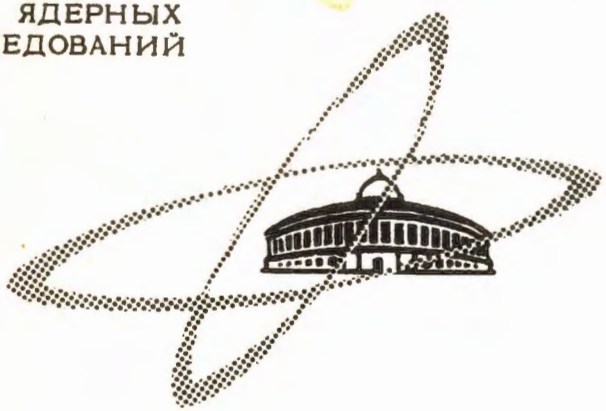
ОБЪЕДИНЕННЫЙ
ИНСТИТУТ
ЯДЕРНЫХ
ИССЛЕДОВАНИЙ

Дубна



R - 2610

Б. Хигмен

ЧТО ДОЛЖЕН ЗНАТЬ КАЖДЫЙ ОБ АЛГОЛЕ

(The Computer Journal, v. 6, 1963)

Перевод И.Н. Силина,
В.П. Ширикова

ВЫЧИСЛИТЕЛЬНЫЙ ЦЕНТР

1966

R - 2610

Б. Хигмен

ЧТО ДОЛЖЕН ЗНАТЬ КАЖДЫЙ ОБ АЛГОЛЕ

(The Computer Journal, v. 6, 1963)

Перевод И.Н. Силина,
В.П. Ширикова

Часть I. "0"-уровень

if вы никогда не пользовались математикой

then дальше не читайте

otherwise if вы не хотите пользоваться АЛГОЛОМ

then уходите на пенсию

otherwise прочтите внимательно эту статью[и].

Это был хороший и правильный пример оператора АЛГОЛА. Ортодоксы предпочитают более короткое **else** более длинному **otherwise** , но эти подчеркнутые (или выделенные жирным шрифтом) слова - это только фокус, с помощью которого обходят малочисленность символов на машинке, и спорить об этом - это все равно, что спорить, печатать ли "a раз по b " или " a ,умноженное на b ", если на машинке нет знака умножения. И то и другое-неуклюжий заменитель креста × .

АЛГОЛ возник потому, что обычных математических обозначений недостаточно для описания процесса вычислений и нужно их расширение. АЛГОЛ есть не что иное как символика, требуемая для описания вычислений, и правила - как ею пользоваться. Приписываемые ему трудности в значительной мере преувеличены. Все мы знаем, что $a+b\times c+d$ таит в себе ловушку, заключающуюся в следующем: $b \times c$ нужно вычислить сначала. Так же и в АЛГОЛЕ есть много ловушек, в которые вы можете попасть, если не знаете правил; они не сложнее и не многочисленнее тех, что связаны с алгеброй.

Каждый оператор АЛГОЛА производит действия. Это совсем не похоже на другие математические выражения, такие как $(a+b)(a-b)= a^2 - b^2$, которые просто сообщают вам нечто[ии].

Имеются три правила пользования операторами, начинающимися с **if** :

(1) Общий вид :

if < условие > **then** < действие, если условие верно >

else < действие, если условие неверно > .

(2) Если второе действие есть "ничего не делай", то оно и **else** могут быть полностью опущены.

(3) Второе действие, но не первое, может быть, в свою очередь, **if**- оператором.

и) Перевод всех английских слов см. в конце статьи.

ии) Но в АЛГОЛЕ достаточно духа чистой математики, которая рассматривает "ничего не делай" единственным возможным действием. И не будьте в дальнейшем введены в заблуждение: "ничего не делай" не означает "остановись". Это больше похоже на "на этот раз пропусти".

Третье правило спасает от путаницы: какому if принадлежит какое else ? И мы увидим позже, что его можно в некотором смысле обойти, если поставить скобки вокруг первого действия.

Заметим, что

if ax + b = 0 then x = - b/a -

неверный алгольный оператор, потому что за then следует условие, а не действие.

Действие может быть построено из группы простых действий, которой присваивается наименование. Вот почему наше предложение в начале статьи может быть правильным оператором; если оно встретится в программе, то три действия, его составляющие, будут предварительно детально определены.

99 % простых действий - одного из двух сортов:

присвоение значения переменной и

переходы, передачи управления (например: вернуться к первому оператору).

Последовательные простые действия отделяются точками с запятой. Точки с запятой отделяют также определения (которые не определяют действий).

Далее (странное дело) математическое обозначение само по себе не является таким недвусмысленным, как хотелось бы.

Пусть l , m , n - три числа. Рассмотрим

$$l n\,(m^2 + n + 1)\,,$$

Вы знаете, что это: $l \times n \times (m^2 + n + 1)$ - или это натуральный логарифм от $m^2 + n + 1$? Ясно, что заставлять машину разбираться в этом из контекста, как вы, может быть, хотели бы, - это значит испытывать судьбу. АЛГОЛ замышляется быть понятным для машины так же, как и для человека. Поэтому, чтобы достичь этого раз и навсегда, вводится правило о наименованиях. Любая последовательность букв или букв и цифр, начинающаяся с буквы, рассматривается как одно наименование (идентификатор). Далее, пропуски, возвраты каретки и пр. полностью игнорируется. Так, Black Smith и BlackSmith считаются за одно наименование. (Но большие и малые буквы считаются разными, если вы этого хотите). Это разрешает спор о "ln" в пользу натурального логарифма. Это также означает, что мы должны внимательно писать

не 2 ab , а 2 × a × b ;

не cos X , а cos (X) .

Но в нашем распоряжении широкий выбор наименований, сколько бы мы их ни пожелали:

A, B, C ;

alpha, beta, gamma ;

aleph, beth, gimel ;

co45, y vIa, Minor I000 ;

Temperaturpotenzreichsentwicklung .

Каждая свобода, однако, относительна; и в этом случае до того, как мы позволим себе использовать какое-то наименование, мы должны определить его. (Исключение может быть сделано для всем известных $\sin$, $\cos$ и т.п.). Сейчас мы можем рассмотреть три способа такого определения:

(I) перечисляя наименования, которые присвоены действительным или целым переменным, в списке о соответствующим заголовком;

(2) присваивая наименования (метки) операторам;

(3) давая подробности любого сложного действия, которому мы хотим дать наименование.

АЛГОЛ добавляет повелительное наклонение к алгебре. В алгебре мы отмечаем, что x равно y , написав x= y . В АЛГОЛЕ мы нуждаемся в символическом изображении того, чтобы x сделать равным y , и для этого выбрано

x : = y ;

двоеточие и равенство вместе образуют единый символ (как th эквивалентно θ). Еще один приказ требуется для переходов - это символ go to .

Теперь мы можем написать программу ! По крайней мере, если мы позволим себе одну вольность, то мы можем. Следующая программа будет печатать числа I, 4, 9, I6, 25, ...

integer a; a: = o; R : a: = a + I; print (a × a); go to R.

Это тривиальная программа, конечно, но она демонстрирует первые два вида определений (описаний) и оба сорта действий. a - переменная, определенная списком, и R - метка, определенная приписыванием к выражению через двоеточие. (Вольность в предположении, что print не должно определяться. На самом деле любое ее детальное определение будет включать не только АЛГОЛ, но и некоторый машинный код. Это касается любого ввода и вывода, и АЛГОЛ, не указывая деталей, позволяет определению сложного действия в таких случаях быть в машинном коде).

Решение квадратного уравнения есть "0" уровень алгебры. Можно с уверенностью оказать, что любой дочитавший до этого места не затруднится сделать это. Но нужно много мерзкой работы, чтобы запрограммировать счет множества специальных случаев и гарантировать полную точность. Это уже будет хорошим примером "реальной" программы после того, как мы рассмотрим еще несколько пунктов.

(I) Предположим, мы захотели написать

```
if  a = 0   then  p: = 0;     q : = I
    else  p:=I;     q:=2.
```

Когда дело дойдет до интерпретации этого, то ничего не выйдет, ибо как только встретится первая точка с запятой, будет предполагаться, что это конец "короткого" условного (if) оператора (в котором нет **else** , потому что второе действие есть "ничего не делай"). Нам, значит, нужен некоторый вид скобок, чтобы заключить в них "p:=0; q:=I" , так,чтобы было

```
if a=0 then { p:=0; q:=0 } ;
       else { p:=I; q:=2 } ;
```

и им следовало бы для удобства отличаться от круглых скобок, используемых в алгебраических выражениях. В АЛГОЛЕ используют **begin** и **end** для этих скобок и пишут:

```
if a = 0 then  begin  p:=0; q:=0  end
                      else  begin  p:=I; q:=2  end;
```

(2) Эти скобки используются и в определении сложного действия. Чтобы образовать такое определение, мы выписываем детали, окружая их такими скобками, и перед результатом пишем

"**procedure** < наименование > " .

Хоть это и не вся еще история, но это суть определения наименования сложного действия. Такие обычно всю программу окружают

begin ... **end**.

(3) Определения должны следовать сразу за **begin**. Если это определение в начале программы, то наименования определяются для **всей** программы; но,ставя их после некоторого другого **begin** , мы можем локализовать его для части программы.

(4) Получается так, что только три символа могут легально следовать за **end** , а именно,другое **end** , точка с запятой или **else**.Правила позволяют включать после **end** комментарии, которые не содержат ни одного из этих трех символов. Это, в частности, полезно: можно писать кое-что после **end**, а читателю легче разобраться, какое **end** какому **begin** соответствует. Такие комментарии машиной игнорируются.

(5) Наконец, нам нужно, по крайней мере, еще две возможности для ввода-вывода:

next - сокращение "следующее число на вводимой ленте" и **printtext** (X)

printtext(X)- печать, но не значения X , а самой буквы X . И еще: **sign** (X) , так как он понимается не всеми одинаково, будем понимать так, что он есть +I для X>0, -I для X<0 и 0 для X =0 .

Здесь, дальше, приводится программа приема трех коэффициентов квадратного уравнения (первый - при x^2) и печати его корней. Использование обычной формулы для численного определения наибольшего корня и следующее за тем использование знания произведения корней для получения наименьшего - есть хорошо известная техника избежания потерь значащих цифр.

```
begin real a,b,c;
        a = next; b = next; c = next;
if a=0 then begin
        if b=0 then begin
if c=0 then printtext                    ('не определены')
        else printtext                   ('оба бесконечны')
        go to            пятое колесо    end b=0
        else print (-c/b): printtext     ('и бесконечный')
        end a=0
        else begin real d;
        d = b×b -4 ×a × c;
        if d<0 then go to complex;
        d:= sqrt (d);
        if b≠0 then d = -b- d× sign (b);
        print (d /(2 × a ) );
        if c=0 then print (0) else print (2 × c/d ) ;

    go to     пятое колесо;
        Complex : print ( -b/ (2×a) );
        printtext ('+ ␣ and ␣ - ␣ i ␣ x')
        print (sqrt (-d) / (2×a) ) end a≠0;
```

пятое колесо: end программы ...

Здесь d - промежуточная переменная.

Заметим, что действие, помеченное "пятое колесо", есть "ничего не делай", чтобы перескочить, так сказать, в самую середину конца программы. Название выбрано, чтобы подчеркнуть факт, что программа закончена раньше, чем достигнуто это выражение. d - вспомогательная переменная, определенная только для второй части проблемы. Так как пробелы игнорируются, printtext имеет специальное обозначение для пробела: ␣

Часть II. "А" уровень

Можете ли вы прочитать следующие формулы вслух так, чтобы другой человек мог их записать и правильность результата была бы очевидной?

(1) $e^{p+q} + 2$,

(2) $F_G f_g + F_g f_G + F_{G-g} f_{G+g} + F_{G+g} f_{G-g}$,

(3) $x^{2}_{m_1, m_2^2} + x^{-2}_{m_1^2, m_2}$.

Современная математика – это, скорее, игра руки и глаза, каждый из которых оперирует двухмерно, чем рта и уха, которые ограничены одной размерностью (если вы не станете повышать и понижать тон вашего голоса для показателей степени и индексов; но даже тогда пример 3 был бы насилием над музыкой).

Бумажная лента в этом смысле одномерна. Поэтому АЛГОЛ неизбежно имеет что-то от неуклюжести разговорной математики в связи с расстановкой показателей и индексов. На языке АЛГОЛА наши формулы примут вид:

(1) `e↑(p+q)+2`,

(2) `F[G]×f[g]+F[g]×f[G]+F[G-g]×f[G+g]+F[G+g]×f[G-g]`,

(3) `x[m[1],m[2]↑2]↑2+x[m[1]↑2,m[2]]↑(-2)`.

Направленная вверх стрелка поднимает следующее за ней число, наименование или заключенное в скобки выражение на уровень показателя степени. Число должно быть без знака или, в противном случае, заключено в скобки. Функциональное наименование, как, например, `Sin`, обязательно требует последующих скобок, содержащих аргумент. Квадратные скобки опускают их содержимое на уровень индекса и могут вставляться друг в друга так часто, как этого требуют обстоятельства.

На бумажной ленте с менее чем семью отверстиями будут еще сдвигающие символы, соответствующие таким, например, словам, как "заглавная" и "маленькая", которые употребляются в разговоре. В случае, когда программа не предназначена для непосредственного ввода в машину, можно, если хочется, применять более обычные обозначения.

Переменные, снабженные индексами, определяются перечислением их в списке с указанием информации о границах индексов, например,

`array X,Y[1:3],Z[0:2],A,B[1:3,1:3];`

определяют `X`, `Y` и `Z` как трехкомпонентные вектора (`Z` имеет отличные от `X` и `Y` границы изменения индекса), а A и B – как матрицы 3x3.

Так поступают только с алгебраическими индексами, т.е. индексами, которые сами есть переменные или выражения. Если y_i и y_o означают значения y при вводе и выводе (про-

межуточные значения), то в АЛГОЛЕ они будут просто yi и $y0$. Если i случится быть равным 3 в какой-то момент, то $y[i]$ будет $y[3]$, но yi не будет $y3$.

Мы можем испечь гораздо худшие формулы, чем вышеприведенные (I)-(3)! Как вам нравится детерминант компоненты которого определенные интегралы, пределы которых сами определенные интегралы? К счастью, подобного сорта вещи всегда могут быть укатаны с помощью функциональных обозначений. Например, можно написать формулу

$$\sum_{k=a}^{b} f(k) \qquad \text{как} \quad \text{sigma}\,(f(k), k, a, b)\,.$$

Это делается даже слишком просто. А выгода? Только та, что мы теперь должны представить определение для $Sigma$. Но этого самого по себе достаточно, чтобы потребовался новый раздел.

Конечно, нам не нужно вырабатывать заново определение $Sigma$, если мы можем найти его в библиотеке. Библиотека, по-видимому, даст нам что-то подобное:

```
real procedure sigma (w,x,y,z); value y,z, real w;
integer x,y,z;
begin real s; s: =0;
for x:=y step 1 until z do s:=s+w;
sigma :=s end .
```

Это не так плохо, как выглядит, если мы рассмотрим основные черты по очереди.

(I) Это более сложная форма структуры

procedure < наименование > begin... end,

которую мы упоминали в части I.

(2) Слово real означает, что это не столько "сложное действие", сколько функция с вещественным значением, требующая для его вычисления "сложного действия". Функция получает значение в последнем операторе

sig ma : =s .

(3) Когда программа встречает sigma (f (h),h,a,b,), она выполняет действия, описанные в определении с заменой w на f(h), x на h, y на a и z на b.

(4) Процесс может отказать, если величины, заменяющие x, y и z, не целые (integer), или величина, заменяющая w, не вещественная (real)(что в данном контексте включает и целую).

(5) Список, озаглавленный символом value, означает, что величины, замещающие y и z, определяются один раз перед началом выполнения процедуры. Напротив, замещение величин w и x делается алгебраически, так что каждый раз, когда в определении процедуры

встречается x , выполняется f(h) для текущего значения h .

(6) Новые символы for, step, until, do могут пониматься в их прямом значении. Так же, как и у then , влияние do кончается у первой точки с запятой, не защищенной операторными скобками begin ... end.

Это должно быть кристально ясно! Если нет, попытайтесь прочитать еще раз (процесс, называемый итерацией). Хотя, так как приведенные комментарии можно расширить для охвата более разнообразных контекстов, читать дальше тоже полезно. Например, есть другие типы операторов, начинающиеся с for. Так, чтобы вычислить

$$\sum_{x = 1,2,3} w[x] \quad ,$$

мы можем написать

```
s:=0; for x:=1,2,3 do s:=s+w[x] ;
```

а чтобы вычислить

$$\sum_{x = 1 \ldots 2k} w \quad ,$$

исключая $k \pm 1$,

мы можем написать

```
s:=0;
for x:=1 step 1 until k-2, k,k+2 step 1 until 2×k do   s:= S+W ;
```

и есть еще третья возможность в применении, скажем, к случаю

$$\sum_{n=1}^{\infty} \frac{1}{n^p}$$ до 9-й значащей цифры,

а именно:

```
s:=0; for n:=1,  n+1 while n^p<10^10 do
                    s:=s +n↑( -p) ;
```

Эффект списка, озаглавленного value , весьма тонкий, но может быть в общих чертах описан в трех пунктах. (I) Процедура делает свою собственную копию каждой величины в этом списке, чтобы делать с ней что ей нравится, и таким образом защищает экземпляры этих переменных, которые являются собственностью основной программы. Здесь она похожа на человека, который делает свою собственную копию кроссворда, чтобы, решая его, не испортить оригинал для других членов семьи. (2) Если процедура имеет дело с чем-то вроде

```
sigma ( k↑2, k,1, p↑2 - p×q + q↑2),
```

тогда, если бы z не было в списке значений (value), процедура обязана была бы пересчи-

тывать p^2-pq+q^2 каждый раз, когда с ним сравнивается x на всякий случай, если бы p или q могли за это время как-то измениться, что было бы большой потерей времени. Но (3) непреложный факт, что w не включено в список значений и оно разное каждый раз, когда вычисляется, это-то вообще и позволяет процедуре работать.

На этом пути ничто не исключается, что соответствует стандартным функциональным обозначениям. Так,

$$\sum_{j=1}^{n}\sum_{k=1}^{n} f(j,k)$$

становится

```
sigma ( sigma (f(j,k)k,1,n),j,1,m) ж).
```

С интегралами можно поступать в том же духе, с той оговоркой, что в этом случае определение, поскольку оно предусматривает детали, будет предполагать конкретный выбор квадратурных правил (например, Симпсона). Слово предостережения необходимо относительно фиктивных параметров в подобных выражениях: j и k в предыдущем примере. Они должны быть определены в основной программе, но они использованы процедурой. И это делает рискованным использование их в основной программе, за исключением, может быть, весьма временных надобностей. Если это предостережение забыто, процедура может подложить свинью, тайком меняя числа за спиной основной программы.

Часть III. " S " уровень

S - это из-за strings (строк), side effects (побочных эффектов), switches (переключателей), synonyms (синонимов) и several other things (некоторых других вещей).

Строки мы встречали уже ранее, не называя их так. Строка есть нечто в перевернутых запятых. Две программы вывода, которые мы использовали при решении квадратного уравнения, могли бы иметь определения, начинающиеся так:

```
procedure print (x), real x ...
```

и

```
procedure printtext (x); string x ; ...
```

Алгол включает в себя строки по очевидным причинам, но это ему явно не по душе.

Побочный эффект мы также уже встречали, не определяя его, и так как враг пытался сделать из него пугало, то это пугало нужно повалить.

ж) Прим.перевод. Вот яркий, хотя и простой пример так называемого рекурсивного обращения к процедуре. Это напоминает надевание носков, не снимая ботинок, и не всякий транслятор умеет это делать.

Тот факт, что sigma(w,x,y,k) получает в свое собственное употребление какую-то переменную из основной программы, подставляемую вместо x , и таким образом вменивается в любое долговременное использование этой переменной, есть пример побочного эффекта. Пока это довольно безвредно, потому что затронутая переменная наблюдается среди параметров, когда вызывается sigma . Но предположим теперь, что нам любопытно знать, как часто на самом деле вызывается sigma в некоторой программе. Тогда мы добавим к программе:

(I) в ее исходных описаниях еще одно целое, count

(2) среди ее начальных приказов добавим count :=0

(3) внутри sigma добавим count := count +1,

(4) в конце программы: print (count)

А почему бы нет? Мы теперь имеем побочный эффект, который полностью скрыт, когда мы обращаемся к sigma , но основная идея была оделать его походя.

Ясно, что это скромное, но многообещающее начало и что мы можем теперь ввести самые ужасные побочные эффекты, черт бы их побрал (см.ссылку (3))! Но самое главное в сути дела:

(I) что побочные эффекты в принципе необходимы и

(2) программист, который настолько безответственен, что вводит побочные эффекты без необходимости, скоро потеряет доверие своих коллег, и совершенно справедливо.

Переключатели - это суть устройства для многоканального переключения. Предположим, что мы имеем большую программу, в которой нужно в разных местах решать квадратное уравнение. Тогда нашу программу, приведенную выше, следует превратить в процедуру. Наша реакция на разные трудные случаи может быть различной в каждом отдельном обращении к ней. В этих обстоятельствах мы могли бы использовать переключатель следующим способом: мы определим *solve* так:

```
Procedure solve (a,b,c,x1,x2,s); value a,b,c;
real a,b,c,x1,x2 ; switch s;
begin integer n;
if a=0 then begin
if b=0 then begin
if c=0 then n:=1 else n:=2;

go to ключ   end
else x1: =-c/b; n:=3 end
else begin real d; d:=b×b -4 ×a× c;
if d<0 then go to complex ;
n:=4; d := sqrt (d);
```

```
        if b ≠ 0 then d:=-b -d × sign (b);
            x1:=d /(2×a);


Сюрприз: x2 :=if o= 0 then 0 else 2 x (c/d);
         go to    ключ ;

         Complex : n:=5; x1:= -b /(2×a);
                   x2:=sqrt (-d) /(2×a) end;
         ключ  go to S [n]  end.
```

Когда мы захотим воспользоваться этой процедурой, мы напишем нечто вроде:

```
solve (p,q,r,s,t,A);
```

и затем должны включить в определения определение переключателя A , которое мы сделаем в виде:

```
swith A: =alpha, betta, gamma, delta, epsilon;
```

где alpha ... epsilon пять меток (не обязательно все разные и не обязательно простые), показывающих, куда мы хотим вернуться в основную программу в различных удовлетворительных или неудовлетворительных ситуациях, представленных n = 1,2,3,4,5.

Среди некоторых (several) других вещей мы можем упомянуть, во-первых, что конструкция if...then..else может быть использована не только для "условных действий", но также и для получения "условных значений". Например, посмотрите на действие, помеченное как "сюрприз" в только что выписанной процедуре, и сравните с соответствующей строкой в программе "0"-уровня. Метки после go to или в переключателе могут также иметь такой вид.

Во-вторых, утверждение, заключенное между if и then, имеет в качестве своего "значения" true или false (истина или ложь). Если мы хотим дать наименование такому утверждению, то это наименование определяется как логическая или булевская переменная. Такие значения и переменные влекут за собой использование логических функций, таких как and (пишется также & или ∧) и or (пишется также ∨). Покажем, как они используются:

```
        boolean x,y;
x:=  программа плохая;
y:=  время спать;
if x∨y then          выключайте телевизор;
```

∧ и ∨ между двумя булевскими переменными струкутрно подобны "+" и "-" между двумя численными величинами; кроме того, ∧ больше похож на × , поскольку он имеет приоритет перед ∨ . Подобным образом можно написать, скажем,

```
первый вход:=true    ; так же как и можно написать x:=25
```

S - это и для синонимов. Мы надеялись, что их использование оправдано тем, что мы избежали технических терминов. Тем не менее, хорошо иметь что-то вроде словаря, поэтому приведем список использованных нами слов, которые имеют технический эквивалент.

наименование - читай: идентификатор ;

действие - оператор ;

определение - описание (исключая те, которые определяют формальные параметры процедур, когда используется "спецификация") ;

сложное действие - процедура ("идентификатор" или "описание", смотря по контексту);

утверждение - булевское выражение ;

алгебраическая замена - вызов по наименованию (не по "значению").

Далее, begin ..end называется "составной оператор" , если никаких определений за begin не следует, или "блок", если такие определения есть. Различие - важное из-за правила, что наименования должны быть определены до того, как они используются, которое означает, что в блок следует входить через его begin , в то время как нет никаких возражений против прыжков к метке в середине (сложного) составного оператора.

Часть IУ. Ответы на некоторые вопросы

В. Сколько есть еще этих подчеркнутых слов?

О. Только три: comment, label и own . Первое слишком очевидно, чтобы его пононять, второе используется только как заголовок описка, а частое применение третьего настолько мало правдоподобно, что его можно спокойно забыть. Есть еще несколько математических и логических знаков, таких как ¬ (или not) и ÷ (для целого деления), которые мы не упомянули.

В. Использование английских слов таким образом принято международно?

О. Да. Но это не очень сильно подчеркивается, поскольку это не слова, а символы. Немцу не столько нужно выучить, что if это wenn , сколько, что за ним следует где-то then , так же как за знаком интеграла должно следовать где-нибудь d( ). И наоборот, для машины принятие wenn вместо if подобно принятию нами эквивалентности рукописного и печатного написаний одной и той же буквы и для чисто местных целей было бы тривиальным вариантом.

В. За АЛГОЛом -58 последовал АЛГОЛ-60. Как долго мы можем,доверять АЛГОЛу -60?

В. Достаточно ли хорош АЛГОЛ - не слишком ли рано замораживать его форму?

О. Эти вопросы дополняют друг друга. АЛГОЛ-58 был пробным, но АЛГОЛ-60 - совсем наоборот. Ответ на эти два вопроса: настолько долго, насколько можно предвидеть. АЛГОЛ-60, видимо, будет расширяться, а не изменяться (возможные добавления - разрешение определять переменные как complex и какая-нибудь техника для действительного манипулирования со строками).

В. Не пройдет ли много времени, прежде чем программы на АЛГОЛе будут приниматься для прямого ввода в машину?

О. Алгольные программы уже принимаются в математических центрах в Амстердаме, Копенгагене, Стокгольме и Майнце для прямого ввода, и две английские фирмы имеют алгольное оборудование в стадии пробной эксплуатации для своих наиболее современных машин (English Electric KDF9 и Elliot 503 или 803 на 8000 слов). В этой связи стоит заметить, что фразы вроде "КДФ9 АЛГОЛ" относятся к соглашениям, используемым при вводе, к методу обращения с библиотечными процедурами и т.д. (АЛГОЛ оставляет это открытым). Здесь нет никаких фундаментальных отклонений от АЛГОЛа.

В. Есть ли у АЛГОЛА реальные преимущества перед, скажем, автокодом Меркурия?

О. Однажды физик-ядерщик захотел знать значение

$$\int_0^1 (\ln u)^{-6} \int_0^{e^{-k/(-\ln u)^{1/2}}} (-\ln w)^{-n}\, dw\, du$$

для определенных целых значений n и k . Он не имел опыта программирования на машине в обычном смысле, но научился пользоваться АЛГОЛом. Он знал также достаточно, чтобы избегать просить машину вычислить $\ln(0)$ и потому приложил определенное усилие мысли, чтобы обойти проблему, поставленную его пределами интегрирования. Он нашел библиотечную процедуру для вычисления интеграла по Симпсону и, переписав, включил в свои определения, замечая, что в дополнение к очевидным требованиям она требует от него также задать требуемую точность, что ему показалось резонным, а также любопытное число V , которое, как ему показалось, машина отлично могла бы поставить сама. Он также заметил, что библиотечная процедура использовала некоторые такие же символы, которые он предполагал использовать, но это его не обеспокоило, потому что он знал, что концепция "локальных определений" АЛГОЛа позаботится об этом автоматически. Тогда он укомплектовал программу в форме, данной ниже, вручил операторам вместе со списком значений k и n и очень скоро получил требуемые результаты.

```
begin real u,w; integer k,n;
real procedure simps(F,x,a,b,delta, v);
value a,b, delta, v; real F,x,a,b, delta , v;
```

comment интегрирует F по x от a до b по Симпсону, деля пополам шаг до тех пор, пока относительная разность между двумя последовательными приближениями не станет меньше delta. V должно превышать утроенное максимальное значение abs(F) в интервале;

```
begin integer n,k; real h,J,I;
v:=(b-a)×v; n:=1; h:= (b-a) /2;
x:=a; J:=F; x:=b; J:=(J+F) ×h;
J1: b : =0;
for k:=1 step 1 until n do begin
x:=(2×k-1)×h+a;b:=b +F
end;
I:=4×h×b+J;
if abs (I-v)> abo (v)×delta then begin
v:=I; J:=(I+J)/4; n:=2×n;
h:=h/2; go to J1 end;
simps:= I/3 end simps;
actual program: n:= next; k:= next;
print (simps (if 10^-7>u or u> 0.999999 then 0 else (ln(u)↑(-6) ×
simps (if 10^-7>w then 0 else (-ln (w)↑(-n) ,
w,0,exp (-k/sqrt (-ln (u) ) ) 0.001, 10^35),
u,0,1,0.001, 10^35 ));
go to actual programm end.
```

Эта история в основном правдивая. Я получил ее из вторых рук и не могу поручиться за второстепенные детали. Например, как перебирались различные значения n и k , но основной print -оператор таков, каким я его получил.

Я вызываю любого поклонника автокода произвести что-либо сравнимое в прямоте отношения к оригинальному выражению проблемы или в ясности, предполагая, что позднее возникнет дискуссия, касающаяся того, как были получены числа.

СЛОВАРЬ

If- если ,then- тогда , otherwise - иначе , go to - перейти к ;
integer - целый , real - вещественный , complex - комплексный,
print - печать , begin - начало , end - конец , procedure - процедура,
sqrt - корень квадратный;
array - массив , value- значение , for - для , step - шаг , until -до,
do - выполнить;
while- при;
boolean - булевский , comment - примечание, label - метка , own- собственный.

ЛИТЕРАТУРА

1. Naur, P. (Ed) (1963). Пересмотренный отчет об алгоритмическом языке АЛГОЛ-60 . The Computer Journal,vol5, p.349.
2. Naur, P.(Ed) (1960). Отчет об алгоритмическом языке АЛГОЛ-60 . Numerical Mathematik, Bd. 2,s.106-36.
3. Dijkstra E.W. (1962). Букварь программирования на АЛГОЛе-60. London: Akademic Press.
4. Higman B.and Goodman R.H. (1963). Язык вычислений. Программированное введение в АЛГОЛ. London: English Universities Press (in press).
5. Mc Cracken D.D. (1962). Руководство к программированию на АЛГОЛе. London and New York: Wisley.
6. Dijkstra E.W. (1962). Опыт работы с АЛГОЛом-60 . The Computer Journal,vol. 5, p.125.
7. Duncan F.G. (1962). Внедрение АЛГОЛа-60 для English Electric KDF9 . The Computer Journal,vol5.
8. Duncan F.G. (1963). Ввод и вывод для АЛГОЛа-60 на KDF9 .
9. Heare, C.A.R. (1962). Отчет о трансляторе с АЛГОЛа для Эллиот . The Computer Journal, vol.5, p.127.
10. Heare C.A.R. (1963). Система ввода и вывода для АЛГОЛа Эллиот . The Computer Journal vol.5, p.345.
11. Hockniy R.W. (1962), ABS/2ALGOL: Расширение АЛГОЛа-60 для целей индустрии . The Computer Jeurnal , vol.4,p.292.
12. Woodger M. (1960). Введение к АЛГОЛу-60 . The Computer Journal. vol.3,p.67.
13. Knuth D.E., add Merner, J.N (1961). ALGOL 60 Confidential . Communications of the A.C.M., vol.4, p.268.

Рукопись поступила в издательский отдел
22 февраля 1966 г.