Д. М. Златопольский

ЕГЭ ПО ИНФОРМАТИКЕ

РЕШЕНИЕ ЗАДАЧ ПО ПРОГРАММИРОВАНИЮ

Санкт-Петербург
«БХВ-Петербург»
2013

УДК 681.3.068(075.3)
ББК 32.973.26-018.1я721
З-67

Златопольский Д. М.

З-67 ЕГЭ по информатике. Решение задач по программированию. — СПб.: БХВ-Петербург, 2013. — 304 с.: ил. — (ИиИКТ)

ISBN 978-5-9775-0868-1

Книга предназначена для подготовки учащихся к Единому государственному экзамену по информатике в части решения задач по программированию. Рассмотрена методика решения основных типовых задач по программированию, а также заданий из демонстрационных вариантов ЕГЭ и из пособий, написанных разработчиками контрольно-измерительных материалов по информатике. Книга предназначена также студентам вузов и колледжей, преподавателям информатики и другим читателям при изучении программирования вне связи с ЕГЭ.

Для образовательных учреждений

УДК 681.3.068(075.3)
ББК 32.973.26-018.1я721

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Людмила Еремеевская*
Зав. редакцией	*Екатерина Капалыгина*
Редактор	*Владимир Красовский*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Марины Дамбиевой*

Подписано в печать 31.08.12.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 24,51.
Тираж 1200 экз. Заказ №
"БХВ-Петербург", 191036, Санкт-Петербург, Гончарная ул., 20.

Первая Академическая типография "Наука"
199034, Санкт-Петербург, 9 линия, 12/28

ISBN 978-5-9775-0868-1

Оглавление

Предисловие

На Едином государственном экзамене по информатике и ИКТ задания, связанные с программированием, занимают важное место. Так в [14] их 8[1] (А2, А14, В3, В6, В7, С1, С2, С4) при общем числе заданий — 32. При этом высока весомость заданий (максимальный балл за выполнение заданий группы части 3 равен 3–4).

Это говорит о том, что от умения решать задачи по программированию в значительной степени зависит успешность сдачи ЕГЭ в целом.

В то же время, как показывает опыт, такие задачи часто вызывают у школьников заметные трудности, особенно задачи части 3. В большой степени это связано с недостаточным числом часов, отводимых на изучение программирования в школе.

Данная книга должна восполнить этот недостаток — помочь учащимся подготовиться к экзамену самостоятельно. В ней системно, подробно и доступно описана методика решения задач по программированию, встречающихся в ЕГЭ.

Сначала в *главах 1* и *2* рассмотрены практически все частные, вспомогательные задачи, умение решать которые позволит успешно выполнить задания экзамена по программированию, после чего в *главах 3–7* описана методика решения задач С2 и С4 из ЕГЭ [2–7, 12, 16]. В *приложениях* приведены другие материалы, связанные с задачами программирования в Едином государственном экзамене.

При разработке программ (частных и из заданий ЕГЭ) использован школьный алгоритмический язык [11, 13]. Русский синтаксис этого языка и большое число комментариев сделают программы максимально понятными и легко переносимыми на любой другой язык программирования. Актуальность этого языка повышается в связи с тем, что он является одним из двух языков программирования, которые планируется использовать в компьютерном варианте ЕГЭ по информатике (его введение ожидается в 2013 году). В большинстве случаев приводятся также фрагменты программ на языке Паскаль. Полностью программы решения задач С4 на

[1] Без учета заданий, связанных с формальными исполнителями.

школьном алгоритмическом языке представлены в архиве, который выложен на FTP-сервер издательства по адресу: **ftp://ftp.bhv.ru/9785977508681.zip**. Ссылка доступна и со страницы книги на сайте **www.bhv.ru**.

Обратим внимание на широкое использование в книге задач от разработчиков контрольно-измерительных материалов для ЕГЭ [12, 16] — существует большая вероятность того, что подобные задачи будут включены в экзамен в будущем.

Кроме учащихся, готовящихся к сдаче экзамена самостоятельно, книгу могут использовать учителя и преподаватели информатики, а также студенты и учащиеся, изучающие программирование вне связи с ЕГЭ.

ГЛАВА 1

Задачи из Кодификатора для ЕГЭ

В Кодификаторе элементов содержания и требований к уровню подготовки выпускников общеобразовательных учреждений для проведения в 2012 году Единого государственного экзамена по информатике и ИКТ [10] приведен список возможных алгоритмических задач для подраздела 1.1 перечня требований к уровню подготовки выпускников, достижение которого проверяется на Едином государственном экзамене по информатике и ИКТ.

В перечисленных задачах представлены основные базовые программные конструкции, встречающиеся в задачах по программированию в ЕГЭ (в заданиях всех трех частей экзамена: 1, 2 и 3). Существует также большая вероятность того, что ряд задач будет включен в экзамен в будущем.

Далее рассмотрены возможные методы решения задач. Как отмечалось в *предисловии*, описание проводится с использованием школьного алгоритмического языка (в ряде случаев приводятся также аналогичные фрагменты программ на языке Паскаль).

1.1. Поиск минимума и максимума двух, трех, четырех данных чисел без использования массивов и циклов[1]

1.1.1. Поиск максимума/минимума среди двух чисел (*a* и *b*)

Для двух чисел задачи решаются так, как показано на рис. 1.1 и 1.2.

Обращаем внимание на тот факт, что случаи равенства переменных можно не рассматривать, т. к. нам безразлично, какое из двух равных значений считать максимумом или минимумом.

Соответствующие фрагменты программ на школьном алгоритмическом языке имеют вид[2]:

---

[1] Формулировки всех задач приведены согласно [10].

[2] Фрагменты программ, связанные с описанием переменных величин, вводом исходных данных, выводом результатов (как правило) и т. п., приводить не будем.

```
если a > b
   то
      макс := a
   иначе
      макс := b
все
```

и

```
если a < b
   то
      мин := a
   иначе
      мин := b
все
```

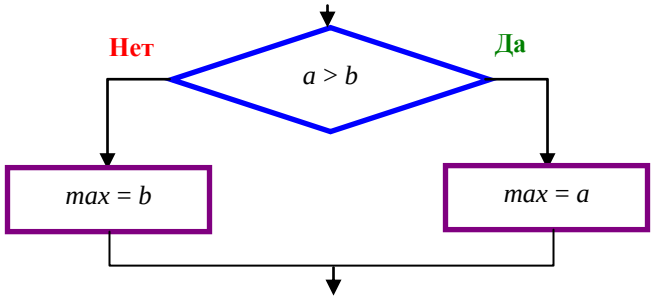

Рис. 1.1. Поиск максимума

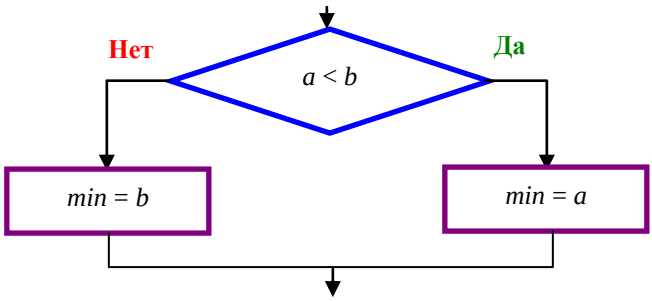

Рис. 1.2. Поиск минимума

Язык Паскаль

```
{Поиск максимума}
if a > b then max := a
  else max := b
```

```
{Поиск минимума}
if a < b then min := a
else min := b
```

Можно также использовать два неполных условных оператора:

1) при поиске максимума:

```
если a > b
  то
    макс := a
  все
если b > a
  то
    макс := b
все
```

2) при поиске минимума:

```
если a < b
  то
    мин := a
все
если b < a
  то
    мин := b
все
```

Язык Паскаль

```
{Поиск максимума}
if a > b then max := a;
if b < a then max := b;
{Поиск минимума}
if a < b then min := a;
if b < a then min := b;
```

1.1.2. Поиск максимума/минимума среди трех чисел (*a*, *b* и *c*)

Здесь также можно использовать три неполных условных оператора, каждый из которых соответствует отдельному варианту возможного ответа:

1) при поиске максимума:

```
если a > b и a > c
  то
    макс := a
все
если b > a и b > c
  то
    макс := b
```

```
все
если с > а и с > b
  то
    макс := с
все
```

2) при поиске минимума:

```
если а < b и а < с
  то
    мин := а
все
если b < а и b < с
  то
    мин := b
все
если с < а и с < b
  то
    мин := с
все
```

Язык Паскаль

```
{Поиск максимума}
if (a > b) and (a > c) then max := a;
if (b > a) and (b > c) then max := b;
if (c > a) and (c > b) then max := c;
{Поиск минимума}
if (a < b) and (a < c) then min := a;
if (b < a) and (b < c) then min := b;
if (c < a) and (c < b) then min := c;
```

Заметим, что попытка объединить любые два неполных условных оператора в один полный:

1) при поиске максимума:

```
если а > b и а > с
  то
    макс := а
все
если b > а и b > с
  то
    макс := b
  иначе
    макс := с
все
```

2) при поиске минимума:

```
если b < a и b < c
  то
    мин := b
все
если c < a и c < b
  то
    мин := c
  иначе
    мин := a
все
```

в ряде случаев дает неправильный результат. Например, результатом выполнения приведенных фрагментов в первом случае при `a = 5`, `b = 4`, `c = 3` будет значение величины `макс`, равное `3`, а во втором — при `a = 5`, `b = 4`, `c = 8` — значение величины `мин`, равное `5`.

Правильным вариантом решения является также такой — вместо трех неполных условных операторов применить один оператор для *поиска максимума*, построенный по следующему принципу:

1. Сначала сравниваются два любых числа, например *a* и *b*.
2. Если оказалось, что *a* больше, чем *b*, то именно число *a*, как возможный "кандидат" на статус максимума, мы сравниваем с третьим числом, *c*; если же, наоборот, число *b* больше, чем *a*, то именно *b* и сравнивается затем с числом *c*.
3. По результатам этого второго сравнения принимается окончательное решение о том, какое число из трех максимальное.

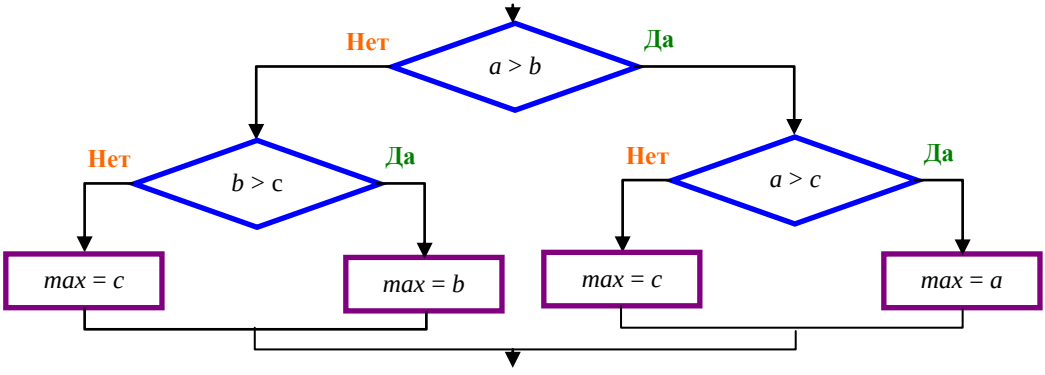

Рис. 1.3. Поиск максимума с помощью одного условного оператора

Эти рассуждения можно оформить в виде блок-схемы (рис. 1.3) и фрагмента программы:

```
если a > b
  то
    если a > c
```

```
        то
          макс := a
        иначе
          макс := c
      все
    иначе
      если b > c
        то
          макс := b
        иначе
          макс := c
      все
все
```

Заметим, что такую конструкцию называют *вложенный условный оператор*.

Язык Паскаль

```
if a > b then
  if a > c then max := a
  else max := c
else
  if b > c then max := b
  else max := c;
```

Аналогично, для *поиска минимума* также требуется конструкция из нескольких команд **если** (условных операторов), построенная по принципу:

1. Сначала сравниваются два любых числа, например *a* и *b*.
2. Если оказалось, что *a* меньше, чем *b*, то именно число *a*, как возможный "кандидат" на статус минимума, сравнивается с третьим числом, *c*; если же, наоборот, число *b* меньше, чем *a*, то именно *b* и сравнивается затем с числом *c*.
3. По результатам этого второго сравнения принимается окончательное решение о том, какое число из трех является минимальным.

Соответствующий фрагмент программы имеет вид:

```
если a < b
  то
    если a < c
      то
        мин := a
      иначе
        мин := c
    все
  иначе
    если b < c
      то
```

```
            мин := b
         иначе
            мин := c
      все
все
```

Язык Паскаль

```
if a < b then
   if a < c then
      min := a
   else
      min := c
else
   if b < c then
      min := b
   else
      min := c;
```

1.1.3. Поиск максимума/минимума среди четырех чисел (*a*, *b*, *c* и *d*)

Ситуация с четырьмя числами — наиболее сложная. Здесь при *поиске максимума* нужно вести сравнение (тоже с помощью цепочки многократно вложенных команд **если**) в следующем порядке:

- *a* сравнивается с *b*
- если *a* > *b*, то *a* сравнивается с *c*
  - если *a* > *c*, то *a* сравнивается с *d*
    - если *a* > *d*, то *макс* = *a*
      иначе *макс* = *d*
  - иначе *c* сравнивается с *d*
- если *c* > *d*, то *макс* = *c*

иначе *макс* = *d*

и т. д.

Приведем соответствующие фрагменты программ:

1) поиск максимума:

```
если a > b
  то
    если a > c
      то
        если a > d
          то
            макс := a
```

```
          иначе
            макс := d
        все
      иначе
        если c > d
          то
            макс := c
          иначе
            макс := d
        все
    все
  иначе |b > a
      если b > c
        то
          если b > d
            то
              макс := b
            иначе
              макс := d
          все
        иначе
          если c > d
            то
              макс := c
            иначе
              макс := d
          все
      все
все
```

Язык Паскаль

```
if a > b then
  if a > c then
    if a > d then
      max := a
    else
      max := d
  else
    if c > d then
      max := c
    else
      max := d
else {b > a}
  if b > c then
```

```
    if b > d then
      max := b
    else
      max := d
  else
    if c > d then
      max := c
    else
      max := d;
```

2) поиск минимума:

```
если a < b
  то
    если a < c
      то
        если a < d
          то
            мин := a
          иначе
            мин := d
        все
    иначе
      если c < d
        то
          мин := c
        иначе
          мин := d
      все
  иначе |b < a
    если b < c
      то
        если b < d
          то
            мин := b
          иначе
            мин := d
        все
      иначе
        если c < d
          то
            мин := c
          иначе
            мин := d
        все
    все
все
```

Язык Паскаль

```
if a < b then
  if a < c then
    if a < d then
      min := a
    else
      min := d
  else
    if c < d then
      min := c
    else
      min := d
else |b < a
  if b < c then
    if b < d then
      min := b
    else
      min := d
  else
    if c < d then
      min := c
    else
      min := d;
```

Полученные фрагменты программ — очень громоздкие и трудночитаемые. В них использованы по семь служебных слов **если** (**if**), **то** (**then**) и **иначе** (**else**)[1].

Можно упростить программу, если использовать *логические переменные*[2] [15].

Итак, пусть описаны переменные логического типа `x1`, `x2`, `x3`, `x4`, `x5`, `x6`.

Сначала мы присваиваем каждой такой переменной результат одного сравнения каждой возможной пары чисел:

```
x1 := a > b; x2 := a > c; x3 := a > d
x4 := b > c; x5 := b > d
x6 := c > d
```

[1] Чтение подобных "запутанных" фрагментов в определенной степени облегчает такое правило: "Каждое **else** (**иначе**) относится к тому ближайшему из предыдущих **if** (**если**), у которых нет «своего» **else** (**иначе**)". Проверьте это правило на приведенных фрагментах.

[2] Естественно, в средах программирования, в которых это возможно. В языках Бейсик (вариант QBasic) и Си можно использовать переменные числового типа.

Первые три логические переменные хранят результаты сравнения значения переменной a с каждой из трех остальных. Далее мы должны сравнить со всеми остальными значение переменной b, но вспомним, что с переменной a мы ее уже сравнивали и что требуемое сравнение b > a можно считать эквивалентным записи **не**(a > b). Поэтому достаточно записать сравнения переменной b только с переменными c и d. Аналогично переменную c нужно сравнивать только с переменной d.

Сформировав таким способом значения логических переменных, на их основе можно сформулировать логические условия, при которых та или иная из четырех имеющихся переменных является максимальной. Например, для переменной a:

```
если x1 и x2 и x3
  то
    макс := a
все
```

А когда максимальным является значение b? Это имеет место, когда b > a (т. е. значение переменной x1 — ложное) и при этом значение переменных x4 и x5 — истинное:

```
если не x1 и x4 и x5
  то
    макс := b
все
```

Рассуждая аналогично, можем записать:

```
если не x2 и не x4 и x6
  то
    макс := c
все
если не x3 и не x5 и не x6
  то
    макс := d
все
```

Язык Паскаль

```
x1 := a > b; x2 := a > c; x3 := a > d;
x4 := b > c; x5 := b > d; x6 := c > d;
if x1 and x2 and x3 then max := a;
if not x1 and x4 and x5 then max := b;
if not x2 and not x4 and x6 then max := c;
if not x3 and not x5 and not x6 then max := d;
```

В заключение заметим, что рассмотренные три задачи могут быть решены следующим образом.

Задача 1.1.1:

1) при поиске максимума:

```
макс := a
если b > макс
  то
    макс := b
все
```

2) при поиске минимума:

```
мин := a
если b < мин
  то
    мин := b
все
```

Задача 1.1.2:

1) при поиске максимума:

```
макс := a
если b > макс
  то
    макс := b
все
если c > макс
  то
    макс := c
все
```

2) при поиске минимума:

```
мин := a
если b < мин
  то
    ммн := b
все
если c < мин
  то
    мин := c
все
```

Задача 1.1.3:

1) при поиске максимума:

```
макс := a
если b > макс
  то
    макс := b
все
```

```
если c > макс
  то
    макс := c
все
если d > макс
  то
    макс := d
все
```

2) при поиске минимума:

```
мин := a
если b < мин
  то
    мин := b
все
если c < мин
  то
    мин := c
все
если d < мин
  то
    мин := d
все
```

Правда, здесь возникает вопрос: можно ли считать, что при их решении не использованы циклы? (Операторы цикла не применялись, но есть повторяющиеся, точнее — аналогичные, действия.)

1.2. Нахождение всех корней заданного квадратного уравнения

Если коэффициент a квадратного уравнения считать не равным нулю, то задача не должна представлять трудностей для решения.

Квадратное уравнение с коэффициентами a, b, c при $a \neq 0$ может иметь от 0 до 2 корней в зависимости от значения дискриминанта $d = b^2 - 4ac$:

- при $d > 0$ корней — два, и они вычисляются по формулам:

$$x1 = \frac{-b + \sqrt{d}}{2a};$$

$$x2 = \frac{-b - \sqrt{d}}{2a};$$

- при $d = 0$ корень — один:

$$x1 = \frac{-b}{2a};$$

- при $d < 0$ корней нет.

Так как возможны три варианта ответа, то, как и при решении *задачи 1.1.2*, в программе можно использовать три неполных условных оператора:

```
если d > 0
  то
    x1 := (-b + sqrt(d))/(2 * a)
    x2 := (-b - sqrt(d))/(2 * a)
    вывод нс, "Уравнение имеет два корня: "
    вывод "x1=", x1, " x2=", x2
все
если d = 0
  то
    x1 := (-b + sqrt(d))/(2 * a)
    вывод нс, "Уравнение имеет один корень: "
    вывод "x1=", x1
все
если d < 0
  то
    вывод нс, "Уравнение не имеет корней"
все
```

Язык Паскаль

```
if d > 0 then
  begin
    x1 := (-b + sqrt(d))/(2 * a);
    x2 := (-b - sqrt(d))/(2 * a);
    write('Уравнение имеет два корня: x1=', x1:7:2, ' x2=', x2:7:2)
  end;
if d = 0 then
  begin
    x1 := (-b + sqrt(d))/(2 * a);
    write('Уравнение имеет один корень: x1=', x1:7:2)
  end;
if d < 0 then
  write('Уравнение не имеет корней');
```

Здесь, как и при решении *задачи 1.1.2*, объединять любые два неполных условных оператора в один полный нельзя.

Можно вместо трех неполных условных операторов применить один вложенный условный оператор:

```
если d > 0
  то
    x1 := (-b + sqrt(d))/(2 * a)
    x2 := (-b - sqrt(d))/(2 * a)
    вывод нс, "Уравнение имеет два корня "
    вывод "x1=", x1, " x2=", x2
```

```
  иначе
    если d = 0
      то
        x1 := (-b + sqrt(d))/(2 * a)
        вывод нс, "Уравнение имеет один корень "
        вывод "x1=", x1
      иначе
        вывод нс, "Уравнение не имеет корней"
    все
все
```

Язык Паскаль

```
if d > 0 then
  begin
    x1 := (-b + sqrt(d))/(2 * a);
    x2 := (-b - sqrt(d))/(2 * a);
    write('Уравнение имеет два корня: 'x1=', x1:7:2, ' x2=', x2:7:2)
  end
else
  if d = 0 then
    begin
      x1 := (-b + sqrt(d))/(2 * a);
      write('Уравнение имеет один корень: x1=', x1:7:2)
    end
  else
    write('Уравнение не имеет корней');
```

Обратим внимание на то, что распространенной ошибкой при разработке программ решения обсуждаемой задачи является отсутствие скобок в знаменателе выражения для расчета корней.

1.3. Нахождение наибольшего общего делителя двух натуральных чисел (алгоритм Евклида)

Древнегреческий математик Евклид в своей знаменитой работе "Начала" (330–320 гг. до нашей эры) изложил алгоритм нахождения наибольшего общего делителя (НОД) двух натуральных чисел. Суть этого (считающегося самым древним) алгоритма состоит в следующем. Если число a больше числа b, то нужно от a отнять b, в противном случае, наоборот, от b отнять a и повторять эти действия до тех пор, когда a станет равно b. После этого искомый НОД будет равен одному из полученных чисел. Сказанное иллюстрирует следующая схема:

$$\text{НОД}(a, b) = \begin{cases} \text{НОД}(a - b, b), \text{ если } a > b \\ \text{НОД}(a, b - a), \text{ если } b < a \\ a \text{ (или } b\text{)}, \text{ если } a = b \end{cases}$$

Например, при $a = 26$, $b = 18$ имеем: НОД(26, 18) = НОД(8, 18) = НОД(8, 10) = НОД(8, 2) = НОД(6, 2) = НОД(4, 2) = НОД(2, 2) = 2.

Приведем фрагмент программы, работающий по этому алгоритму:

```
нц пока a <> b
  если a > b
    то
      a := a − b
    иначе
      b := b − a
  все
кц
НОД := a |Или НОД := b
```

Язык Паскаль

```
while a <> b do
  if a > b then a := a − b
    else b := b − a;
NOD := a {или NOD := b}
```

Как можно улучшить эту программу? Прежде всего, можно многократные вычитания заменить на определение остатка от деления одного целого числа на другое. Эти действия должны повторяться, пока ни одно из чисел не равно нулю:

```
нц пока a <> 0 и b <> 0
  если a > b
   то
     a := mod(a, b)
   иначе
     b := mod(b, a)
  все
кц
```

где `mod` — функция, возвращающая остаток от деления своего первого аргумента на второй. В других языках программирования для этого используется не функция, а специальная операция (как правило, знак этой операции также обозначается `mod`).

После окончания работы оператора цикла искомое значение НОД может быть определено так:

```
если a = 0
 то
  НОД := b
 иначе
  НОД := a
все
```

или, короче (и эффектно!), так:

```
НОД := a + b
```

Язык Паскаль

```
while (a <> 0) and (b <> 0) do
  if a > b
    then
      a := a mod b
    else
      b := b mod a;
NOD := a + b
```

Можно также применить оператор цикла с постусловием:

```
нц
  если a > b
   то
     a := mod(a, b)
   иначе
     b := mod(b, a)
  все
кц_при a = 0 или b = 0
НОД := a + b
```

Язык Паскаль

```
repeat
  if a > b
    then
      a := a mod b
    else
      b := b mod a
until (a = 0) or (b = 0);
NOD := a + b;
```

Дальнейшее усовершенствование: можно отказаться от многократного сравнения величин `a` и `b` — достаточно сделать это один раз, а потом учесть, что получаемый в цикле остаток всегда будет меньше, чем второй параметр функции `mod`. В приведенной далее программе, учитывающей это обстоятельство, использованы следующие новые величины:

- `макс` — максимальное из двух заданных чисел;
- `мин` — то же, минимальное;
- `остаток` — остаток от деления `макс` на `мин`.

```
|Определяем значения макс и мин
если a > b
 то
   макс := a
```

```
   мин := b
 иначе
   макс := b
   мин := a
все
остаток := mod(макс, мин)
нц пока остаток <> 0
  макс := мин
  мин := остаток
  остаток := mod(макс, мин)
кц
НОД := мин
```

Язык Паскаль

```
{Определяем значения max и min}
if a > b then
  begin
    max := a;
    min := b
  end
else
  begin
    max := b;
    min := a
  end;
ostatok := max mod min;
while ostatok <> 0 do
  begin
    max := min;
    min := ostatok;
    ostatok = max mod min
  end
NOD := min
```

И, наконец, самый короткий вариант решения задачи:

```
нц пока b <> 0
  остаток := mod(a, b)
  a := b
  b := остаток
кц
НОД := a
```

В его особенностях разберитесь самостоятельно. В частности, проанализируйте, правильно ли будет работать программа при $b > a$ и при $a = b$.