# 4. Структура программы на Турбо-Паскале и некоторые простые операторы.

Каждый язык программирования, как и любой естественный язык, подчиняется синтаксису, т.е. законам, по которым составляются осмысленные предложения и по которым можно проверить корректность высказанной мысли. В этой главе мы введем правила синтаксиса языка программирования Турбо-Паскаль и рассмотрим элементарные операторы.

Перед тем, как начать писать непосредственно текст программы, надо сначала хорошо продумать все обозначения, которые будут представлять данные Вашей задачи, определить диапазоны допустимых значений для них, подобрать осмысленные имена переменным и другим элементам согласно синтаксическим правилам.

Текст программы состоит из трех основных структурных частей:

Заголовок,
Блок объявлений (или описаний) и
Тело программы.

Для составления текста используются:
буквы латинского алфавита,
цифры,
знаки пунктуации, пробел,
символы специального назначения.
Рассмотрим пример программы:

```
Program circle;
Const p:=3.14;
Var radius, circle, area : Real;
BEGIN Write(' Радиус = ');
      Read(radius);
      circle := 2 * p * radius;
```

```
    area := p * radius * radius;
    Writeln;
    Writeln(' Длина окружности равняется', circle,',');
    Writeln(' а площадь круга — ', area);
END.
```

Здесь нужно отметить следующие моменты:

1. Любой текст между символами «{» и «}» воспринимается компилятором как комментарии и позволяет пользователю вставлять замечания в код программы. Комментарии – это необязательные атрибуты программы, но желательно в тексте указать хотя бы имя программиста и назначение самой программы. Кроме того, если программа большая, она должна содержать описание основных этапов решения задачи и сложных программных блоков. Потребность в этом особенно чувствуется, если Вам приходится модифицировать или исправлять программу спустя некоторое время.

2. Первая строка «Program circle;» является *заголовком* программы, необязательным атрибутом и невыполняемой инструкцией. *Program* — это специальное, т.н. *зарезервированное, кодовое, ключевое* слово *(КС),* указывающее на то, что данный текст является кодом программы (а не модуля, или данных и т.д.). Здесь же отметим, что заглавные и строчные буквы в тексте программы используются равнозначно.

3. Следующий блок — раздел *объявлений, описания* элементов программы. В данном примере раздел описаний состоит из двух частей: блока объявления постоянных величин, т.н. *констант,* который начинается с ключевого слова *Const.,* и блока объявления *переменных,* начинающегося с ключевого слова *Var.*

Объявление констант подразумевает определение имени и значения величины, а объявление переменных — определение их имени и типа. Так, например, запись:

```
Const p:=3.14;
```

говорит компилятору, что в программе будет использоваться величина *p,* значение которой не меняется в процессе работы программы и равняется 3.14, а запись:

Var radius, circle, area : Real;

указывает, что в программе будут использоваться целочисленные переменные *radius, circle* и *area,* и компилятор выделит для них место в памяти.

4. Дальше идет *тело программы,* которое заключено в т.н. *программные скобки* «BEGIN END.». Тело программы содержит последовательность выполняемых инструкций на языке Турбо-Паскаль, которые называются *операторами.*

5. Каждый оператор, который передает компьютеру команду для выполнения какой-либо операции, завершается символом *точка с запятой* «;». Зарезервированные слова не являются выполняемыми операторами и, следовательно, они необязательно должны завершаться этим символом.

6. Последовательность символов, заключенная в одинарные апострофы, является литеральной *строкой.* Такие инструкции, как

Write(' Радиус = ');

обеспечивают вывод на экран строкового выражения, заключенного в апострофах. Причем оператор «Writeln;» выводит новую пустую строку.

7. Значения переменным можно присваивать с помощью *оператора присвоения «:=* », например, фрагмент

area = p * radius * radius;

вычисляет значение арифметического выражения в правой части, используя текущие значения соответствующих переменных, и присваивает результат переменной *area.*

8. Расклад операторов в тексте программы свободный, т.е. их можно писать произвольно, начиная с любых позиций, новые строки или пробелы могут быть вставлены в любое место, так как они игнорируются компилятором. Их главное назначение заключается в том, чтобы сделать текст программы более читабельным. Однако нельзя разбивать ключевые слова и названия элементов программы — констант, переменных, функций и т.д.

Для имен, обозначений элементов программы используется общий термин – *идентификатор.* Синтаксис Турбо-Паскаля предъявляет следующие требования для формирования идентификаторов:

- Он должен начинаться с латинской буквы;
- Может содержать только латинские буквы, цифры и знак подчеркивания «_»;
- Не должен быть ключевым словом.

Ключевые слова имеют специальные назначения. Их полный список дается в таблице 4.2.

## *4.1. Константы (постоянные величины) и их объявление.*

В программировании часто используются постоянные величины, т.н. *константы,* которые не меняют своего значения в процессе обработки данных. Присвоение осмысленных имен таким константам делает удобным их использование. Эти имена могут быть присоединены к соответствующим значениям в блоке объявления констант. Имена констант должны соблюдать все правила, определенные для формирования идентификаторов. Например, следующее объявление константы:

```
Const days_in_year = 365
```

определяет постоянную Величину *days_in_year,* которая представляет целое число 365. Теперь в теле программы вместо числа 365 можно использовать идентификатор *days in year*, что придаст больше читабельности тексту. Общая форма объявления констант имеет вид:

```
Const Идентификатор-константы = Значение;
```

«Идентификатор-константы» определяет имя величины, а «Значение» — его содержание. Нельзя объявлять константу без присвоения значения.

Другое преимущество объявления констант можно увидеть на примере:

Const City_People = 1 750 000;

Эта величина определяет население города. Однако со временем она может измениться, и если бы она не была описана, как постоянная величина, нам пришлось бы просмотреть весь текст программы и везде изменить вхождение числа «I 750 000». А сейчас значение константы определяется в одном месте программы и чтобы ее изменить, нужно сделать только одну корректировку в ее объявлении.

Хотя не существует строгого правила насчет месторасположения блоков объявления, описание констант почти всегда размещают перед блоком описания переменных. Нет никаких ограничений на количество описываемых констант, или блоков описания констант.

И, наконец, отметим, что если идентификатор объявлен как константа, то дальнейшее изменение его значения в теле программы недопустимо и вызывает ошибку компилятора.

## *4.2. Переменные.*

Переменные используются для обозначения величин, значения которых обрабатываются компьютерной программой. Например, если программа считывает последовательность чисел и суммирует их, то мы должны иметь переменные для представления вводимого числа и суммы чисел.

Чтобы различать переменные, обозначающие различные величины, им приписываются имена, которые позволят отличать одну переменную от остальных. Эта договоренность аналогична элементарной алгебре, когда мы говорим: «Пусть S будет пройденным расстоянием, а V и T — скоростью и временем, соответственно». Здесь мы использовали идентификаторы для представления значений расстояния, скорости и времени. Имена переменных также должны удовлетворять правилам формирования идентификаторов.

Примеры допустимых и недопустимых идентификаторов приведены в таблице 4.1.

Таблица 4.1.

| Допустимые | Недопустимые |
|---|---|
| length | days-in-year |
| days-in-year | 1data |
| Data Setl | int |
| Profit95 | first. val |
| first_one first_1 | "throw" |

Идентификаторы должны быть подобраны так, чтобы отражать смысл соответствующей переменной. Естественно, в программе легче использовать идентификаторы с именем в одну букву, но это затруднит дальнейшую модификацию и корректировку текста.

Как уже было отмечено, синтаксис Турбо-Паскаля некоторым символам придает специальное назначение. Они называются ключевыми словами и не должны использоваться для других целей. Мы уже столкнулись с некоторыми из них: *Program, Begin, End.* В таблице 4.2 приводится полный список ключевых слов.

Таблица 4.2.

| | | |
|---|---|---|
| Absolute | And | Array |
| Begin | Case | Const |
| Div | Do | Downto |
| Else | End | External |
| File | For | Forward |
| Function | Goto | If |
| Implementation | In | Inline |
| Interface | Interrupt | Label |
| Mod | Nil | Not |
| Of | Or | Packed |
| Procedure | Program | Record |
| Repeat | Set | Shl |

| Shr | String | Then |
|---|---|---|
| To | Type | Unit |
| Until | Uses | Var |
| While | With | Xor |

Некоторый принятый стандарт программирования советует избегать также использования имен встроенных функций, процедур, постоянных и переменных, объявленных в модулях System, Crt, Graph, Printer, DOS. Такое объявление может изменить функциональное назначение этих элементов.

Еще раз отметим, что регистр буквы не имеет значения, т.е. как строчные, так и заглавные буквы трактуются компилятором идентично. Для удобства в приведенных программных фрагментах—примерах ключевые слова, а также имена встроенных элементов (процедур, функций, переменных и др.) будут записаны с заглавной буквы, а элементы, определенные пользователем — со строчной.

В Турбо-Паскале, как во многих других языках программирования, все переменные, до их использования в теле программы, предварительно должны быть объявлены. Объявление переменных служит следующим целям:

- Объявление определяет имя и *тип* переменных. Тип позволяет компилятору правильно интерпретировать операторы. Например, на машинном языке центрального процессора сложение двух целых и двух действительных чисел осуществляется с помощью различных операторов сложения. Следовательно, компилятор должен знать тип переменных, чтобы правильно генерировать инструкцию для вычисления результата.
- Тип переменных определет диапазон их допустимых значений, объявление типа позволяет компилятору распределить память для хранения значений, присоединенных с идентификаторами и присвоить адрес, по которому программа будет обращаться к переменной при генерации кода.

На этом этапе рассмотрим только три типа переменных, а именно

Integer, Real и Char:

| | |
|---|---|
| **Integer -** | Переменные могут представлять положительные и отрицательные целые числа. Диапазон значений определяется количеством байтов, отводимых компилятором для хранения соответствующих переменных и зависит от компьютерной системы. На персональных компьютерах большинство компиляторов для каждой целой переменной отводит 2 байта оперативной памяти, что позволяет диапазон значений от -32_768 до +32_767. (На Рабочих станциях для этого отводятся 4 байта, что позволяет диапазон значений от -2_147_483_648 до +2_147_483_647). Важно отметить, что целые числа представляются в компьютерной памяти точно (не приближенно). |
| **Real -** | Переменные могут представлять любые действительные величины, что включает как целые, так и дробные числа, в записи которых используется десятичная точка. Точность и диапазон значений зависит от компьютерной системы. Обычно используется 6 байтов памяти, что дает точность в шесть значащих цифр после точки, и диапазон $10^{-29}$ — $10^{29}$. Важно отметить, что действительные величины представлены в памяти только приближенно. |
| **Char -** | (От слова Character, символ ) Переменные могут представлять один символ — букву, цифру или знак препинания. Они занимают 1 байт памяти, давая диапазон 256 различных символов. Обычно представление символа в битах соответствует стандартной ASCII - таблице. |

Примеры значений переменных различных типов:

| тип | Допустимые значения |
|---|---|
| Integer | 12 –256 0 5645 |
| Real | 16.315 –0.67 31.567 |
| Char | '+' 'A' 'a' '*' '7' '!' |

Типичные примеры объявления переменных:

```
Var
    i, j, count: Integer;
    sum, product: Real;
    ch: Char;
```

Этот блок описания объявляет, что переменные *i, j* и count являются целочисленными, переменные sum и product – действительными и переменная *ch* – символьной.

Общий вид объявления имеет форму:

Var Список – переменных: Тип;

где *Список – переменных* состоит из идентификаторов переменных, разделенных запятыми, а *Тип* – определяет их тип.

## *4.3. Арифметические выражения и оператор присвоения.*

В Турбо-Паскале определены следующие арифметические операции:

| | | |
|---|---|---|
| + | - | Сложение |
| - | - | Вычитание |
| * | - | Умножение |
| / | - | Деление |
| Div | - | Целочисленное деление |
| Mod | - | Остаток деления |

Приведем примеры записи сложных математических формул на Турбо-Паскале:

$$y = \frac{a+b}{c^2 + d^2}$$ y:=(a+b)/(c*c+d*d)

$$f = \frac{\sin(x) + \cos(y)}{3x^2 + 5y^2}$$ f:=(Sin(x)+Cos(y))/(3*x*x+5*y*y)

$$y = a^2 + b^2 - \frac{\sqrt{c} + 5\sin x}{\cos^2 x - 1}$$

y:=Sqr(a) + Sqr(b) - (Sqrt(c)+Sin(x)) / (Sqr(cos(x))-1);

$f = 5a^2 + 7abc + 9b^2$ f:=5*(Sqr(a) + 7*a*b*c + 9*Sqr(b);

$y = e^{-2x} + tg(x^2 + 1)$ y:=Exp(-2*x) + Tg(Sqr(x)+1);

(Sin, Cos, Tg, Exp, Ln, Sqr и Sqrt – стандартные функции, о них см. ниже).

Главным оператором в Турбо-Паскале для реализации вычислений и определения значений переменных является оператор *присвоения.* Например, оператор

average:= (a+b)/2;

присваивает переменной average половину суммы переменных *a* и *b*. Общая форма оператора присвоения имеет вид:

Результат := Выражение;

«Выражение» вычисляется и затем присваивается переменной «Результат». Очень важно отметить, что типы выражения и результата должны либо совпадать, либо быть совместимыми (см. гл. 10).

Выражение может быть единственной переменной, или константой, или комбинацией переменных и констант, связанных друг с другом арифметическими операторами и круглыми скобками для определения порядка выполнения операций.

Приведем несколько примеров вызова оператора присвоения в теле программы:

```
i:=3;
sum := 0;
```

```
perimeter:=2.0*(length + breadth);
ratio:= (a+b)/(c + d);
```

Тип операнд арифметических операторов очень важен. Соблюдаются следующие правила:

• Операнды операторов Div и Mod, а также результат могут быть только типа Integer.

• Результат выполнения оператора «/» может быть только типа Real.

• Для операторов «+», «-» и «*» выполняются следующие правила:

▫ а) если операнды целые числа, то результат также будет целочисленным (хотя и может быть присвоен действительной переменной, предварительно преобразованной в действительное представление),

▫ 6) если один или оба операнда действительные числа, то результат также будет действительным (здесь, при присвоении этого результата целочисленной переменной, возникнет логическая ошибка).

### 4.3.1. Приоритет операторов.

При вычислении арифметического выражения может возникнуть вопрос последовательности выполнения операторов. Например, как должно быть интерпретировано выражение:

```
a=b*c
```

как – (a + b) * c или как – a + (b * c).

В Турбо-Паскале эта проблема решается с помощью присвоения операторам приоритетов, так что в последовательности сначала выполняются операторы с высоким приоритетом, а затем — с более низким, а операторы с одинаковым приоритетом выполняются по порядку расположения слева направо. Ниже приводятся приоритеты рассмотренных до сих пор операторов по убыванию:

```
( )
Div, M od
*/
+ -
: =
```

Таким образом, выражение

a+b*c

будет вычисляться как запись a + (b * c), так как «*» имеет более высокий приоритет, чем «+». Если бы надо было сначала выполнить сложение, а затем умножение, то тогда нужно было использовать круглые скобки:

(a + b) * c

## 4.4. Составной оператор.

Последовательность нескольких операторов, заключенных в т.н. программные скобки «Begin End», называется *составным оператором.* Компилятором такая группа операторов воспринимается как один оператор. Он может применяться в случаях, когда синтаксис Турбо-Паскаля требует наличия только одного оператора, а для выполнения поставленной задачи необходимо использовать последовательность нескольких операторов.

Пример составного оператора:

```
Begin
        Write(' Введите целое число: ');
        Read(n);
        Writeln(' Квадрат числа ', n,' равняется', n*n,'.');
End;
```

## 4.5. Метки и оператор перехода Goto.

Синтаксис Турбо-Паскаля допускает помечать каждый оператор т.н. *меткой (Label).* Формат использования таков — перед оператором ставится метка и знак двоеточия «:». Меткой могут выступать любой идентификатор, согласно правилам их формирования, а также число от 0 до 9999. Чтобы в тексте программы можно было использовать конкретную метку, она должна быть предварительно объявлена в блоке описания в разделе *Label,* например,

```
Label 1, metka Too, metka_l;
```

Оператор перехода «Goto metka_1;» позволяет с любого места тела программы обратиться к оператору, помеченному меткой *metka_1*. Этот оператор весьма удобен и прост, но его выполнение связано с поиском, что увеличивает время выполнения программы в целом, и поэтому предпочтительнее избегать его, используя другие возможности и структуры языка

Пример применения оператора перехода:

```
Label metka_1;
Var n: Integer,
Begin
        Writeln(' Эта программа ничего не будет делать!!!'),
        Goto metka_1,
{ Следующая часть до ********** не выполняется из-за оператора Goto}
Write(' Введите целое число:  '),
 Read(n),
Writeln (' Квадрат числа ', n,' равняется ', n*n', n', '},
{**************}
metka _1: End
```

## *4.6. Системные процедуры и функции.*

В Турбо-Паскале определены стандартные *процедуры* и *функции (подпрограммы).* Это уже реализованные программные блоки, которые могут быть вызваны в главной программе по имени. Основное различие между процедурой и функцией заключается в том, что функция является носителем значения величины, вычисленной в процессе ее работы, а процедура просто выполняет последовательность операций. В арифметических или логических выражениях идентификатор функции может стоять на месте *любой* переменной. Идентификатор же процедуры не является носителем какого-либо значения, а обращение к ней осуществляется с помощью т. н. *оператора вызова процедуры.*

Рассмотрим некоторые стандартные процедуры и функции.

Самая простая стандартная функция, это — *Pi,* которая выдает значение числа π. Эта функция не имеет ни входных, ни выходных параметров, что обычно встречается весьма редко.

## *Некоторые стандартные функции:*

Тригонометрические функции

Sin(x):Real

Вычисляет значение синуса угла *x (x* в радианах)

*Cos (x): Real*

Вычисляет значение косинуса угла *x (x в* радианах)

Atan(x):Real

Вычисляет значение арктангенса.

Различные функции

Exp(x):Real

Вычисляет значение экспоненциальной функции от *x.*

Ln(x):Real

Вычисляет значение натурального логарифма с основой «е» от *x.*

Abs(x)

Вычисляет абсолютное значение от *x* (тип зависит от типа аргумента).

Sqr(x)

Вычисляет квадрат от *x* (тип зависит от типа аргумента).

Sqrt(x):Real

Вычисляет значение квадратного корня от x.

Trunc(x):Integer

Выдает целую часть числа *x.*

Fract(x):Real

Выдает дробную часть числа *x.*

Round(x):Integer

Округляет число *x.*

Int(x):Integer

Выдает целую часть числа *x.*

Lo (x):Byte

Возвращает первый байт аргумента.

Hi (x):Byte

Возвращает первый и последний байты аргумента.

Swap(x):Тип — аргумента

Меняет местами первый и последний байты аргумента. Тип функции определяется типом аргумента.

Во всех примерах *x* обозначает формальный параметр, который используется для представления аргумента функции и требуется для ее описания. При обращении к функции формальный параметр заменяется реальным, фактическим значением. Это может быть константа, переменная или арифметическое выражение, которое, в свою очередь, может включать обращение к другой функции. Кроме тех случаев, когда тип функции зависит от типа аргумента, в их описании указывается тип функции.

Функция вызывается с помощью имени, следуемого фактическим параметром в круглых скобках, например, Round(x+5*y). Функции могут использоваться в любом месте программы, где может стоять обычная переменная. Приведем несколько примеров вызова каждой функции:

```
Var i,j,k:Integer;
    x,y,z:Real;
    ...
y:=sin(x);
i:=(Round(x*y)) Mod (Round(x/y));
j:=Round( Abs(Sqrt(Sqr(x)-Sqr(y))));
...
```

## *Некоторые стандартные процедуры:*

Exit;

Обеспечивает выход из процедуры с места ее вызова.

Halt;

Прерывает выполнение главной программы.

Inc(n);

Увеличивает значение переменной *n* на единицу.

Inc(n,m);

Увеличивает значение переменной
*n* на *m*.

Dec(n);

Уменьшает значение переменной *n* на единицу.

Dec(n,m);

Уменьшает значение переменной *n* на *m*.

Read(x,y);

Считывает значения переменных *x* и *y* с клавиатуры.

Write(a,b);

Выводит значения переменных *a* и *b* на экран.

Writeln;

Выводит на экран пустую строку.

Writeln(a,b);

Выводит значения переменных *a* и *b* на экран с переводом строки.

Подпрограммы могут не иметь входных параметров (как например, функция Pi, процедуры: Exit, Halt, Writeln). Но чаще подпрограмме передаются исходные данные в виде аргументов. Кроме того, подпрограмма может возвращать значения величин, вычисленных или определенных при ее выполнении также в виде ее аргументов, как, например, процедура «Read» или «Inc». При этом, во время обращения к подпрограмме формального различия между входными и выходными параметрами-аргументами нет, программист сам должен знать, какое назначение имеет каждый параметр.

## *4.7. Общая структура программы, стиль программирования и запуск программы на выполнение.*

Структуру Турбо-Паскалевской программы на этом этапе можно представить в общем виде:

```
Program Имя-программы;
{ Комментарии, имя программиста, описание программы }
Блок объявления констант, переменных
```

```
Begin
    Тело программы: Выполняемые операторы
End.
```

Распределение операторов по тексту программы можно осуществить произвольно, но существуют определенные неписаные правила для оформления кода, которые помогают сделать его более читабельным как с чисто эстетической точки зрения, так в аспекте читабельности, понимания и интерпретации.

Каждый оператор лучше разместить на отдельной строке, а если не помещается, следующую строку начать с некоторым отступом.

Программные скобки (открывающую *Begin* и закрывающую *End)* лучше расположить строго один под другим, при этом во вложенных структурах использовать различный отступ, чтобы легко выделить самостоятельные блоки.

Имена идентификаторов переменных лучше подбирать не произвольно, а по смыслу и значению, которое представляет соответствующая переменная — это считается хорошим стилем программирования.

Рассмотрим приведенный выше фрагмент программы в другой форме записи и сравним, какая из них легче поддается редактированию.

```
Program
c;
Const p:=
3,14;
Var r,
c, a:
Real; BEGIN Write
(' Радиус = '); Read( r);
c:=2*p *r;a := p * r* r; Writeln;
Writeln(' Длина окружности равняется' , c,',');
Writeln(' а площадь круга — ', a) ;
END.
```

Здесь же отметим, что этот текст набирается в каком-либо редакторе, среда Турбо-Паскаля дает эту возможность. Затем в этой же среде можно откомпилировать программу и запустить на выполнение. Инструкции, как это реализовать, можно найти в любом справочнике по Турбо-Паскалю.