Содержание

Алгоритм и программа

Наш курс посвящен изучению не только языка *Pascal*, но и некоторых *алгоритмов*, решающих наиболее известные задачи программирования, поэтому начнем мы со знакомства с некоторыми основополагающими понятиями.

Алгоритм - это последовательность действий, которые необходимо выполнить, чтобы решить поставленную задачу.

Программа же представляет собой набор команд на языке, понятном исполнителю, реализующий некоторый *алгоритм*. В нашем случае исполнителем является компьютер, а языком программирования будет ***язык высокого уровня*** *Pascal*. К сожалению, любой *язык высокого уровня* удобен только человеку, пишущему или отлаживающему *программу*, но совершенно непонятен компьютеру. *Программа* на таком языке называется исходным текстом и хранится во внешнем файле с расширением .pas.

Для перевода программы на *язык низкого уровня*, понятный исполнителю-компьютеру, существуют специальные программы-переводчики - ***компиляторы***. Результатом работы *компилятора* (иными словами, результатом процесса компиляции) является ***исполняемый код***, который записывается в файл с расширением .exe.

Свойства алгоритма

Любой *алгоритм* должен обладать следующими свойствами:

- массовостью (*алгоритм* должен уметь решать не одну конкретную задачу, а целый класс однотипных задач);
- результативностью (*алгоритм* должен выдавать результат своей работы);
- определенностью (на каждом шаге выполнения *алгоритма* исполнитель должен точно знать, какой шаг будет следующим).

Эти же свойства присущи и *программам*, реализующим *алгоритмы*. Если же хотя бы одно из них оказывается невыполненным, *программа* полностью теряет смысл.

Компиляция, отладка и тестирование

Никто не станет спорить с тем, что неграмотно написанный текст очень сложно, а порой и вовсе невозможно правильно перевести на другой язык. Это верно для естественных языков, это верно и для языков программирования. Но если переводчик-человек иногда может как-то догадаться, что же именно хотел сказать автор неграмотного текста, то программе-переводчику такое не по силам. Любой *компилятор* требует, чтобы *программа*, подаваемая ему для перевода, была абсолютно правильно составлена.

В языке программирования, как и в любом другом языке, существуют ***синтаксис*** - правила записи его конструкций - и ***семантика*** - смысл его конструкций. Компилятор проверяет только *синтаксис*. Поиском же семантических ошибок занимается программист в процессе *тестирования* и *отладки* своей *программы*

Отладка - это поиск и исправление ошибок в *программе*. ***Тестирование*** - это составление специальных наборов входных и *выходных данных*(тестов), а затем исполнение *программы* и проверка полученных результатов в поисках возможных семантических или логических ошибок.

Чтобы уменьшить затраты времени и сил на *отладку*, нужно писать синтаксически и логически правильные *программы*. Технологиям написания надежных *программ*, их *тестирования* и *отладки* будет посвящена последняя лекция нашего курса.

Средства разработки программ

Существует довольно большое количество средств написания *программ* на языке *Pascal*, позволяющих составлять, компилировать, исполнять и отлаживать *программы* на этом удобном языке структурного программирования[1]. Самыми известными сегодня являются Turbo *Pascal* (он же Borland*Pascal*), *Object Pascal* (не путать с *Delphi*) и Free *Pascal*. Их различные, в том числе и бесплатные, версии можно найти в Сети. Для определенности мы будем опираться на самую распространенную (хотя и не во всем соответствующую стандартам ISO) реализацию - Turbo *Pascal*.

Итак, в составе среды разработчика Turbo *Pascal* имеются:

- текстовый редактор, в котором можно набирать тексты *программ* ;
- *компилятор*, превращающий *исходные тексты* в *исполняемый код* ;
- отладчик, помогающий обнаруживать и исправлять ошибки в *программе*

Из многочисленных возможностей, предоставляемых средой Turbo *Pascal*, мы упомянем лишь самые важные - те, без которых написание *программ*становится совсем уж затруднительным.

- Нажатие клавиш F1, Alt+F1, Ctrl+F1 открывает экранную подсказку.
- Нажатие клавиши F2 позволяет сохранить *исходный текст программы*.
- Нажатие клавиши F3 открывает диалог выбора нужного файла (по умолчанию, отображаются только файлы с расширением .pas).
- Нажатие клавиши Alt+F5 показывает консоль (см. п. "Ввод и вывод: консоль" ниже) с результатами работы *программы*.
- Нажатие клавиши Ctrl+F9 начинает процесс выполнения *программы*. Если она еще не была откомпилирована, предварительно будет вызван *компилятор*
- Клавиши F7 и F8 обеспечивают трассировку - *пошаговое выполнение программы*, позволяющее проследить за процессом ее выполнения.
- Дополнительное окно Debug/Watch показывает текущее состояние выбранных *переменных*.

Более подробно о возможностях среды Turbo *Pascal* можно узнать в любом руководстве по ее использованию (в том числе и во встроенном Help).

Структура Pascal-программы

Для того чтобы Pascal-компилятор правильно понял, какие именно действия от него ожидаются, ваша *программа* должна быть оформлена в полном соответствии с *синтаксисом* (правилами построения *программ*) этого языка.

Приступим же к изучению этих правил.

Любая *Pascal*-программа может состоять из следующих блоков (напомним, что квадратными скобками здесь и далее помечены необязательные части):

```
program <имя_программы>;
        [ uses <имена_подключаемых_модулей>;]          (см. лекцию 13)
        [ label <список_меток>;]
                        (см. п. "Метки и безусловный переход" ниже)
        [ const <имя_константы> = <значение_константы>;]
              (см. п. "Константы" ниже)
        [ type <имя_типа> = <определение_типа>;]       (см. лекцию 2)
        [ var <имя_переменной> : <тип_переменной>;]
                 (см. п. "Переменные и типы данных" ниже)
        [ procedure <имя_процедуры> <описание_процедуры>;]
                       (см. лекцию 8)
        [ function <имя_функции> <описание_функции>;]
        begin   {начало основного тела программы}
        <операторы>
        end.  (* конец основного тела программы *)
```

Сразу же необходимо сделать важную оговорку: поздние версии *компиляторов* языка *Pascal* уже не требуют указывать название *программы*, то есть строку

```
program <имя_программы>;
```

проще говоря, можно опустить. Но это возможно только в том случае, если вся *программа* содержится в одном модуле-файле. Если же *программа*состоит из нескольких самостоятельных кусков - модулей (см. лекцию 13), то каждый из них должен иметь заголовок (program или unit).

Любой из перечисленных необязательных разделов может встречаться в тексте *программы* более одного раза, их общая последовательность также может меняться, но при этом всегда должно выполняться главное правило языка *Pascal*: прежде чем объект будет использован, он должен быть объявлен и описан.

Внешний вид исходного текста программы

Компиляторы языка *Pascal* не различают строчные и прописные буквы, а *пробельные символы* игнорируют, поэтому текст *программы* можно структурировать так, чтобы читать и отлаживать его было наиболее удобно.

Например, *операторы* каждого логически единого блока *программы* стоит записывать с небольшим отступом от левого края экрана, и чем глубже *вложенность блока*, тем шире должны быть отступы перед входящими в него *операторами*. Этому правилу подчиняются все примеры, приводимые в курсе наших лекций[2]. Кроме того, встроенный редактор среды Turbo *Pascal* автоматически выравнивает левые края строк. И еще один полезный совет: для облегчения *отладки программы* не следует записывать на одну строку несколько *операторов*.

Комментарии

Помимо отступов, большие логически замкнутые блоки *программы* удобно разделять строками-комментариями, содержащими информацию о смысле последующего блока. ***Комментарий*** - это строка (или несколько строк) из произвольных символов, заключенная в фигурные скобки:

```
{ комментарий }
```

Другой вариант оформления *комментария*:

```
(* комментарий *)
```

Внутри самого комментария символы } или *) встречаться не должны.

Во время компилирования *программы* комментарии игнорируются. Следовательно, их можно добавлять в любом месте *программы*. Можно даже разорвать *оператор* вставкой комментария. Кроме того, все, что находится после ключевого слова end., завершающего текст *программы*, *компилятор* тоже воспринимает как комментарий.

Директивы компилятора

Строка, начинающаяся символами {$, является не *комментарием*, а директивой *компилятора* - специальной командой, от которой зависит процесс компиляции и выполнения *программы*. Директивы мы будем рассматривать в тех разделах, к которым они относятся "по смыслу".

Например, строка {$I-,Q+} отключает контроль правильности ввода-вывода, но включает *контроль переполнения* при вычислениях.

Идентификаторы

Имена, даваемые программным объектам (*константам*, типам, *переменным*, функциям и процедурам, да и всей *программе* целиком) называются***идентификаторами***. Они могут состоять только из цифр, латинских букв и знака "_" (подчеркивание). Однако цифра не может начинать имя. Идентификаторы могут иметь любую длину, но если у двух имен первые 63 символа совпадают, то такие имена считаются идентичными.

Вы можете давать программным объектам любые имена, но необходимо, чтобы они отличались от *зарезервированных слов*, используемых языком *Pascal*, потому что *компилятор* все равно не примет *переменные* с "чужими" именами.

Приведем список наиболее часто встречающихся *зарезервированных слов*:

```
and             goto                set
array           implementation      shl
begin           in                  shr
case            interface           string
const           label               then
div             mod                 text
do              nil                 to
downto          not                 type
else            of                  unit
end             or                  until
file            pointer             uses
far             procedure           var
```

```
for             program             while
forward         record              with
function        repeat              xor
```

Переменные и типы данных

Переменная - это программный объект, значение которого может изменяться в процессе работы *программы*.

Тип данных - это характеристика диапазона значений, которые могут принимать *переменные*, относящиеся к этому типу данных.

Все используемые в *программе переменные* должны быть описаны в специальном разделе **var** по следующему шаблону:

```
var <имя_переменной_1> [, <имя_переменной_2, _>] : <имя_типа_1>;
        <имя_переменной_3> [, <имя_переменной_4, _>] : <имя_типа_2>;
```

Язык *Pascal* обладает большим набором разнообразных типов данных, однако сейчас мы укажем лишь некоторые из них. Обо всех же типах данных мы поговорим в следующей лекции, там же приведем и различные примеры описания *переменных*

Константы

Константа - это объект, значение которого известно еще до начала работы *программы*.

Константы необходимы для оформления наглядных *программ*, незаменимы при использовании в тексте *программы* многократно повторяемых значений, удобны в случае необходимости изменения этих значений сразу во всей *программе*.

В языке *Pascal* существует три вида констант:

- *неименованные константы* (цифры и числа, символы и строки, множества);
- *именованные нетипизированные константы* ;
- *именованные типизированные константы*.

Неименованные константы

Неименованные константы не имеют имен, и потому их не нужно описывать.

Тип *неименованной константы* определяется автоматически, по умолчанию:

- любая последовательность цифр (возможно, предваряемая знаком "-" или "+" или разбиваемая одной точкой) воспринимается *компилятором* как*неименованная константа* - число (целое или вещественное);
- любая последовательность символов, заключенная в *апострофы*, воспринимается как *неименованная константа* - строка (см. лекцию 5);
- любая последовательность целых чисел[1] либо символов через запятую, обрамленная квадратными скобками, воспринимается как *неименованная константа* - множество (см. лекцию 5).

Кроме того, существуют две специальные константы true и false, относящиеся к *логическому типу* данных.

Примерами использования *неименованных констант* могут послужить следующие *операторы*:

```
int1 := -10;
real2 := 12.075 + x;
char3 := 'z';
string4 := 'abc' + string44;
set5 := [1,3,5] * set55;
boolean6 := true;
```

Нетипизированные константы

Именованные константы, как следует из их названия, должны иметь имя. Стало быть, эти имена необходимо сообщить *компилятору*, то есть описать в специальном разделе const.

Если не указывать тип константы, то по ее внешнему виду *компилятор* сам определит, к какому (*базовому) типу* ее отнести. Любую уже описанную константу можно использовать при объявлении других констант, *переменных* и *типов данных*. Вот несколько примеров описания *нетипизированных именованных констант*:

```
const n = -10;
               m = 1000000000;
               mmm = n*100;
               x = 2.5;
               c = 'z';
               s = 'string';
               b = true;
```

Типизированные константы

Типизированные именованные константы представляют собой *переменные* (!) с начальным значением, которое к моменту старта *программы* уже известно. Следовательно, во-первых, *типизированные константы* нельзя использовать для определения других констант, *типов данных* и *переменных*, а во-вторых, их значения можно изменять в процессе работы *программы*.

Описание *типизированных констант* производится по следующему шаблону:

```
const <имя_константы> : <тип_константы> = <начальное_значение>;
```

Из приведенных ниже примеров видно, как это сделать:

```
const n: integer = -10;
               x: real = 2.5;
               c: char = 'z';
               b: boolean = true;
```

Примеры *типизированных констант* других типов мы будем приводить по мере изучения соответствующих типов данных.

Простейшие операторы

Перейдем теперь к изучению *операторов* - специальных конструкций языка *Pascal*.

Если говорить строго, то ***оператором*** называется (минимальная) структурно законченная единица *программы*.

Важно! Все *операторы* языка *Pascal* должны заканчиваться знаком ";" (точка с запятой), и ни один *оператор* не может разрываться этим знаком. Единственная возможность не ставить после оператора ";" появляется в том случае, когда сразу за этим *оператором* следует ключевое слово end.

К простейшим *операторам* языка *Pascal* относятся:

1. a:= b; - *присваивание переменной* a значения *переменной* b. В правой части *присваивания* может находитьсяпеременная, *константа*, *арифметическое выражение* или вызов функции.
2. ; - пустой *оператор*, который можно вставлять куда угодно, а также вычеркивать откуда угодно, поскольку на целостность *программы* это никак не влияет.
3. ***Операторные скобки***, превращающие несколько *операторов* в один:

```
begin
  <несколько операторов>
end;
```

Везде далее, где в записи конструкций языка *Pascal* мы будем использовать обозначение <один_оператор>, его следует понимать как "один оператор или несколько операторов, заключенные в *операторные скобки* begin - end".

Метки и безусловный переход

Метка помечает какое-либо место в тексте *программы*. *Метками* могут быть числа от 0 до 9999 или *идентификаторы*, которые в этом случае уже нельзя использовать для каких-либо иных нужд. Все *метки* должны быть описаны в специальном разделе label:

```
label <список_всех_меток_через_запятую>;
```

Меткой может быть помечен любой *оператор программы*

```
<метка>: <оператор>;
```

Любая *метка* может встретиться в тексте *программы* только один раз. Используются *метки* только *операторами* безусловного перехода *goto*:

```
goto <метка>;
```

Это означает, что сразу после *оператора goto* будет выполнен не следующий за ним *оператор* (как это происходит в обычном случае), а тот *оператор*, который помечен соответствующей *меткой*.

В принципе, передавать управление можно вперед и назад по тексту *программы*, внутрь составных *операторов* и наружу и т.п. Исключением являются только процедуры и функции (см. лекцию 8): внутрь них и наружу *безусловные переходы* невозможны.

Вообще же использование *безусловных переходов* в структурном и надежном программировании считается "дурным тоном". Поэтому мы настоятельно советуем нашим читателям воздерживаться от употребления *операторов goto*. Язык *Pascal* обладает достаточным количеством структурных конструкций и возможностей, позволяющих достичь хороших результатов надежными средствами.

Ввод и вывод: консоль

Как мы уже говорили, любой *алгоритм* должен быть результативным. В общем случае это означает, что он должен сообщать результат своей работы потребителю: пользователю-человеку или другой *программе* (например, *программе управления принтером*). Мы не будем описывать здесь внутренние автоматические процессы, использующие сигналы непрерывно функционирующих *программ*, а сосредоточим внимание на взаимодействии *программы* и человека, то есть на процессах ввода информации с клавиатуры и вывода ее на экран.

В программировании существует специальное понятие ***консоль***, которое обозначает клавиатуру при вводе и монитор при выводе.

Ввод с консоли

Для того чтобы получить данные, вводимые пользователем вручную (то есть с *консоли*), применяются команды

```
read(<список_ввода>) и readln(<список_ввода>).
```

Первая из этих команд считывает все предложенные ей данные, оставляя курсор в конце последней строки ввода, а вторая - сразу после окончания ввода переводит курсор на начало следующей строки. В остальном же их действия полностью совпадают.

Список ввода - это последовательность имен *переменных*, разделенных запятыми. Например, при помощи команды

```
readln(k,x,c,s);      {k:byte; x:real; c:char; s:string}
```

программа может получить с клавиатуры данные сразу для четырех *переменных*, относящихся к различным типам данных.

Вводимые значения необходимо разделять пробелами, а завершать ввод - нажатием клавиши Enter. Ввод данных заканчивается в тот момент, когда последняя *переменная* из списка ввода получила свое значение. Следовательно, вводя данные при помощи приведенной выше команды, вы можете нажать Enter четыре раза - после каждой из вводимых *переменных*, - либо же только один раз, предварительно введя все четыре *переменные* в одну строчку (обязательно нужно разделить их пробелами).

Типы вводимых значений должны совпадать с типами указанных *переменных*, иначе возникает ошибка. Поэтому нужно внимательно следить за правильностью вводимых данных.

Вообще, вводить с клавиатуры можно только данные *базовых типов* (за исключением логического). Если же *программе* все-таки необходимо получить с *консоли* значение для boolean-величины, придется действовать более хитро: вводить оговоренный символ, а уже на его основе присваивать логической *переменной* соответствующее значение. Например[1]:

```
repeat
  writeln('Согласны ли Вы с этим утверждением? y - да, n - нет');
  readln(c);   {c:char}
  case c of
    'y': b:= true;
    'n': b:= false;
     else writeln('Ошибка!');
   end;
until (c='n')or(c='y');
```

Второе исключение: строки, хотя они и не являются базовым типом, вводить тоже разрешается. Признаком окончания ввода строки является нажатие клавиши Enter, поэтому все следующие за нею *переменные* необходимо вводить с новой строчки.

Вывод на консоль

Сделаем одно важное замечание: ожидая от человека ввода с клавиатуры, не нужно полагать, что он окажется ясновидящим и просто по мерцанию курсора на черном экране догадается, какого типа *переменная* нужна ожидающей *программе*. Старайтесь всегда придерживаться правила: "лысый" ввод недопустим! Перед тем как считывать что-либо с *консоли*, необходимо сообщить пользователю, что именно он должен ввести: смысл вводимой информации, *тип данных*, максимальное и *минимальное допустимые значения* и т.п.

Примером неплохого приглашения служит, скажем, такая строчка:

```
Введите два вещественных числа (0.1<x,y<1000000) - длины катетов.
```

Впрочем, и ее можно улучшить, сообщив пользователю не только допустимый диапазон ввода, но и ожидаемую точность (количество знаков после запятой).

Средства, позволяющие организовать выдачу информации на экран, мы здесь и рассмотрим.

Для того чтобы вывести на экран какое-либо сообщение, воспользуйтесь процедурой `write(< список_вывода >)` или `writeln(<список_вывода>)`.

Первая из них, напечатав на экране все, о чем ее просили, оставит курсор в конце выведенной строки, а вторая переведет его в начало следующей строчки.

Список вывода может состоять из нескольких *переменных*, записанных через запятую; все эти *переменные* должны иметь тип либо базовый[2], либо строчный. Например, `writeln(a,b,c);`

Форматный вывод

Если для вывода информации воспользоваться командой, приведенной в конце предыдущего пункта, то выводимые символы окажутся "слепленными". Чтобы этого не случилось, нужно либо позаботиться о пробелах между выводимыми *переменными*:

```
writeln(a,' ',b,' ',c);
```

либо задать для всех (или хотя бы для некоторых) *переменных формат вывода*:

```
writeln(a:5,b,c:20:5);
```

Первое число после знака ":" обозначает количество позиций, выделяемых под всю *переменную*, а второе - под дробную часть числа. Десятичная точка тоже считается отдельным символом.

Если число длиннее, чем отведенное под него пространство, количество позиций будет автоматически увеличено. Если же выводимое число короче заданного формата, то спереди к нему припишутся несколько пробелов. Таким образом можно производить вывод красивыми ровными столбиками, а также следить за тем, чтобы *переменные* не сливались.

Например, если a = 25, b = 'x', а c = 10.5, то после выполнения команды writeln(a:5,' ',b,c:10:5) на экране или в файле будет записано следующее (подчерки в данном случае служат лишь для визуализации пробелов):

```
_ _ _25_x_ _10.50000
```

Особенно важен формат при выводе вещественных *переменных*. К примеру, если не указать формат, то число 10.5 будет выведено как 1.0500000000E+0001. Такой формат называется записью с плавающей точкой.

Если же задать только общую длину вещественного числа, не указывая длину дробной части, то оно будет занимать на экране заданное количество символов (в случае надобности, спереди будет добавлено соответствующее количество пробелов), но при этом останется в формате плавающей точки. Минимальной длиной для вывода вещественных чисел является 10 (при формате _x.xE+yyyy). Первая позиция зарезервирована под знак "-".

Необходимо помнить, что в случае недостаточной длины вывода число будет автоматически округлено, например (подчерк служит для визуализации пробела):

Оператор форматного вывода	Результат вывода на экран
write (125.2367:10);	_1.3E+0002
write (125.2367:11);	_1.25E+0002
write (125.2367:12);	_1.252E+0002
write (125.2367:13);	_1.2524E+0002
write (125.2367:14);	_1.25237E+0002
write (125.2367:15);	_1.252367E+0002
write (125.2367:16);	_1.2523670E+0002

Пример простейшей программы на языке Pascal

```
program start;
var s: string;
begin
 write('Пожалуйста, введите Ваше имя: ');
 readln(s);
 writeln('Мы рады Вас приветствовать, ',s,'!');
end.
```

Во время работы этой *программы* на экране появится примерно следующее:

```
Пожалуйста, введите Ваше имя: Иван Иваныч
Мы рады Вас приветствовать, Иван Иваныч!
```