqrt{(XA - XB)^2 + (YA - YB)^2}.$$

Действия по вычислению длины отрезка представляют собой логически целостный фрагмент, который целесообразно оформить в виде вспомогательного алгоритма.

Вспомогательный алгоритм — это алгоритм, целиком используемый в составе другого алгоритма.

На рисунке 2.12 представлены:

1) блок-схема алгоритма вычисления периметра треугольника, предполагающая вызов вспомогательного алгоритма Отрезок;
2) блок-схема вспомогательного алгоритма Отрезок.

При вызове вспомогательного алгоритма указываются его параметры (входные данные и результаты). Параметрами вспомогательного алгоритма Отрезок являются величины $X1$, $Y1$, $X2$, $Y2$, D. Это формальные параметры, они используются при описании алгоритма. При конкретном обращении к вспомогательному алгоритму формальные параметры заменяются фактическими параметрами, т. е. именно теми величинами, для которых будет исполнен вспомогательный алгоритм. Типы, количество и порядок следования формальных и фактических параметров должны совпадать.

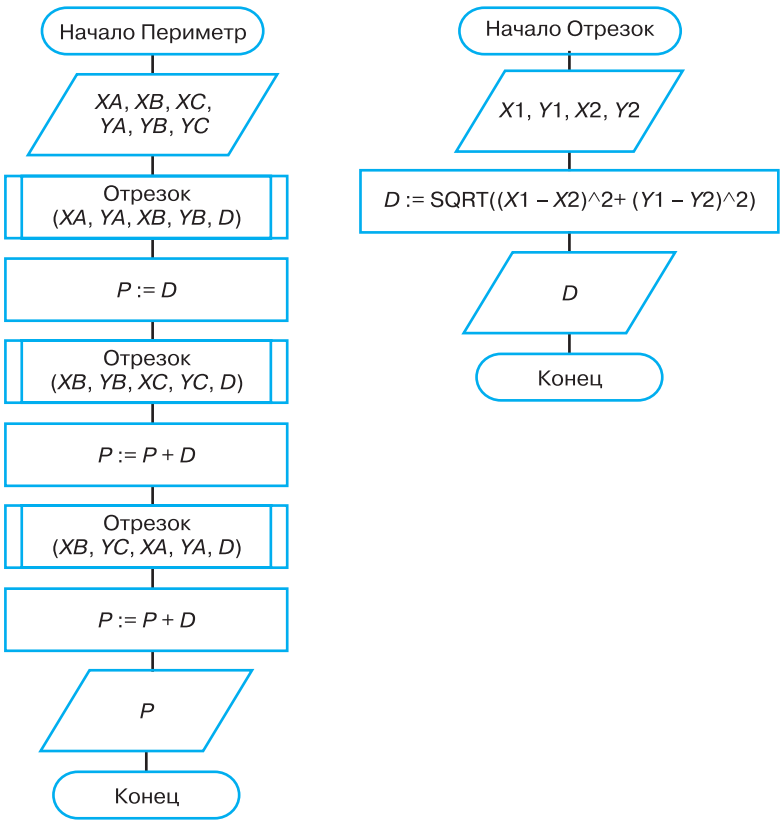


Рис. 2.12. Алгоритм вычисления периметра треугольника и вспомогательный алгоритм Отрезок

Команда вызова вспомогательного алгоритма исполняется следующим образом:

1) формальные входные данные вспомогательного алгоритма заменяются значениями фактических входных данных, указанных в команде вызова вспомогательного алгоритма;
2) для заданных входных данных исполняются команды вспомогательного алгоритма;
3) полученные результаты присваиваются переменным с именами фактических результатов;
4) осуществляется переход к следующей команде основного алгоритма.

Каким будет результат работы алгоритма при следующих исходных данных: $XA = 1$, $XB = 2$, $XC = 3$, $YA = 1$, $YB = 3$, $YC = 1$.

9.3. Рекурсивные алгоритмы

Алгоритм называется **рекурсивным**, если на каком-либо шаге он прямо или косвенно обращается сам к себе.

Пример 2. Как известно, факториал натурального числа n определяется следующим образом: $n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n$; $0!$ считается равным единице $(0! = 1)$.

Иначе это можно записать так:

$F(n) = 1$ при $n \le 1$;
$F(n) = F(n - 1) \cdot n$ при $n > 1$.

В определении факториала через рекурсию имеется условие $n \le 1$, при достижении которого вызов рекурсии прекращается.

В рекурсивном определении должно присутствовать ограничение (граничное условие), при выходе на которое дальнейшая инициация рекурсивных обращений прекращается.

Пример 3. Определим функцию $S(n)$, вычисляющую сумму цифр в заданном натуральном числе n:

$S(n) = n$ при $n < 10$;
$S(n) = S(n \text{ div } 10) + n \bmod 10$ при $n \ge 10$.

Самостоятельно определите функцию $K(n)$, которая возвращает количество цифр заданного натурального числа n.

Пример 4. Алгоритм вычисления значения функции $F(n)$, где n — натуральное число, задан следующими соотношениями:

$F(n) = 1$ при $n \le 2$;
$F(n) = F(n - 1) + 3 \cdot F(n - 2)$ при $n > 2$.

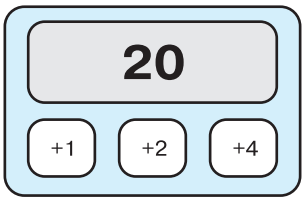
Требуется выяснить, чему равно значение функции $F(7)$.
По условию, $F(1) = F(2) = 1$.

$F(3) = F(2) + 3 \cdot F(1) = 1 + 3 \cdot 1 = 4$.
$F(4) = F(3) + 3 \cdot F(2) = 4 + 3 \cdot 1 = 7$.
$F(5) = F(4) + 3 \cdot F(3) = 7 + 3 \cdot 4 = 19$.
$F(6) = F(5) + 3 \cdot F(4) = 19 + 3 \cdot 7 = 40$.
$F(7) = F(6) + 3 \cdot F(5) = 40 + 3 \cdot 19 = 97$.

Подобные вычисления можно проводить в уме, а их результаты фиксировать в таблице:

n	1	2	3	4	5	6	7
$F(n)$	1	1	4	7	19	40	97

Пример 5. Исполнитель Плюс имеет следующую систему команд:



1) прибавь 1;
2) прибавь 2;
3) прибавь 4.

С помощью первой из них исполнитель увеличивает число на экране на 1, с помощью второй — на 2, с помощью третьей — на 4. Программа для исполнителя Плюс — это последовательность команд. Выясним, сколько разных программ, преобразующих число 20 в число 30, можно составить для этого исполнителя.

Количество программ, с помощью которых можно получить некоторое число n, будем рассматривать как функцию $K(n)$.

Число, меньшее 20, при заданных начальных условиях и системе команд исполнителя Плюс получить невозможно. Следовательно, при $n < 20$ $K(n) = 0$.

Для начального числа 20 количество программ равно 1: существует только одна пустая программа, не содержащая ни одной команды. Можем записать: $K(n) = 1$ при $n = 20$.

Любое число $n > 20$ может быть получено из чисел $n - 1$, $n - 2$ и $n - 4$ одной из трёх команд, входящих в систему команд исполнителя — «прибавь 1», «прибавь 2» и «прибавь 4» соответственно. При этом каждая программа получения из исходного числа чисел $n - 1$, $n - 2$ и $n - 4$ удлинится на одну команду и будет приводить к числу n. Следовательно, $K(n) = K(n - 1) + K(n - 2) + K(n - 4)$.

Запишем все соотношения, определяющие функцию $K(n)$:

$K(n) = 0$ при $n < 20$;

$K(n) = 1$ при $n = 20$;

$K(n) = K(n - 1) + K(n - 2) + K(n - 4)$ при $n > 20$.

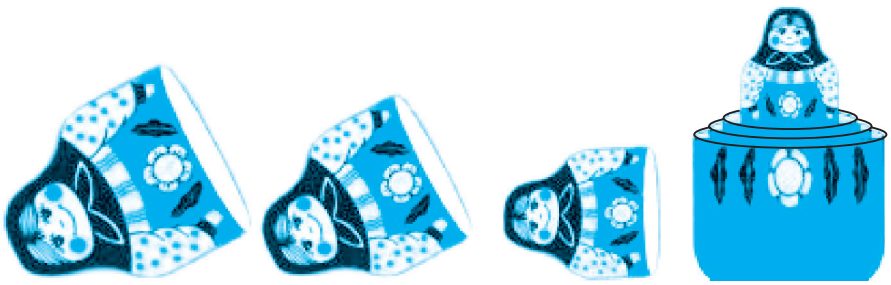
Заполним по этой формуле таблицу для всех значений n от 20 до 30:

n	20	21	22	23	24	25	26	27	28	29	30
$K(n)$	1	1	2	3	6	10	18	31	55	96	169

Итак, существует 169 различных программ, с помощью которых исполнитель Плюс может преобразовать число 20 в 30.

Любой объект, который частично определяется через самого себя, называется рекурсивным. Нас окружает множество рекурсивных объектов. Приведём примеры только некоторых из них.

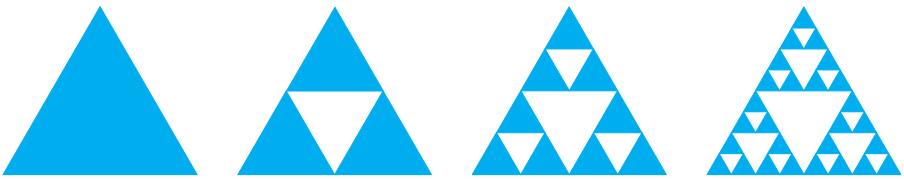
1. Матрёшка — русская деревянная игрушка в виде расписной куклы, внутри которой находятся подобные ей куклы меньшего размера.



2. Два зеркала, поставленные друг напротив друга, — в них образуются два коридора из затухающих отражений. Это, например, можно наблюдать в спальном железнодорожном вагоне.

3. Примером рекурсивной структуры является замечательное стихотворение Р. Бернса «Дом, который построил Джек» в переводе С. Маршака.

4. Рекурсивную природу имеют геометрические фракталы. На рисунке представлено построение одного из геометрических фракталов — треугольника Серпинского. Чтобы его получить, нужно взять равносторонний треугольник с внутренней областью, провести в нём средние линии и «выкинуть» центральный из четырёх образовавшихся маленьких треугольников. Дальше эти же действия нужно повторить с каждым из оставшихся трёх треугольников, и т. д.



9.4. Запись вспомогательных алгоритмов на языке Pascal

Запись вспомогательных алгоритмов в языках программирования осуществляется с помощью подпрограмм. В Паскале различают два вида подпрограмм: процедуры и функции.

Процедура — подпрограмма, имеющая произвольное количество входных и выходных данных.

Описание процедуры имеет вид:

```
procedure <имя_процедуры>(<описание параметров-значений>;
          var: <описание параметров-переменных>);
begin
  <операторы>
end;
```

В заголовке процедуры после её имени приводится перечень формальных параметров и их типов. Для вызова процедуры достаточно указать её имя со списком фактических параметров. При этом между фактическими и формальными параметрами должно быть полное соответствие по количеству, порядку следования и типу.

Пример 6. Запишем на языке Pascal программу нахождения периметра треугольника, заданного координатами его вершин. Вспомогательный алгоритм оформим с помощью процедуры.

```
program perimetr;
var xa,ya,xb,yb,xc,yc,d,p: real;
procedure otrezok(x1,y1, x2,y2: real; var d: real);
begin
  d:=sqrt(sqr(x1-x2)+sqr(y1-y2));
end;
begin
  p:=0;
  writeln('Ввод координат концов отрезка:');
  writeln('xa, ya');
  read(xa,ya);
  writeln('xb, yb');
  read(xb,yb);
  writeln('xc, yc');
  read(xc,yc);
  otrezok(xa,ya,xb,yb,d);
  p:=p+d;
  otrezok(xa,ya,xc,yc,d);
  p:=p+d;
  otrezok(xc,yc,xb,yb,d);
  p:=p+d;
  writeln('p=',p)
end.
```

Выполните программу на компьютере.

Подумайте, каким образом можно модифицировать программу, чтобы вычислять с её помощью периметр *n*-угольника. Каким образом при решении этой задачи можно использовать массивы?

Функция — подпрограмма, имеющая единственный результат, записываемый в ячейку памяти, имя которой совпадает с именем функции.

Описание функции имеет вид:

```
function  <имя_функции>(<описание входных данных>):
          <тип_функции>);
begin
  <операторы>
end;
```

В заголовке функции после её имени приводится описание входных данных — указывается перечень формальных параметров

и их типов. Там же указывается тип самой функции, т. е. тип результата. В блоке функции обязательно должен присутствовать оператор

```
<имя_функции> := <результат>;
```

Для вызова функции достаточно указать её имя со списком фактических параметров в любом выражении, в условиях (после слов if, while, until) или в операторе write главной программы.

Пример 7. Запишем на языке Pascal программу нахождения периметра треугольника, заданного координатами его вершин. Вспомогательный алгоритм оформим с помощью функции.

```
program perimetr;
  var xa,ya,xb,yb,xc,yc,p: real;
function d(x1,y1, x2,y2: real): real;
begin
  d:=sqrt(sqr(x1-x2)+sqr(y1-y2));
end;
begin
  writeln('Ввод координат концов отрезка:');
  writeln('xa, ya');
  read(xa,ya);
  writeln('xb, yb');
  read(xb,yb);
  writeln('xc, yc');
  read(xc,yc);
  p:=p+d(xa,ya,xb,yb)+d(xa,ya,xc,yc)+d(xc,yc,xb,yb);
  writeln('p=',p)
end.
```

Выполните программу на компьютере.

На основе этой программы напишите функцию, вычисляющую площадь треугольника по целочисленным координатам его вершин. Используйте эту функцию для вычисления площади n-угольника.

САМОЕ ГЛАВНОЕ

Структурное программирование — технология разработки программного обеспечения, в основе которой лежит представление программы в виде иерархической структуры логически целостных фрагментов (блоков).

Основные принципы структурного программирования заключаются в том, что:

1) любая программа строится из трёх базовых управляющих конструкций: последовательность, ветвление, цикл;
2) в программе базовые управляющие конструкции могут быть вложены друг в друга произвольным образом;
3) повторяющиеся фрагменты программы можно оформить в виде подпрограмм (процедур и функций). В виде подпрограмм можно оформить логически целостные фрагменты программы, даже если они не повторяются;
4) все перечисленные конструкции должны иметь один вход и один выход;
5) разработка программы ведётся пошагово, методом «сверху вниз».

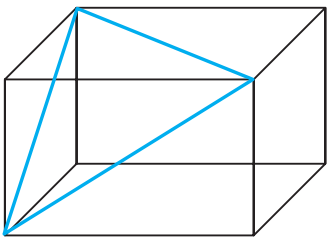
Вспомогательный алгоритм — это алгоритм, целиком используемый в составе другого алгоритма.

Алгоритм называется рекурсивным, если на каком-либо шаге он прямо или косвенно обращается сам к себе.

Запись вспомогательных алгоритмов в языках программирования осуществляется с помощью подпрограмм. В Паскале различают два вида подпрограмм: процедуры и функции.

Вопросы и задания

1. В чём заключается сущность структурного программирования? Какие преимущества обеспечивает эта технология?
2. Какой алгоритм называется вспомогательным?
3. Вспомните, в чём состоит суть метода последовательного построения (уточнения) алгоритма. Как он называется иначе?
4. Опишите основные шаги разработки программы методом «сверху вниз».
5. Дан прямоугольный параллелепипед, длины рёбер которого равны a, b и c. Требуется определить периметр треугольника, образованного диагоналями его граней. Какой алгоритм целесообразно использовать при решении этой задачи в качестве вспомогательного?



6. Какой вспомогательный алгоритм называется рекурсивным? Что такое граничное условие и каково его назначение в рекурсивном алгоритме?

7. Алгоритм вычисления значения функции $F(n)$, где n — натуральное число, задан следующими соотношениями:

$$F(n) = 2 \text{ при } n \le 0;$$
$$F(n) = F(n-2) + F(n-1) + F(n \text{ div } 2) \text{ при } n > 0.$$

Требуется выяснить, чему равно значение функции $F(10)$.

8. Исполнитель Калькулятор имеет следующую систему команд:
   1) прибавь 1;
   2) умножь на 2.

   С помощью первой из них исполнитель увеличивает число на экране на 2, с помощью второй — в 2 раза.
   1) Выясните, сколько разных программ, преобразующих число 1 в число 20, можно составить для этого исполнителя.
   2) Сколько среди них таких программ, у которых в качестве промежуточного результата обязательно получается число 15?
   3) Сколько среди них таких программ, у которых в качестве промежуточного результата никогда не получается число 12?

9. Попробуйте найти рекурсивные синтаксические структуры:
   1) в поэме А. Блока «Двенадцать»;
   2) в стихотворении М. Лермонтова «Сон»;
   3) в романе М. Булгакова «Мастер и Маргарита»;
   4) в фольклоре.

10. Найдите информацию о таких геометрических фракталах, как Снежинка Коха, Т-квадрат, Н-фрактал, кривая Леви, Драконова ломаная.

11. Напишите программу вычисления значения функции $F(n)$, рассмотренной в примере 4 этого параграфа. Вычислите с её помощью значение функции F(7).

12. Напишите программу вычисления $C_n^k = \dfrac{n!}{(n-k)! \cdot k!}$. Используйте подпрограмму.
13. Дана программа:

```
program rek;
procedure F(n: integer);
```

```
begin
  if n>0 then
  begin
    F(n-4);
    writeln(n);
    F(n div 3)
  end;
end;
begin
  F(9)
end.
```

Не выполняя программу на компьютере, выясните, что получится в результате работы этой программы.

Проверьте свой результат, выполнив программу на компьютере.

Дополнительные материалы к главе смотрите в авторской мастерской.