Министерство образования и науки Российской Федерации
Московский физико-технический институт
(государственный университет)
Заочная физико-техническая школа

ИНФОРМАТИКА и ИКТ

Элементы программирования

Задание №1 для 11-х классов

(2016 – 2017 учебный год)

Долгопрудный, 2016

Составитель: В.В. Мерзляков, ассистент кафедры информатики СУНЦ МГУ

Информатика: задание №1 для 11-х классов (2016 – 2017 учебный год). 2016, 28 с.

Дата отправления задания - 20 октября 2016 г.

Составитель:
Мерзляков Василий Владимирович

Подписано 02.09.16. Формат 60×90 1/16.
Бумага типографская. Печать офсетная. Усл. печ. л. 1,75.
Уч.-изд. л. 1,55. Тираж 600. Заказ №33-з.

Заочная физико-техническая школа
Московского физико-технического института
(государственного университета)

ООО «Печатный салон ШАНС»

Институтский пер., 9, г. Долгопрудный, Москов. обл., 141700.
ЗФТШ, тел/факс (495) 408-51-45 – **заочное отделение,**
тел./факс (498)744-63-51 – **очно-заочное отделение,**
тел. (499) 755-55-80 – **очное отделение.**

e-mail: zftsh@mail.mipt.ru

Наш сайт: www.school.mipt.ru

§1. Алфавит языка Pascal

Изучение любого нового языка всегда начинается с алфавита. В алфавит языка Pascal входят следующие элементы:

1) Заглавные и строчные латинские буквы, символ подчёркивания (по грамматике языка символ подчёркивания считается буквой): _, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z.

2) Цифры: 1, 2, 3, 4, 5, 6, 7, 8, 9, 0.

3) Знаки операций: + (плюс), – (минус), * (умножить), / (разделить), < (меньше), > (больше), <= (меньше или равно), > = (больше или равно), = (равно), <> (не равно). Последний знак состоит из знаков «меньше» и «больше», записанных без пробелов.

4) Знаки пунктуации, специальные символы:

{ } или (* *)	Скобки комментариев
[ ]	Выделение индексов массивов, элементов множеств
' '	Выделение символа или строковой константы
( )	Выделение выражений, списков параметров
:=	Знак оператора присваивания
;	Разделение операторов и объявлений
:	Отделение переменной или константы от типа. Отделение метки от оператора
=	Отделение имени типа от описания типа. Отделение константы от её значения
,	Запятая для разделения элементов в списке
..	Разделение границ диапазона
.	Конец программы, отделение целой части от дробной
#	Обозначение символа по его коду

В таблице приведены не все знаки пунктуации, а лишь те, которые будут использоваться при дальнейшем изложении.

5) Служебные зарезервированные слова.

Некоторые слова имеют предопределённое значение и используются в качестве элементов при построении сложных конструкций языка.

Приведём список зарезервированных служебных слов, которые нам понадобятся в дальнейшем:

```
and, array, begin, case, const, div, do, downto, end, for, if, mod, not, of, or, program, repeat, string, then, to, type, until, var, while, xor.
```

§2. Структура программы

Рассмотрим общую структуру программы на языке Pascal. Программа состоит из 2 частей: разделов описаний и раздела действий (команд, операторов). Раздел операторов представляет собой некую последовательность команд (операторов), которые должен выполнить компьютер. Другими словами, раздел действий – это собственно программа. В разделах описаний программист сообщает компьютеру, какие объекты он будет использовать в своей программе, сколько памяти под них нужно выделить и т. д.

Из всего перечисленного выше обязательным для каждой программы является лишь раздел действий. Разделов описания в программе может и не быть (хотя в содержательных задачах они будут). Раздел действий начинается с ключевого слова `begin`, далее записывается список операторов (собственно программа) и в конце пишется ключевое слово `end` и ставится точка. Внутри списка все операторы отделяются друг от друга специальным символом – точкой с запятой. В этом задании мы познакомимся со следующими операторами:

1) пустой оператор;
2) составной оператор;
3) оператор присваивания;
4) условный оператор;
5) операторы цикла;
6) операторы вывода;
7) операторы ввода.

Начнём с пустого оператора. В языке Pascal пустой оператор – это просто ничего. Он не содержит никаких символов и не выполняет никакого действия**.** Рассмотрим пример программы состоящей из одного пустого оператора:

```
Begin
End.
```

Эта программа начинается и сразу же заканчивается, не выполняя никаких содержательных действий. Рассмотрим другой пример:

```
Begin
; ; ;
End.
```

Эта программа состоит уже из четырёх пустых операторов, отделённых друг от друга точками с запятой.

Для того чтобы познакомиться с разделами описаний, сначала нужно изучить объекты, которые мы можем использовать в программе.

§3. Константы и переменные

Познакомимся с двумя важнейшими в программировании понятиями.

Определение 1. ***Константой*** *называется объект, который получает значение до начала выполнения программы и не может менять его в ходе выполнения.*

Определение 2. ***Переменной*** *называется объект, который может менять своё значение в ходе выполнения программы.*

С каждой переменной, используемой в программе, связывается область в оперативной памяти компьютера, размер которой зависит от типа объекта. Каждая константа и каждая переменная имеет своё имя (идентификатор), по которому мы и обращаемся к этим объектам, чтобы с ними работать. Каждое имя должно быть уникальным, чтобы не возникала ситуация неопределённости – какой объект выбирать. Если в программе две переменные имеют одно и то же имя, компьютер откажется выполнять данную программу.

Правила именования в языке Pascal следующие: именем может являться любая последовательность латинских букв и цифр, начинающаяся с буквы. При этом заглавные и строчные латинские буквы не различаются, то есть имена aBBA и AbBa на самом деле обозначают один и тот же объект. Ещё одна интересная особенность построения имён заключается в том, что символ подчёркивания (_) считается латинской буквой, поэтому он также может входить в состав имени и, более того, имя с него может начинаться. Последнее правило именования заключается в том, что имена переменных или констант не должны совпадать со служебными зарезервированными словами языка.

Каждая переменная и каждая константа помимо имени ещё имеет свой тип. Тип определяет три вещи:

1) размер области оперативной памяти, отводимой под соответствующую переменную;

2) множество значений, которые может принимать соответствующая константа или переменная;

3) набор операций, которые можно выполнять с соответствующей константой или переменной.

Для того чтобы определить в программе константу или переменную, её нужно сначала описать до раздела действий. Переменные описываются следующим образом: сначала записывается ключевое слово var, затем ставится пробел, указывается имя, которое мы хотим дать нашей переменной, ставится двоеточие и записывается тип переменной. После этого ставится точка с запятой. Например, запись

```
var x:integer;
```

означает, что в нашей программе будет использоваться переменная с именем x, имеющая тип integer (целое число). Если в программе будет несколько переменных одного типа, то можно до двоеточия перечислить их имена через запятую, а не выписывать отдельную строчку для каждой переменной. Пример: `var x,y:integer;`

Для описания константы необходимо записать ключевое слово const, затем указать имя константы (отделив его как минимум одним пробелом от ключевого слова), поставить знак равенства и тут же задать её значение, например:

```
const N=1000;
```

После окончания описания константы также ставится точка с запятой. В программе принято сначала описывать константы (если они есть), а уже затем переменные (а они есть практически в любой программе).

Замечание. Термином «константа» в программировании принято обозначать ещё одно понятие – если в программе встречается некоторое конкретное значение (например, число 1000), то оно также называется константой соответствующего типа.

§4. Числовые типы переменных. Оператор присваивания

Рассмотрим два основных числовых типа переменных.

1) INTEGER. Этот тип характеризует целые числа. Переменные этого типа занимают в оперативной памяти 4 байта и могут принимать значения из диапазона [–2147483648, 2147483647]. Точное значение запоминать необязательно, главное помнить, что переменная этого типа может вмещать целые числа примерно до 2 миллиардов по модулю.

2) REAL. Этот тип предназначен для работы с вещественными (действительными) числами. Переменные этого типа занимают в оперативной памяти 8 байт. При записи констант этого типа целая часть числа отделяется от дробной точкой, а не запятой, как в математике, например: 3.14.

Оператор присваивания позволяет изменить значение любой переменной программы (присвоить ей новое значение). Этот оператор выглядит следующим образом: записывается имя переменной, затем знак присваивания (:=), а потом значение, которое мы хотим присвоить переменной. Присвоить можно константу или выражение соответствующего типа.

Пример 1.

```
X:=5; {в переменную X присвоили число 5}
Y:=X; {в переменную Y присвоили текущее значение переменной X}
Z:=X+Y; {в переменную Z присвоили сумму текущих значений переменных X и Y}
```

Если переменной присвоено некоторое значение, то в дальнейшем в программе при вычислениях вместо её имени будет подставляться это значение, пока мы не присвоим ей новое.

Пример 2.

```
X:=5;
Y:=X+4; {в переменную Y запишется число 9, так как текущее значение переменной X равно 5}
```

При использовании операторов присваивания необходимо соблюдать **правило совместимости типов**. Это правило заключается в том, что тип присваемого значения должен соответствовать типу переменной, которой мы хотим это значение присвоить. Есть исключение из этого правила: переменной типа `real` можно присвоить целое значение.

§5. Арифметические выражения

Арифметические выражения состоят из операций и операндов. В языке программирования Pascal существует шесть операций: сложение (обозначается знаком «+»), вычитание (обозначается знаком «–»), умножение (обозначается знаком «*»), деление (обозначается знаком «/»), деление нацело (обозначается словом «`div`») и взятие остатка от деления нацело (обозначается словом «`mod`»). Слова `div` и `mod` являются служебными зарезервированными.

Важным понятием в арифметике является понятие операнда. Операндами называются те объекты, над которыми выполняется арифметическая операция. В математике различные операции могут иметь разное количество операндов, но все арифметические имеют два операнда. Операндом для операции может являться как одиночное число или имя переменной, так и целое арифметическое выражение. Рассмотрим выражение (2+2)*2. У операции сложения операндами являются два числа 2, а у операции умножения правый операнд – это число 2, а левый – это выражение в скобках (2+2). Прежде чем выполнять операцию, необходимо вычислить оба её операнда.

Приоритет операций в Паскале точно такой же, как и в математике. Сначала выполняются операции умножения, деления, `div` и `mod` (это тоже операции деления), а потом операции сложения и вычитания. Операции одного приоритета выполняются слева направо. Для изменения порядка действий можно использовать круглые скобки. Операции в скобках имеют более высокий приоритет, чем операции вне скобок. Так при вычислении выражения 2+2*2 получается число 6, потому что операция умножения имеет более высокий приоритет, чем сложение, и, следовательно, выполняется первой. Если же записать выражение (2+2)*2, то при вычислении получается число 8, потому что сложение в скобках выполняется раньше умножения.

Рассмотрим, как определить тип результата при вычислении арифметического выражения. Операции сложения, вычитания и умножения выдают целый результат, если оба их операнда целые, и вещественный, если хотя бы один из операндов – вещественный. Операция деления «/» **всегда** выдаёт вещественный результат. Даже если мы 4 делили на 2, всё равно в итоге получается нецелое число. На первый взгляд это кажется странным, но в отличие от математики в программировании каждое число кроме значения ещё имеет тип, и если типы у чисел не совпадают, то они НЕ считаются равными. Нужно уяснить, что **1 ≠ 1.0.** Это несложно понять, если помнить, что раз числа 1 и 1.0 имеют различные типы, то будут представлены совершенно разными последовательностями битов. Операции `div` и `mod` всегда выдают целый результат и, в отличие от всех остальных арифметических операций, могут иметь только целые операнды. Попытка применить данные операции к вещественным числам приведёт к тому, что программа просто не будет работать.

Давайте подробнее познакомимся с двумя последними операциями. Операция `a div b` выдаёт целую часть от деления числа `a` на число `b`. То есть `5 div 2 = 2`, а `3 div 7 = 0`. Операция `a mod b` выдаёт остаток от деления `a` на `b` по следующему закону

```
a mod b = a – ((a div b) * b)
```

Приведём примеры выполнения этих их операций для всех возможных знаков операндов:

```
 5 div  3 =  1;  5 mod  3 =  2;
-5 div  3 = -1; -5 mod  3 = -2;
 5 div -3 = -1;  5 mod -3 =  2;
-5 div -3 =  1; -5 mod -3 = -2;
```

Операндами в арифметическом выражении также могут быть стандартные математические функции, которые приведены в таблице ниже.

Функция	Комментарий	Тип аргумента	Тип Результата
abs(x)	$\|x\|$ — модуль x	integer, real	совпадает с типом аргумента
sqr(x)	x^2	integer, real	совпадает с типом аргумента
sqrt(x)	$\sqrt{x}$ — корень квадратный из x	integer, real	real
Pi	3.14159265358979323 85	нет	real
sin(x)	$\sin x$	integer, real	real
cos(x)	$\cos x$	integer, real	real
arctan(x)	$\operatorname{arctg} x$	integer, real	real
trunc(x)	отсекание дробной части x	real	integer
round(x)	округление x до ближайшего целого. Половины округляются в сторону увеличения модуля.	real	integer

Необходимо отметить, что функциям `sin` и `cos` угол следует подавать в радианах, а не в градусах! Также функция `arctan` возвращает результат в радианах.

§6. Операторы вывода. Модификаторы формата

Операторы вывода являются важнейшей частью языка программмирования, ведь только благодаря им, мы можем увидеть на экране компьютера результат работы нашей программы. В языке Pascal существует два оператора вывода: `write` и `writeln`. Правило их использования одно и то же: после слова `write` или `writeln` в скобках через запятую перечисляются параметры, которые мы хотим вывести (называемые списком вывода). Число этих параметров не ограничено. Разделителем между параметрами служит запятая:

```
writeln(параметр, параметр,...,параметр)
```

Существует три вида параметров: *константы*, *переменные* и *выражения* (например, арифметические выражения). Константы бывают числовые (это просто различные числа — целые и вещественные), логические и строковые. Любой текст, набранный с клавиатуры и заключённый в апострофы (одиночные кавычки), называется *строковой константой*. Если в текст нам нужно поместить апостроф, например, в слове O'key, на этом месте нужно набить два апострофа подряд вместо одного: `write('O''key')`. Все параметры в `write` или `writeln` независимы друг от друга, поэтому в одном и том же операторе могут встречаться параметры разных типов, в произвольном порядке.

При выполнении оператора вывода все параметры будут выведены в одной строке в том же порядке, в каком они перечислены в списке параметров. Любая константа, числовая или строковая, будет выведена так, как вы её написали в вызове `write` или `writeln` (в строковой константе начальный и конечный апострофы отображаться на экране не будут, а вместо двух апострофов, расположенных в строковой константе подряд, на экране появится в этом месте один); вместо переменной на экране появится её значение, а вместо арифметического выражения — результат его вычисления.

Между `write` и `writeln` существует единственное различие: после выполнения `writeln` курсор переходит на новую строку, а после выполнения `write` курсор остаётся в той же строке, и новый вывод данных с помощью `write` или `writeln` или ввод данных при помощи операторов ввода данных будут проходить в той же строке.

При выводе параметров пробелы между ними автоматически не вставляются, например, при печати чисел 1, 2, 3 с помощью `writeln(1,2,3)` все они сольются в одно число — `123`. Чтобы разделить выводимые элементы, можно поместить между ними символ пробела, например, `writeln(1,' ',2,' ',3)` или отформатировать вывод, поставив после каждого элемента списка вывода двоеточие и целое число (называемое **модификатором ширины поля**), которое указывает, сколько позиций на экране должна занимать выводимая величина, например, `writeln(1:3,2:3,3:3)`. Отметим, что элемент дополняется начальными пробелами слева с тем, чтобы соответствовать указанной

после двоеточия величине. Результаты выполнения двух последних операторов будут выглядеть так:

```
1 2 3
  1   2   3
```

Если указанное в модификаторе ширины поля число меньше, чем необходимо, то модификатор ширины поля игнорируется.

При выдаче на экран значений вещественных выражений в формате вывода полезно использовать ещё один модификатор, который записывается через двоеточие после модификаторы ширины поля и называется **модификатором точности**. Он будет обозначать количество символов после десятичной точки, которые мы хотим вывести. Например, при выводе результата стандартной функции `pi`, которая с машинной точностью выдаёт значение числа π, оператор `write(pi:0:0,pi:6:2, pi/2:2:0)` выдаст на экран:

```
3  3.14 2
```

Заметим, что при печати фиксированного количества цифр вещественного числа оно предварительно округляется по правилам математики. Если вещественное число содержит после десятичной точки меньше цифр, чем количество символов для печати, указанное в модификаторе точности, то число выводится с незначащими нулями, например, оператор `write(3.14:3:4)` выдаст на экран:

```
3.1400
```

Модификатор точности можно применять только к параметрам вещественного типа. Использование модификатора точности с параметрами других типов является критической ошибкой (программа не будет работать). Модификатор ширины поля можно использовать с любым типом параметра вывода.

§7. Операторы ввода

Операторы `read` и `readln` предназначены для задания значений переменным путём ввода их с клавиатуры. Правило их применения одно и то же: после слова `read` или `readln` в скобках через запятую перечисляются имена переменных, значения которых мы хотим ввести (список ввода). Число этих имён не ограничено. Запятая служит раз-делителем между именами переменных:

```
readln(имя, имя, ..., имя)
```

При срабатывании оператора `read` или `readln` выполнение программы будет приостановлено до тех пор, пока пользователь не введёт соответствующее количество значений. Вводимые значения должны быть того же типа, что и переменные. Если в `read` или `readln` переменных несколько, то они могут быть набиты в одной строке, но одно число от другого должно отделяться пробелом или переводом строки. Чтобы выполнить оператор `read` или `readln` после набивания значений с клавиатуры, нужно нажать клавишу «Enter». В результате переменные приобретут заданные вами значения. Между `read` и `readln` существует единственное различие: после выполнения `readln` курсор переходит на новую строку, игнорируя всю оставшуюся информацию в прежней строке, а после выполнения `read` курсор остаётся в той же строке, и новая набивка данных для `read` или `readln` будет проходить в той же строке. Но, так как после нажатия клавиши «Enter» курсор в любом случае переходит на новую строчку, для однократного ввода значений переменных разницу между операторами `read` и `readln` заметить невозможно. Тем не менее, в данном случае лучше использовать `readln`. Оператор `readln` можно использовать и без параметров вообще. Тогда программа просто будет находиться в режиме ожидания, пока пользователь не нажмёт клавишу «Enter». Такой оператор, например, удобно ставить самым последним оператором в программе. Тогда можно сразу посмотреть результат работы программы, а потом нажать «Enter», и только тогда работа программы завершится.

Замечание. Перед вводом данных с клавиатуры рекомендуется выдавать на экран приглашение, например:

```
write('Введите число a => ');
readln(a);
```

§8. Примеры простейших программ

Теперь воспользуемся всеми полученными знаниями и рассмотрим примеры простейших программ. Условимся в примерах не различать заглавные и строчные буквы, а ключевые слова выделять жирным шрифтом.

Пример 3. *Ввести координаты трёх вершин треугольника. Вывести его площадь. Гарантируется, что треугольник существует (ввод корректный).*

Решение. Для решения этой задачи нам потребуется 6 переменных, в которые будут введены значения координат (x1, y1, x2, y2, x3, y3). Можно сделать их целыми для простоты, а можно вещественными, чтобы решать задачу в более общем случае. Так же потребуются переменные, в которые мы запишем длины сторон треугольника (a, b, c) и площадь (S). Эти переменные однозначно должны быть вещественными, так как при вычислении расстояния между точками нам придётся извлекать квадратный корень, и, соответственно, результат получится вещественным. И ещё предлагается ввести вещественную переменную p, в которой будет храниться полупериметр треугольника, так как площадь будет вычисляться по формуле Герона. Приведём полный текст программы:

```
  Var x1,y1,x2,y2,x3,y3:integer; a,b,c,p,S:real;
      Begin
        Write('Введите координаты вершин
треугольника ');
        Readln(x1,y1,x2,y2,x3,y3);
        a:=sqrt(sqr(x1-x2)+sqr(y1-y2));
        b:=sqrt(sqr(x1-x3)+sqr(y1-y3));
        c:=sqrt(sqr(x2-x3)+sqr(y2-y3));
        p:=(a+b+c)/2;
        S:=sqrt(p*(p-a)*(p-b)*(p-c));
        Writeln('Площадь треугольника равна
',S:5:2);
        Readln;
      End.
```

Пример 4. *Идёт k-ая секунда суток (k вводится). Вывести, сколько полных часов h и полных минут m прошло с начала суток.*

Решение. В этой задаче все параметры целые. Решается она с помощью операций `div` и `mod`. Эти операции можно использовать для «срезания периодов» при переводе мелких единиц измерения в более крупные (например, секунд в минуты). Операция `div` нам выдаст количество полных периодов (сколько полных минут содержится в большом количестве секунд), а операция `mod` – количество единиц в последнем неполном периоде (сколько секунд не укладывается в полное количество минут). Приведём полный текст решения.

```
  Var k,h,m:integer;
  Begin
        Write('Введите номер секунды в сутках');
        Readln(k);
        h:=k div 3600;
        m:=k mod 3600 div 60;
        writeln('С начала суток прошло ',h,' часов
и ',m,' минут');
        readln;
   end.
```

Пример 5. *Вводится четырёхзначное число. Вывести произведение его цифр.*

Решение. Эта задача показывает ещё одно применение операций div и mod – выделение цифр из целого числа. Описанное ниже решение работает только для случая, когда количество цифр в числе заранее известно. В противном случае придётся использовать циклический алгоритм. Приведём текст решения.

```
  Var N,c1,c2,c3,c4:integer;
  Begin
        Write('Введите     целое     четырёхзначное
число');
        Readln(N);
        c1:=N div 1000;
        c2:=N div 100 mod 10;
        c3:=N div 10 mod 10;
        c4:=N mod 10;
        Writeln('Произведение  цифр  вашего  числа
равно ',c1*c2*c3*c4);
        Readln;
  End.
```

§9. Логический тип переменных

В языке Pascal кроме уже изученных нами числовых типов ещё есть логический, который называется Boolean. Переменные этого типа занимают 1 байт оперативной памяти и могут принимать всего два значения – true и false (истина и ложь). Логическим переменным можно присваивать значения точно так же, как и числовым. Так же можно выводить их значения на экран, а вот вводить их с клавиатуры нельзя!

В языке Pascal определены 6 операций сравнения, результатом которых является логическое значение. Это операции: «больше» (>), «больше или равно» (>=), «меньше» (<), «меньше или равно» (<=), «равно» (=), и «не равно» (<>). Например, операция `5 > 2` выдаст значение `true`, а операция `x<>3` выдаст значение `true`, если переменная X имеет любое значение, кроме 3. Сравнивать можно не только числа (причём как целые, так и вещественные), но и логические значения. При этом считается, что значение true больше, чем значение false.

Помимо операций сравнения ещё существуют и логические операции: AND (конъюнкция, логическое умножение, операция «И»), OR (дизъюнкция, логическое сложение, операция «ИЛИ»), NOT (отрицание, инверсия), XOR (строгая дизъюнкция, исключающее «ИЛИ», сложение по модулю 2). В скобках указаны возможные названия данных операций в алгебре логики. Операнды этих операций должны быть логического типа. Результат вычислений также будет логический. При этом операции AND, OR, XOR имеют по два операнда, а операция NOT – всего один, который записывается справа от названия операции. Названия логических операций являются ключевыми словами языка. Приведём таблицы результатов логических операций для всех возможных значений операндов (в алгебре логики такие таблицы называются таблицами истинности):

x	**not** x
false	true
true	false

X	y	x **and** y	x **or** y	x **xor** y
false	false	false	false	false
false	true	false	true	true
true	false	false	true	true
true	true	true	true	false

Логический результат даёт также стандартная функция `odd(x)`, которая применяется к целочисленному аргументу *x*:

`odd(x) = true`, если *x* нечётно;

`odd(x) = false`, если *x* чётно.

Приоритет операций в логическом выражении следующий:

1) Операция NOT.
2) Операции группы умножения AND, *, / ,div, mod
3) Операции группы сложения OR, XOR, +, -
4) Операции сравнения >, <, >=, <=, =, <>

Операции одного приоритета выполняются слева направо. Операции в круглых скобках имеют более высокий приоритет, чем операции вне скобок.

Пример 6. Записать логическое выражение, истинное в случае, когда переменная X имеет значение из отрезков [2,5] или [-1,1].

Решение. `(X>=2) AND (X<=5) OR (abs(X)<=1)`.

§10. Условный оператор

В рассматриваемых ранее задачах процесс вычисления был линейным, то есть программа не должна была выполнять разные действия в зависимости от того, какие данные ей ввели. Теперь вспомним задачи с ветвящимся алгоритмом.

Пример 7. Ввести номер года. Вывести слово YES, если год високосный, и NO, если он – не високосный.

По условию очевидно, что в зависимости от входных данных программа должна будет выполнить один из двух операторов вывода: Writeln('YES') или Writeln('NO'). При этом написать в программе нам придётся оба, а вот выполняться должен будет только один из них. Для того чтобы реализовывать подобные ветвления алгоритма, в языке Pascal существует условный оператор. В общем виде он выглядит следующим образом:

```
if логическое выражение
   then оператор
   else оператор
```

В этой конструкции слова `if`, `then` и `else` являются служебными зарезервированными словами языка. Работает эта конструкция так: сначала вычисляется логическое выражение, стоящее после `if`. Если получилось значение `true`, то выполняется оператор, стоящий после слова `then`, а если получилось значение `false`, то выполняется оператор, стоящий после слова `else`.

Обратите внимание, что внутри условного оператора нет никаких точек с запятой, поскольку он является единой конструкцией, а точка с запятой – это разделитель между операторами. Для удобства чтения программ принято условие записывать на одной строке, а ветви `then` и `else` начинать с новой строки, однако это не является синтаксическим правилом языка.

В качестве примера условного оператора рассмотрим решение задачи, поставленной выше. Год считается високосным, если он делится нацело на 400, или если он делится нацело на 4, но не делится нацело на 100. Приведём полный текст решения:

```
var y:integer;
begin
  write('Введите номер года ');
  readln(y);
  if (y mod 400 = 0)or(y mod 4 = 0)and(y mod
100 <> 0)
    then writeln('YES')
    else writeln('NO');
end.
```

По грамматике языка после слов `then` и `else` должен стоять только один оператор языка. То есть запись `if x>0 then x:=4; y:=0 else z:=9;` является синтаксически неверной. А как быть, если всё-таки нужно выполнить более одного оператора? Для таких случаев в языке Pascal предусмотрен составной оператор, который позволяет превратить группу операторов в один. Выглядит он следующим образом: сначала записывается ключевое слово `begin`, далее – интересующая нас последовательность операторов через точку с запятой, а в конце пишется ключевое слово `end`. В отличие от конца программы, точка после этого слова не ставится. Слова `begin` и `end` называются операторными скобками. Запишем правильную версию условного оператора, приведённого выше: `if x>0 then begin x:=4; y:=0 end else z:=9;`

Обратите внимание на следующий тонкий момент: если требуется выполнить более одного оператора в ветке `then`, и при этом мы забудем написать операторные скобки, то это является синтаксической ошибкой, и программа просто не будет работать. Если же забыть написать операторные скобки в ветке `else`, то программа работать будет, но не так, как предполагалось. Рассмотрим пример: `if x>0 then y:=9 else z:=8; c:=5;`

В этом примере условный оператор заканчивается после `z:=8;` в то время как оператор `c:=5;` является следующим оператором программы и выполняется независимо от результата сравнения *x* с нулём. Если же написать операторные скобки, то присваивание в `c` числа 5 произойдёт только в случае `x<=0`.

Последнее замечание заключается в том, что в ветке `else` в качестве оператора может стоять и пустой оператор. Рассмотрим следующий пример.

Задача. *Вводятся 3 целых числа – a,b,c. Требуется в переменную a записать минимальное из этих чисел, в b – среднее и в c – максимальное.*

Решение. Алгоритм решения этой задачи такой: сначала сравним значения переменных `a` и `b`, если значение `a` – больше, поменяем их местами. После этого сравним значения переменных `a` и `c`, и если значение `a` – больше, поменяем их местами. После этих двух сравнений в переменной `a` гарантированно окажется наименьшее из трёх чисел. Осталось сравнить переменные `b` и `c`, и в случае, когда в переменной `b` находится большее значение, поменять их местами.

Очевидно, что в этом алгоритме у нас три сравнения, следовательно, три последовательных условных оператора. При этом в каждом из них какие-то действия (поменять местами значения двух переменных) нужно выполнять только в ветке `then`, в ветке `else`(например, если в первом сравнении в переменной `a` находится уже более маленькое число, чем в переменной `b`) никаких действий выполнять не нужно. Но мы всё равно будем её записывать, чтобы избегать путаницы. Приведём полный текст решения, используя для обмена значений двух переменных дополнительную переменную `x`.

```
var a,b,c,x:integer;
begin
  writeln('введите три целых числа ');
  readln(a,b,c);
  if a>b then begin x:=a; a:=b; b:=x end
else;
  if a>c then begin x:=a; a:=c; c:=x end
else;
  if b>c then begin x:=b; b:=c; c:=x end
else;
  writeln(a,b,c);
  readln
end.
```

§11. Операторы цикла

Зачастую в задаче нужно повторять одни и те же действия много раз. Повтор некоторого фрагмента программы несколько раз называется циклом. Рассмотрим следующий пример.

Пример 8. *Вывести на экран квадраты чисел от 1 до 100.*

Очевидно, что для решения этой задачи нам придётся 100 раз выполнять команду вывода соответствующего числа на экран. Писать 100 операторов вывода как-то не хочется (слишком трудоёмко), поэтому будем знакомиться с операторами цикла. В языке Pascal существует три оператора цикла: `for`, `while`, `repeat`. Начнём с цикла `for`. Этот оператор цикла реализует следующую идею: «Повторять некоторую последовательность команд N раз, где N известно до начала повторения». Познакомимся с синтаксисом этого оператора.

`for` имя переменной := начальное значение **`to`** конечное значение **`do`** оператор

В этой конструкции переменная, стоящая после слова `for`, называется параметром или счётчиком цикла, а оператор, стоящий после слова `do`, называется телом цикла. Начальное и конечное значения, по сути, являются константами или выражениями одного типа со счётчиком. Алгоритм выполнения цикла `for` следующий:

1) вычисляются начальное и конечное значения;

2) счётчику цикла присваивается начальное значение;

3) значение счётчика сравнивается с конечным. Если оно больше конечного, то выполнение цикла завершается и начинает выполняться следующий оператор программы, в противном случае переход к пункту 4;

4) выполняется тело цикла;

5) значение счётчика увеличивается на 1;

6) переход к пункту 3.

В качестве примера рассмотрим решение задачи, поставленной выше. В качестве счётчика будем использовать переменную i.

```
var i:integer;
begin
   for i:=1 to 100 do write(i*i,' ');
   Readln
end.
```

Согласитесь, что решение фактически в одну строчку выглядит гораздо приятнее, чем в 100 строк (если не пользоваться оператором цикла).

Необходимо сделать несколько замечаний по поводу цикла `for`.

1) Типы счётчика начального и конечного значений должны совпадать, при этом в настоящий момент из известных нам типов можно использовать только `integer` и `boolean`. Вещественный тип использовать нельзя.

2) Начальное и конечное значения вычисляются один раз до начала цикла (и после не перевычисляются). Рассмотрим пример. `i:=1; for i:=i to i do writeln('HI');` Этот оператор цикла выполнится всего один раз, а не бесконечно много.

3) В теле цикла значение счётчика изменять нельзя. Так прописано в стандарте языка Pascal, и это требование поддерживается в системах семейства Delphi. Однако в системах семейства Borland Pascal значение счётчика изменять можно, что может приводить к непредсказуемым последствиям (поэтому будем считать, что независимо от системы значение счётчика изменять нельзя).

4) После завершения цикла значение счётчика не определено, то есть нельзя считать, что оно равно конечному значению или больше на единицу и пользоваться этим в дальнейшем алгоритме.

5) Тело цикла по грамматике должно состоять только из 1 оператора. Если же там по алгоритму должно быть несколько, нужно использовать составной оператор.

6) Можно слово **to** заменить на слово **downto**. В этом случае значение счётчика после каждого выполнения тела цикла будет уменьшаться на 1, а выход из цикла произойдёт, когда значение счётчика окажется меньше, чем конечное.

Перейдём к знакомству с другими операторами цикла: `while` и `repeat`. Эти операторы реализуют следующие идеи: «Повторять некоторую последовательность команд пока выполняется некоторое условие» (цикл `while`) и «Повторять некоторую последовательность команд до тех пор, пока не выполнится некоторое условие» (цикл `repeat`). Познакомимся с синтаксисом оператора цикла `while`.

`while` условие **`do`** оператор.

После слова `while` должно быть логическое выражение (называемое условием), которое может принимать значение `true` или `false`. Оператор, стоящий после `do`, аналогично оператору `for` является телом цикла. Выполняется цикл `while` так:

1) вычисляется значение условия, если получилось `true`, то переход к пункту 2, иначе выход из цикла и переход на следующий оператор программы;

2) выполняется тело цикла;

3) переход к пункту 1.

Фактически оператор `while` является многократным применением оператора `if` с пустой веткой `else`. Аналогично оператору `for`, тело цикла должно состоять из 1 оператора. Однако в отличие от цикла `for` возможна ситуация, когда цикл будет выполняться бесконечное количество раз (зациклится). Например, `while 2*2=4 do…`
Что написать после `do`, совершенно не важно, важно, что оно будет выполняться, пока `2*2=4`, а это всегда так и никогда не изменится. Значит, чтобы избегать зацикливания, параметры условия должны быть переменными, например `while x*x=4 do …` Хотя это тоже не гарантирует отсутствие зацикливания.

Осталось познакомиться с последним оператором цикла. Рассмотрим его синтаксис.

```
repeat
  Оператор 1;
  Оператор 2;
….
  Оператор N
until условие
```

Все операторы, написанные между `repeat` и `until`, являются телом цикла. Это выгодно отличает оператор `repeat` от других циклов – составной оператор здесь не требуется, а операторными скобками можно считать слова `repeat` и `until`. Работает этот оператор по следующему алгоритму:

1) выполняется тело цикла;

2) вычисляется значение условия. Если получилось `true`, то выход из цикла и переход к следующему оператору программы, в противном случае переход к пункту 1.

Отличительная особенность оператора цикла `repeat` заключается в том, что тело всегда выполняется, по крайней мере, один раз. Это нужно учитывать в задачах при выборе оператора цикла. Аналогично оператору `while`, цикл `repeat` может зациклиться, правда в случае, когда условие никогда не принимает значение `true`, например, `repeat…until 2*2=5.`

§12. Примеры задач на операторы цикла

Задача 1. *Ввести значения x и n. Вычислить сумму ряда* $1+x+x^2+x^3+\ldots+x^n$.

Решение. Сумма вычисляется накопительным путём: сначала вычисляется каждое слагаемое, а потом оно добавляется в сумму. Каждое слагаемое также вычисляется накопительным путём.

```
var x,n,i,s,a:integer;
begin
  write('введите значения x  и n');
  readln(x,n);
  S:=1; a:=1
  for i:=1 to n do
      begin
          a:=a*x; s:=s+a
      end;
   writeln(s);
  readln
end.
```

Заметим, что эту задачу можно решить, воспользовавшись формулой суммы первых `n` членов геометрической прогрессии, но наша цель не решить конкретную задачу, а понять принцип накапливания (суммы). Ибо далеко не любая последовательность чисел является геометрической или арифметической прогрессией.

Задача 2. *Вводится последовательность натуральных чисел. Признак конца ввода – число 0. Вывести среднее арифметическое чисел из последовательности (0 не учитывается).*

Решение. Эта задача относится к очень важному классу задач – обработка последовательностей с признаком конца и заранее неизвестным количеством элементов. В таких задачах нужно пользоваться операторами `while` и `repeat`. Причём, если оговаривается, что последовательность непустая, можно использовать `repeat`, а если этого не оговаривается, как в данной задаче, то нужно обязательно пользоваться циклом `while`, так как цикл `repeat` всегда выполняется, по крайней мере, один раз. Приведём полный текст решения. Переменная `s` подсчитывает сумму элементов последовательности, а переменная `k` – их количество.

```
var a,s,k:integer;
begin
  s:=0; k:=0;
  writeln('Введите последовательность
натуральных чисел. Ноль – признак конца');
  read(a);
  while a<>0 do
     begin  s:=s+a; k:=k+1; read(a) end;
  Writeln('среднее арифметическое=',s/k:0:4)
end.
```

Обратите внимание, что для ввода данных используется оператор `read`. Это позволяет набивать элементы в строчку через пробел. Если же использовать `readln`, то набивать значения придётся в столбик, что неудобно. Ещё одно важное замечание в этой задаче – оператор ввода внутри цикла должен быть последним, чтобы сразу попадать на проверку признака конца. Эти замечания относятся ко ВСЕМ задачам на обработку последовательностей с признаком конца.

Задача 3. *Ввести целое число n. Вывести YES, если оно простое, и NO, если оно составное.*

Решение. Эта задача демонстрирует сразу две важные вещи. Во-первых, как проверять делимость целых чисел, а во-вторых, технику флажков. Флажком называется переменная, которая имеет некоторое начальное значение и меняет его, если происходит определённое событие. Как правило, флажок имеет тип `boolean`.

В нашей задаче мы будем перебирать числа от 2 до квадратного корня из n и проверять, делится ли n на каждое из них. Изначально предположим, что n – простое, и присвоим флажку значение true, но если n поделится на какое-нибудь число, это будет значить, что оно составное, и, соответственно, флажок «упадёт» на значение false. Проверять на делимость нужно, сравнивая остаток от деления с нулём.

```
var n,i:integer;
        f:boolean;
 begin
  Write('Введите значение n ');
  Readln(n);
  f:=true;
  for i:=2 to round(sqrt(n)) do
    if n mod i = 0 then f:=false else;
    if f=true
        then writeln('YES')
        else writeln('NO');
Readln
end.
```

Контрольные вопросы

1(2). Укажите, какие из перечисленных последовательностей символов могут быть именами переменных, а какие нет и почему. Запятые являются разделителями между последовательностями:

TheChosenOne, Henry_ford, variki, var_y:real, ____, downto, C, _C++, beginf

2(2). Верна ли следующая программа? Если да, то сколько операторов она содержит, если нет, то почему:

```
var x,y:integer;
begin
 readln(y);;
 x:=0;
 if x>0 then;
 writeln(x,y);
 readln;
end.
```

3(2). Возможен ли следующий оператор присваивания при каком-нибудь описании переменной x? Ответ обосновать.

```
x:=round(sqrt(x)) mod 6*x+x mod 2/x.
```

4(2). Что будет выведено на экран после выполнения программы, если с клавиатуры введены числа 5, 4, 3, 2, 1?

```
var a,b,c:integer;
begin
    readln(a, b, c, c, a);
    write(a, b, c);
   end.
```

5(2). Сколько раз выполнится цикл и что будет выдано на экран в предложенном фрагменте программы:

```
i:=134; y:=0;
for i:=1 to i do y:=y*4; writeln(y)
```

Задачи

Внимание! Перед написанием собственно текста программы необходимо объяснить свой алгоритм на русском языке и прокомментировать все используемые переменные. Программы без пояснений проверяться не будут! Инструментарий, который не описан в тексте задания, использовать нельзя.

1(2+2). Ввести с клавиатуры 2 натуральных числа – N и M. Проверить, являются ли они взаимно простыми (то есть, не имеют общих делителей кроме единицы). В качестве ответа вывести слово YES или NO. Плюс два балла, если алгоритм решения задачи эффективный.

2(2+1). Будем считать, что у осы 6 ног, у паука – 8, у тигра – 4 и у кенгуру – 2. На вход программе подаётся общее количество ног – N. Требуется подсчитать, сколькими способами можно составить это количество, используя описанных выше животных. Если не удастся найти ни одного способа, то сообщить об этом специальным выводом: IMPOSSIBLE. Плюс 1 балл, если алгоритм эффективный.

3(2+1). Даны три натуральных числа — длины сторон треугольника. Определите, существует ли треугольник с такими сторонами, и если он существует, то определите его тип (остроугольный, тупоугольный, прямоугольный). На вход программе подаются 3 натуральных числа, не превосходящих 10 000. Выведите тип треугольника (ACUTE, OBTUSE, RECTANGULAR) или сообщение, что такого треугольника не существует (IMPOSSIBLE). Можно получить 1 бонусный балл к основным за эффективный алгоритм.

4(3). Вводится последовательность натуральных чисел. Признак конца ввода – ноль. Необходимо проанализировать подпоследовательности из подряд идущих чисел «14» и вывести количество чисел в самой длинной из них.

5(3). Дано натуральное число N, не превосходящее 1000000. Требуется выяснить, является ли последовательность цифр данного числа при просмотре справа налево возрастающей (строго). Например, для числа 4321, ответ положительный, а для числа 4312 – отрицательный. В качестве ответа программа должна выводить YES или NO.

6(2). По заданным натуральным числам n и m вычислите

$$\sqrt{m+\sqrt{2m+\ldots+\sqrt{(n-1)m+\sqrt{nm}}}}\ .$$

Вводятся натуральные $n \le 100$ и $m \le 100$. Выведите значение указанного выражения с точностью до 6 значащих цифр после десятичной точки.

7(2). Вводятся целые числа `a, b, c, d, e, f`, каждое из которых не превосходит по модулю 1000, и рассматривается уравнение $a*x^5+b*x^4+c*x^3+d*x^2+e*x+f=0$. Требуется написать программу, которая будет находить все его целые корни в промежутке от -10 до 10 и выводить их на экран в столбик в порядке возрастания. Если уравнение не имеет целых корней в указанном промежутке, то вывести фразу NO SOLUTION.

8(4). В некоторой игре используются карточки с номерами от 1 до N. На вход программе сначала подаётся число N, не превосходящее 1000, а затем N-1 число - номера карточек (в произвольном порядке). Программа должна вывести номер оставшейся карточки или сообщить, что ввод ошибочный (если программа сумела это понять), словом ERROR. Инструментарий, который не рассматривался в задании (например, массивы), не использовать. При заданных ограничениях не все ошибки ввода возможно отловить, но таких тестов не будет.

9(8). Вводится последовательность натуральных чисел, не превосходящих 20000. Ноль – признак конца. Требуется вывести минимальное из произведений двух элементов последовательности, которое делится на 39. Множители, составляющие произведение, могут быть равными, если какое-то число встречается в последовательности более одного раза. Если же требуемое произведение отсутствует, то вывести просто минимальное произведение двух элементов последовательности. Гарантируется, что в последовательности будет не менее двух элементов. Инструментарий, который не рассматривался в задании (например, массивы), не использовать.

10(10). На вход программе подаётся последовательность целых чисел, каждое из которых не превосходит по модулю 10000 . Числа записаны в одной строке через пробел. Признак конца ввода (в последовательность не входит) – число 2*109. Требуется найти и вывести максимальное нечётное произведение двух элементов этой последовательности, которые располагаются друг от друга на расстоянии не менее 4 (между ними должно быть как минимум два других элемента последовательности). Гарантируется, что в последовательности будет не менее пяти элементов. Если искомого нечётного произведения не существует, то вывести сообщение IMPOSSIBLE. Инструментарий, который не рассматривался в задании (например, массивы), не использовать.