Язык программирования *Object Pascal*

Паскаль – это язык профессионального программирования, который назван в честь французского математика и философа Блеза Паскаля (1623-1662) и разработан в 1968-1971 гг. Никлаусом Виртом. Первоначально был предназначен для обучения, но вскоре стал использоваться для разработки программных средств в профессиональном программировании.

Язык программирования *Pascal* отличается простотой для обучения; отражает фундаментальные идеи алгоритмов в легко воспринимаемой форме, что предоставляет программисту средства, помогающие проектировать программы; позволяет четко реализовать идеи структурного программирования и структурной организации данных; использует простые и гибкие структуры управления: ветвления, циклы; надежен для разрабатываемых программ.

3.1.1 Структура программ на языке Паскаль

Программы на языке Паскаль имеют блочную структуру.

1. Блок типа *PROGRAM* имеет имя, состоящее только из латинских букв и цифр. Его присутствие не обязательно, но рекомендуется для быстрого распознавания нужной программы среди других листингов.

2. Программный блок, состоящий в общем случае из семи разделов: раздел описания модулей (**`uses`**); раздел описания меток (**`label`**); раздел описания констант (**`const`**); раздел описания типов данных (**`type`**); раздел описания переменных (**`var`**);

раздел описания процедур и функций; раздел описания операторов.

Заголовок программы начинается со слова **`Program`** (программа), за которым следует произвольное имя, придуманное программистом: **`Program <имя программы>.`**

Раздел описания переменных начинается со слова **`Var`** (*variables* – переменные), за которым идет список имен переменных через запятую. Тип указывается после двоеточия.

В стандарте языка Паскаль существует два числовых типа величин: вещественный и целый. Слово *integer* обозначает целый

тип (является идентификатором целого типа). Вещественный тип обозначается, словом *real*. Например, раздел описания переменных может быть таким:

```
Var a, b: integer;
 c, d: real.
```

Идентификаторы переменных составляются из латинских букв и цифр; первым символом обязательно должна быть буква.

Раздел операторов – основная часть программы. Начало и конец раздела операторов программы отмечаются служебными словами *begin* (начало) и *end* (конец). В самом конце программы ставится точка:

```
begin
< операторы >
end.
```

3.1.2 Простые типы данных

Тип определяет множество значений, которые могут принимать элементы программы, и совокупность операций, допустимых над этими значениями.

Например, значения –34 и 67 относятся к целочисленному типу и их можно умножать, складывать, делить и выполнять с ними другие арифметические операции, а значения **`abed`** и **`sdfhi23`** относятся к строковому типу, и их можно сцеплять (складывать), но нельзя делить или вычитать.

Типы данных можно разделить на следующие группы: простые; структурные; указатели; процедурные; вариантные.

Простые и структурные типы включают в свой состав другие типы, например целочисленные или массивы. Приводимое деление на типы в некоторой мере условно – иногда указатели причисляют к простым типам, а строки, которые относятся к структурным типам, выделяют в отдельный тип.

Важное значение имеет понятие «совместимости типов», которое означает, что типы равны друг другу или один из них может быть автоматически преобразован в другой. Совместимыми, например, являются вещественный и целочисленный типы, так как целое число автоматически преобразовывается в вещественное, но не наоборот.

Простые типы не содержат в себе других типов, и данные этих типов могут одновременно содержать одно значение.

К простым относятся следующие типы: целочисленные, литерные (символьные), логические (булевы), вещественные.

Все типы, кроме вещественного, являются порядковыми, то есть значения каждого из этих типов образуют упорядоченную конечную последовательность. Номера соседних значений в ней отличаются на единицу.

Целочисленные типы включают целые числа. Наиболее часто используется тип ***integer***, допускающий значения в диапазоне от ***-2147483648 до 2147483647.***

Значениями литерного типа являются элементы из набора литер, то есть отдельные символы. В *Object Pascal* определен литерный тип ***char***, который занимает один байт, а для кодирования символов используется код американского национального института стандартов *ANSI* (*American National Standards Institute*).

В *Object Pascal* к логическому относится тип ***Boolean.*** Этот тип представлен двумя возможными значениями: ***True*** (истина) и ***False*** (ложь). Для представления логического значения требуется один байт памяти.

Интервальные типы описываются путем задания двух констант, определяющих границы допустимых для данных типов значений. Эти границы и определяют интервал (диапазон) значений. Компилятор для каждой операции с переменной интервального типа, если это возможно, проверяет, находится ли значение переменной внутри установленного для нее интервала, и в случае его выхода за границы выдает сообщение об ошибке. Во время выполнения программы при выходе значения интервального типа за границы интервала сообщение об ошибке не выдается, однако значение переменной будет неверным. Интервал можно задать только для порядкового типа, то есть для любого простого типа, кроме вещественного. Обе константы, определяющие интервал, должны принадлежать одному из простых типов. Значение первой константы должно быть меньше значения второй. Формат описания интервального типа:

Type <Имя> = <Константа1> .. <Константа2>.

Вещественные (действительные) типы включают в себя вещественные числа. Наиболее часто используется тип *Real*, обеспечивающий точность 15–16 цифр мантиссы.

Запись вещественных чисел возможна в форме с фиксированной и в форме с плавающей точкой. Вещественные числа с фиксированной точкой записываются по обычным правилам арифметики. Целая часть отделяется от дробной десятичной точкой. Перед числом может указываться знак + или –. Если знак отсутствует, то число считается положительным. Для записи вещественных чисел с плавающей точкой указывается порядок числа со знаком, отделенным от мантиссы знаком E (или e).

3.1.3 Структурные типы данных

Структурные типы имеют в своей основе один или более других типов, в том числе и структурных. К структурным типам относятся: строки, записи, массивы, файлы, множества, классы.

Строки обеспечивает тип ***string***, который представляет строку с максимальной длиной около 2 x 1031 символов. Символы строки кодируются в коде *ANSI*. Так как строки фактически являются массивами символов, то для обращения к отдельному символу строки можно указать название строковой переменной и номер (позицию) этого символа в квадратных скобках, например ***strName [i].***

Массивом называется упорядоченная индексированная совокупность однотипных элементов, имеющих общее имя. Элементами массива могут быть данные различных типов, включая структурированные. Каждый элемент массива однозначно определяется именем массива и индексом (номером этого элемента в массиве) или индексами, если массив многомерный. Для обращения к отдельному элементу массива указываются имя этого массива и номер (номера) элемента, заключенный в квадратные скобки, например: ***Marpl[3, 35].***

Количество индексных позиций определяет мерность массива (одномерный, двумерный и т. д.), при этом мерность массива не ограничивается. В математике аналогом одномерного массива является вектор, а двумерного массива – матрица. Индексы элементов массива должны принадлежать порядковому типу. Разные индексы одного и того же массива могут иметь различные типы. Наиболее часто типом индекса является целочисленный тип.

Множество представляет собой совокупность элементов, выбранных из предопределенного набора значений. Все элементы

множества принадлежат одному порядковому типу, число элементов в множестве не может превышать 256. Формат описания множественного типа: **`Set of <Тип элементов>.`**

Переменная множественного типа может содержать любое количество элементов своего множества – от нуля до максимального. Значения множественного типа заключаются в квадратные скобки. Пустое множество обозначается как **`[ ].`**

При выполнении программы осуществляется обработка данных, в ходе которой с помощью выражений вычисляются и используются различные значения. Выражение представляет собой конструкцию, определяющую состав данных, операции и порядок выполнения операций над данными. Выражение состоит из операндов, знаков операций, круглых скобок.

Тип значения выражения определяется типом операндов и составом выполняемых операций.

Операнды представляют собой данные, над которыми выполняются действия. В качестве операндов могут использоваться константы (литералы), переменные, элементы массивов и обращения к функциям.

Операции определяют действия, которые выполняются над операндами.

Операции могут быть унарными и бинарными. Унарная операция относится к одному операнду, и ее знак записывается перед операндом, например *-x*.

Бинарная операция выражает отношение между двумя операндами, и знак ее записывается между операндами, например *X+Y*.

Круглые скобки используются для изменения порядка выполнения операций.

В зависимости от типов операций и операндов выражения могут быть: арифметическими, логическими и строковыми.

3.1.4 Операторы

Операторы представляют собой законченные предложения языка, которые выполняют некоторые действия над данными.

Операторы *Pascal* можно разделить на две группы: простые, структурированные.

Операторы разделяются точкой с запятой. Точка с запятой является разделителем операторов, и ее отсутствие между операторами является ошибкой.

Наличие между операторами нескольких точек с запятой не является ошибкой, так как они обозначают пустые операторы. Отметим, что лишняя точка с запятой в разделе описаний и объявлений является синтаксической ошибкой.

Точка с запятой может не ставиться после слова **begin** и перед словом **end**, так как они являются операторными скобками, а не операторами.

В условных операторах и операторах выбора точка с запятой не ставится после слова **then** и перед словом **else**. Отметим, что в операторе цикла с параметром наличие точки с запятой сразу после слова **do** синтаксической ошибкой не является, но в этом случае тело цикла будет содержать только пустой оператор.

Простыми называются операторы, не содержащие в себе других операторов.

К ним относятся: оператор присваивания, оператор перехода, пустой оператор, оператор вызова процедуры.

Оператор присваивания является основным оператором языка. Он предписывает вычислить выражение, заданное в его правой части, и присвоить результат переменной, имя которой расположено в левой части оператора. Переменная и выражение должны иметь совместимый тип, например вещественный и целочисленный, но не наоборот. Допустимо присваивание любого типа данных, кроме файлового. Формат оператора присваивания:

<Имя переменной>:= <Выражение>.

Вместо имени переменной можно указывать элемент массива или поле записи. Отметим, что знак присваивания **:=** отличается от знака равенства **=** и имеет другой смысл. Он означает, что сначала вычисляется значение выражения и затем оно присваивается указанной переменной. Поэтому при условии, что x является числовой переменной и имеет определенное значение, будет допустимой и правильной следующая конструкция: **x:= x + 1**.

Операторы присваивания:

```
Var x, y: real; n: integer; stroka: string;
n:= 17 * n-1;
stroke:= 'Дата '+DateToStr(Date);
```

x:= -12.3 * sin(pi / 4); y:= 23.789E+3.

Оператор перехода предназначен для изменения естественного порядка выполнения операторов программы. Он используется в случаях, когда после выполнения некоторого оператора требуется выполнить не следующий по порядку, а какой-либо другой, помеченный меткой оператор. **Метка,** стоящая перед оператором, отделяется от него двоеточием. Меткой может быть идентификатор или целое число без знака в диапазоне 0–9999, и все метки должны быть предварительно объявлены в разделе объявления меток того блока процедуры, функции или программы в котором эти метки используются. Формат оператора перехода: ***goto <Метка>.***

Пример использование оператора перехода:

Label m1; goto m1; m1: <Оператор>.

Передавать управление с помощью оператора перехода можно на операторы, расположенные в тексте программы выше или ниже оператора перехода.

Запрещается передавать управление операторам, находящимся внутри структурированных операторов, а также операторам, находящимся в других блоках (процедурах, функциях).

Пустой оператор представляет собой точку с запятой и может быть расположен в любом месте программы, где допускается наличие оператора. Как и другие операторы, пустой оператор может быть помечен меткой. Пустой оператор не выполняет никаких действий и может быть использован для передачи управления в конец цикла или составного оператора.

Оператор вызова процедуры служит для активизации стандартной или предварительно описанной пользователем процедуры и представляет собой имя этой процедуры со списком передаваемых ей параметров.

Структурированные операторы представляют собой конструкции, построенные по определенным правилам из других операторов. К структурированным операторам относятся: составной оператор, операторы цикла (повтора), условный оператор, оператор доступа, операторы выбора.

Составной оператор представляет собой группу из произвольного числа любых операторов, отделенных друг от друга точкой с запятой, и ограниченную операторными скобками ***begin*** и ***end***. Формат составного оператора:

```
begin <Оператор1>; … ; <ОператорN>; end.
```

Независимо от числа входящих в него операторов, составной оператор воспринимается как единое целое и может располагаться в любом месте программы, где допускается наличие оператора. Наиболее часто составной оператор используется в условных операторах и операторах цикла. Составные операторы могут вкладываться друг в друга.

Пример составного оператора:

```
begin
   Beep;
   Edit1.Text:= 'Ошибка';
   Exit;
end.
```

Условный оператор обеспечивает выполнение или невыполнение некоторых операторов в зависимости от соблюдения определенных условий. Условный оператор в общем случае предназначен для организации разветвления программы на два направления и имеет формат:

```
if <Условие> then <Оператор1>
             else <Оператор2>.
```

Для организации разветвлений на три направления и более можно использовать несколько условных операторов, вложенных друг в друга. При этом каждое **else** соответствует тому **then**, которое непосредственно ему предшествует. Из-за возможных ошибок следует избегать большой вложенности условных операторов друг в друга.

Условные операторы:

```
if x > 0 then x:= x + 1
         else  Begin
               x:= 0;
               if q = 0 then a:= 1;
             end.
```

Оператор выбора является обобщением условного оператора и позволяет сделать выбор из произвольного числа имеющихся вариантов, то есть организовать разветвления на произвольное число направлений. Этот оператор состоит из выражения, называемого селектором, списка вариантов и необязательной ветви **else**, имеющей тот же смысл, что и в условном операторе.

Формат оператора выбора:

```
Case Number of
 1..2: WriteLn('Малый номер');
 3..6: WriteLn('Нормальный,средний номер');
else
 WriteLn ('Большой номер').
```

Операторы цикла используются для организации циклов (повторов). Цикл представляет собой последовательность операторов, которая может выполняться более одного раза. Группу повторяемых операторов называют телом цикла. Для построения цикла в принципе можно использовать ранее рассмотренные условный оператор и оператор перехода. Однако в большинстве случаев удобно использовать операторы цикла.

Всего имеется три вида операторов цикла: с параметром, с предусловием, с постусловием.

Обычно, если количество повторов известно заранее, то применяется оператор цикла с параметром, в противном случае – операторы с пост- или предусловием (чаще используется оператор с предусловием).

Выполнение оператора цикла любого вида может быть прервано с помощью оператора перехода ***goto*** или предназначенной для этих целей процедуры без параметров ***Break***, которая передает управление на следующий за оператором цикла оператор.

С помощью процедуры без параметров ***continue*** можно задать досрочное завершение очередного повторения тела цикла, что равносильно передаче управления в конец тела цикла.

Операторы циклов могут быть вложенными друг в друга.

Оператор цикла с параметром имеет два следующих формата:

```
for <Парам>:= <Выраж1> to <Выраж2> do <Опер>;
```

и

```
for <Парам>:=<Выраж1> downto <Выраж2> do <Опер>.
```

То есть значение параметра последовательно увеличивается (***for ... to***) или уменьшается (***for ... downto***) на единицу при каждом повторении цикла.

Оператор цикла с предусловием целесообразно использовать в случаях, когда число повторений тела цикла заранее неизвестно и тело цикла может не выполняться. Во многом этот оператор аналогичен оператору *repeat...until*, но проверка условия выполняется в начале оператора.

Формат оператора цикла с предусловием:

```
while <Условие> do <Оператор>.
```

Оператор тела цикла выполняется до тех пор, пока логическое выражение не примет значение **False**, то есть, в отличие от цикла с постусловием, цикл выполняется при значении логического выражения **True**.

Оператор цикла с предусловием:

```
var x: integer; sum: real;
m: array[1 .. 10] of real;
begin
x: = 1; sum:= 0;
while x <= 10 do
begin
   sum:= sum + m[x];
   x:= x + 1;
end;
end.
```

Если перед первым выполнением цикла условие не выполняется, значение логического выражения равно **False**, то тело цикла не выполняется ни разу, и происходит переход на следующий за оператором цикла оператор.

Оператор доступа **with** служит для удобной и быстрой работы с составными частями объектов, в том числе с полями записей. Оператор доступа имеет следующий основной формат:

```
with <Имя объекта> do <Оператор>.
```

В операторе, расположенном после слова **do**, для обращения к составной части объекта можно не указывать имя этого объекта, которое уже задано после слова **with**.

Использование оператора доступа:

– составные имена пишутся полностью

```
Form1.Canvas.Pen.Color:= clRed;
Form1.Canvas.Pen.Width:= 5;
Form1.Canvas.Rectangle(10, 10, 100, 100);
```

– или используется оператор доступа

```
with Form1.Canvas do
begin
Pen.Color:= clRed; Pen.Width:= 5;
Rectangle(10, 10, 100, 100);
end.
```

3.1.5 Подпрограммы

Подпрограмма представляет собой группу операторов, логически законченную и специальным образом оформленную. Подпрограмму можно вызывать неограниченное число раз из различных частей программы. Использование подпрограмм позволяет улучшить структурированность программы и сократить ее размер.

По структуре подпрограмма почти полностью аналогична программе и содержит заголовок и блок, однако в блоке подпрограммы отсутствует раздел подключения модулей. Кроме того, заголовок подпрограммы по своему оформлению отличается от заголовка программы.

Любая подпрограмма должна быть предварительно описана, после чего допускается ее вызов. При описании подпрограммы указывается ее имя, список параметров и выполняемые подпрограммой действия. При вызове подпрограммы указываются имя подпрограммы и список аргументов, передаваемых подпрограмме для работы.

Подпрограммы делятся на процедуры и функции, которые имеют между собой много общего. Основное различие между ними заключается в том, что функция может возвращать под своим именем в качестве результата значения и соответственно может использоваться в качестве операнда выражения.

```
function MyFunc(i: integer): integer;
begin
   Result:= i * 2;
end.
```

С подпрограммой взаимодействие осуществляется по управлению и по данным. Взаимодействие по управлению заключается в передаче управления из программы в подпрограмму и организации возврата в программу.

```
procedure Udvoenie(st: string; var Rr:real);
var r: real;
begin
//полученную строку преобразуем в число:
r:= StrToFloat(st);
// удвоим его
Rr:= r * 2;
//теперь выведем результат в сообщении:
```

```
ShowMessage(FloatToStr(Rr));
end.
```

Взаимодействие по данным заключается в передаче подпрограмме данных, над которыми она выполняет определенные действия. Этот вид взаимодействия может осуществляться с помощью параметров и аргументов.

Параметры являются элементами подпрограммы и используются при описании алгоритма, выполняемого подпрограммой.

Аргументы являются элементами вызывающей программы. Они замещают параметры при вызове подпрограммы.

Для использования процедур и функций их следует определить в классе формы, а затем расположить в точке кода программы, когда возникает потребность в использовании подпрограммы, например `Udvoenie(2; RrLok);` `MyFunc(3).`

Для прекращения работы подпрограммы можно использовать процедуру `Exit`, которая прерывает выполнение операторов подпрограммы и возвращает управление вызывающей программе.

3.1.6 Особенности объектно-ориентированного программирования

Язык *Object Pascal* является объектно-ориентированным расширением языка *Pascal* и реализует концепцию объектно-ориентированного программирования. Это означает, что создаваемое приложение состоит из объектов, которые взаимодействуют между собой. Каждый объект имеет свои свойства, то есть характеристики (атрибуты) этого объекта, методы, определяющие поведение этого объекта, и события, на которые реагирует объект.

В языке *Object Pascal* классы являются специальными типами данных и используются для описания объектов. Соответственно объект, имеющий тип какого-либо класса, является экземпляром этого класса или переменной этого типа.

Класс представляет собой особый тип записи, имеющий в своем составе такие элементы, как поля, свойства и методы. Поля класса аналогичны полям записи и служат для хранения информации об объекте.

Методами называются процедуры и функции, предназначенные для обработки полей. Свойства занимают промежуточное положение между полями и методами.

С одной стороны, свойства можно использовать как поля, например, присваивая им значения с помощью оператора присваивания; с другой стороны, внутри класса доступ к значениям свойств выполняется методами класса.

Описание класса:

```
type TColorCircle = class(TCircle);
FLeft, FTop, FRight, FBottom: Integer;
Color: TColor;
end.
```

Класс ***TColorCircle*** создается на основе родительского класса ***TCircle***. В отличие от родительского, новый класс дополнительно содержит четыре поля типа ***integer*** и одно поле типа ***TColor***.

Если в качестве родительского используется класс ***Tobject***, который является базовым классом для всех классов, то его имя после слова ***class*** можно не указывать. Тогда первая строка описания будет выглядеть так:

```
type TNewClass = class.
```

Для различных элементов класса можно устанавливать разные права доступа (видимости), для чего в описании класса используются отдельные разделы, обозначенные специальными спецификаторами видимости.

Разделы ***private*** и ***protected*** содержат защищенные описания, которые доступны внутри модуля, в котором они находятся. Описания из раздела ***protected***, кроме того, доступны для порожденных классов за пределами названного модуля.

Раздел ***public*** содержит общедоступные описания, которые видимы в любом месте программы, где доступен сам класс.

Раздел ***published*** содержит опубликованные описания, которые в дополнение к общедоступным описаниям порождают динамическую информацию о типе ***Run-Time Type Information, RTTI***. По этой информации при выполнении приложения производится проверка на принадлежность элементов объекта тому или иному классу.

Одним из назначений раздела ***published*** является обеспечение доступа к свойствам объектов при конструировании приложений. В «Инспекторе объектов» видны те свойства, которые являются опубликованными.

Если спецификатор ***published*** не указан, то он подразумевается по умолчанию, поэтому любые описания, расположенные за строкой с указанием имени класса, считаются опубликованными.