Содержание

Типы данных языка Pascal

Компиляторы языка Pascal требуют, чтобы сведения об объеме памяти, необходимой для работы программы, были предоставлены до начала ее работы. Для этого в разделе *описания переменных* (`var`) нужно перечислить все переменные, используемые в программе. Кроме того, необходимо также сообщить компилятору, сколько памяти каждая из этих переменных будет занимать. А еще было бы неплохо заранее условиться о различных операциях, применимых к тем или иным переменным...

Все это можно сообщить программе, просто указав тип будущей переменной. Имея информацию о *типе переменной*, компилятор "понимает", сколько байт необходимо отвести под нее, какие действия с ней можно производить и в каких конструкциях она может участвовать.

Для удобства программистов в языке Pascal существует множество стандартных типов данных и плюс к тому возможность создавать новые типы.

Конструируя новые типы данных на основе уже имеющихся (стандартных или опять-таки определенных самим программистом), нужно помнить, что любое здание должно строиться на хорошем фундаменте. Поэтому сейчас мы и поговорим об этом "фундаменте".

На основании ***базовых типов данных*** строятся все остальные типы языка Pascal, которые так и называются:***конструируемые***.

Разделение на базовые и конструируемые типы данных в языке Pascal показано в таблице:

<table>
<tr><td rowspan="2">Базовые типы данных</td><td colspan="2">Дискретные типы данных</td><td colspan="2">Арифметические типы данных</td><td rowspan="2">Адресные типы данных</td><td rowspan="2">Структурированные типы данных</td></tr>
<tr><td>Целые</td><td>Вещественные</td></tr>
<tr><td>Логический
<code>boolean</code></td><td>Символьный (литерный)
<code>char</code></td><td><code>shortint</code>
<code>byte</code>
<code>integer</code>
<code>word</code>
<code>longint</code></td><td><code>real</code>
<code>single</code>
<code>double</code>
<code>extended</code>
<code>comp</code></td><td>Нетипизированный указатель
<code>pointer</code></td><td></td></tr>
<tr><td rowspan="5">Конструируемые типы</td><td rowspan="5"></td><td colspan="2">Перечисляемый
<code>week = (su, mo, tu,
we, th, fr,sa);</code></td><td rowspan="5"></td><td rowspan="5">Типизированный указатель
<code>^<тип></code></td><td>Массив <code>array</code></td></tr>
<tr><td rowspan="2">Интервал (диапазон)
<code>budni = mo..fr;</code></td><td rowspan="2"></td><td>Строка <code>string</code></td></tr>
<tr><td>Запись <code>record</code></td></tr>
<tr><td rowspan="2"></td><td>Файл
<code>text</code>
<code>file</code></td></tr>
<tr><td>Процедурный</td></tr>
<tr><td colspan="2">Типы данных, конструируемые программистом</td><td>Объектный[1)]</td></tr>
</table>

Типы данных, конструируемые программистом, описываются в разделе `type` по следующему шаблону:

```
type <имя_типа> = <описание_типа>;
```

Например:

```
type lat_bukvy = 'a'..'z','A'..'Z';
```

Базовые типы данных являются стандартными, поэтому нет нужды описывать их в разделе `type`. Однако при желании это тоже можно сделать, например, дав длинным определениям короткие имена. Скажем, введя новый тип данных

```
type int = integer;
```

можно немного сократить текст программы.

Стандартные конструируемые типы также можно не описывать в разделе `type`. Однако в некоторых случаях это все равно приходится делать из-за требований синтаксиса. Например, в списке параметров процедур или функций *конструкторы типов* использовать нельзя (см. лекцию 8).

Порядковые типы данных

Среди базовых типов данных особо выделяются порядковые типы. Такое название можно обосновать двояко:

1. Каждому элементу порядкового типа может быть сопоставлен уникальный (порядковый) номер. Нумерация значений начинается с нуля. Исключение - типы данных shortint, integer и longint. Их нумерация совпадает со значениями элементов.
2. Кроме того, на элементах любого порядкового типа определен порядок (в математическом смысле этого слова), который напрямую зависит от нумерации. Таким образом, для любых двух элементов порядкового типа можно точно сказать, который из них меньше, а который - больше2).

Стандартные подпрограммы, обрабатывающие порядковые типы данных

Только для величин порядковых типов определены следующие функции и процедуры:

3. Функция ord(x) возвращает порядковый номер значения переменной x (относительно того типа, к которому принадлежит переменная x).
4. Функция pred(x) возвращает значение, предшествующее x (к первому элементу типа неприменима).
5. Функция succ(x) возвращает значение, следующее за x (к последнему элементу типа неприменима).
6. Процедура inc(x) возвращает значение, следующее за x (для арифметических типов данных это эквивалентно оператору x:=x+1).
7. Процедура inc(x,k) возвращает k-е значение, следующее за x (для арифметических типов данных это эквивалентно оператору x:=x+k).
8. Процедура dec(x) возвращает значение, предшествующее x (для арифметических типов данных это эквивалентно оператору x:=x-1).
9. Процедура dec(x,k) возвращает k-е значение, предшествующее x (для арифметических типов данных это эквивалентно оператору x:=x-k).

На первый взгляд кажется, будто результат применения процедуры inc(x) полностью совпадает с результатом применения функции succ(x). Однако разница между ними проявляется на границах допустимого диапазона. Функция succ(x) неприменима к максимальному элементу типа, а вот процедура inc(x) не выдаст никакой ошибки, но, действуя по правилам машинного сложения, прибавит очередную единицу к номеру элемента. Номер, конечно же, выйдет за пределы диапазона и за счет усечения превратится в номер минимального значения диапазона. Получается, что

процедурыinc() и dec() воспринимают любой порядковый тип словно бы "замкнутым в кольцо": сразу после последнего вновь идет первое значение.

Поясним все сказанное на примере. Для типа данных

type sixteen = 0..15;

попытка прибавить 1 к числу 15 приведет к следующему результату:

+ 1 1 1 1

1

1 0 0 0 0

Начальная единица будет отсечена, и потому получится, что inc(15)=0.

Аналогичная ситуация на нижней границе допустимого диапазона произвольного порядкового типа данных наблюдается для процедуры dec(x) и функции pred(x):

dec(min_element)= max_element

Типы данных, относящиеся к порядковым

Опишем теперь *порядковые типы* данных более подробно.

1. *Логический тип* boolean имеет два значения: false и true, и для них выполняются следующие равенства:
2. ord(false)=0, ord(true)=1, false<true,
3. pred(true)=false, succ(false)=true,
4. inc(true)=false, inc(false)=true,dec(true)=false, dec(false)=true.
5. В *символьный тип* char входит 256 символов расширенной таблицы ASCII (например, 'a', 'b', 'я', '7', '#'). Номер символа, возвращаемый функциейord(), совпадает с номером этого символа в таблице ASCII.
6. *Целочисленные типы* данных сведем в таблицу:

Тип данных	Количество		Диапазон	
	байтов	битов		
shortint	1	8	-128..127	$-2^7..2^7-1$
byte	1	8	0..255	$0..2^8-1$
integer	2	16	-32768..32767	$-2^{15}..2^{15}-1$
word	2	16	0..65535	$0..2^{16}-1$
longint	4	32	-2147483648..2147483647	$-2^{31}..2^{31}-1$

7. Перечисляемые[3)] типы данных задаются в разделе type явным перечислением их элементов. Например:
type week =(sun,mon,tue,wed,thu,fri,sat)
0 1 2 3 4 5 6
Напомним, что для этого типа данных:
inc(sat) = sun, dec(sun) = sat.
8. Интервальные типы данных задаются только границами своего диапазона. Например:
type month = 1..12;
budni = mon..fri;
9. Программист может создавать и собственные типы данных, являющиеся комбинацией нескольких стандартных типов. Например:
type valid_for_identifiers = 'a'..'z','A'..'Z','_','0'..'9';

Этот тип состоит из объединения нескольких интервалов, причем в данном случае изменен порядок латинских букв: если в стандартном типе char 'A' < 'a', то здесь, наоборот, 'a' < 'A'. Для величин этого типа выполняются следующие равенства:

inc('z')='A'; dec('0')='_', pred('9')='8'; ord('b')= 2.

Вещественные типы данных

Напомним, что эти типы данных являются арифметическими, но не порядковыми.

Тип	Количество байтов	Диапазон (*абсолютной величины*)
single	4	$1.5*10^{-45}..3.4*10^{38}$
real	6	$2.9*10^{-39}..1.7*10^{38}$
double	8	$5.0*10^{-324}..1.7*10^{308}$
extended	10	$3.4*10^{-4932}..1.1*10^{4932}$
comp	8	$-2^{63}+1..2^{63}-1$

Конструируемые типы данных

Эти типы данных (вместе с определенными для них операциями) мы будем рассматривать далее на протяжении нескольких лекций:

Лекция 3. Массивы
Лекция 5. Строки и множества
Лекции 6 и 7. Файлы
Лекция 7. Записи
Лекция 8. *Процедурный тип данных*
Лекция 10. Указатели

Операции и выражения

Арифметические операции

Как мы уже упоминали, для каждого типа данных определены действия, применимые к его значениям. Например, если переменная относится к *порядковому типу* данных, то она может фигурировать в качестве аргумента стандартных функций `ord()`, `pred()` и `succ()` (см. п. "*Совместимость типов* данных" ниже). А к вещественным типам эти функции применить невозможно.

Итак, поговорим теперь об операциях - стандартных действиях, разрешенных для переменных того или иного*базового типа данных*. Основу будут составлять ***арифметические операции***, но, конечно же, мы не забудем и о *логическом типе* данных (операции, определенные для значений *символьного типа*, будут подробно рассмотрены в лекции 5).

Замечание: Все перечисленные ниже операции (за исключением унарных '-' и not) требуют двух операндов.

1. Логические операции (and, or, not, *xor*) применимы только к значениям типа `boolean`. Их результатом также служат величины типа `boolean`. Приведем таблицы значений для этих операций:

Оператор	Операнд 1	Операнд 2	Результат
not	false	-	true
	true	-	false
and	false	false	false
	false	true	false
	true	false	false
	true	true	true
or	false	false	false
	false	true	true
	true	false	true
	true	true	true
xor	false	false	false
	false	true	true
	true	false	true
	true	true	false

2. Операции сравнения (=, <>, >, <, <=, >=) применимы ко всем *базовым типам*. Их результатами также являются значения типа `boolean`.
3. Операции целочисленной арифметики применимы, как легко догадаться, только к *целым типам*. Их результат - целое число, тип которого зависит от типов операндов.

   `a div b` - деление a на b нацело (не нужно, наверное, напоминать, что деление на 0 запрещено, поэтому в таких случаях операция выдает ошибку). Результат будет принадлежать к типу данных, общему для тех типов, к которым принадлежат операнды. Например, (shortint div byte = integer). Пояснить это можно так: integer - это минимальный тип, подмножествами которого являются одновременно и byte, и shortint.

   `a mod b` - взятие остатка при делении a на b нацело. Тип результата, как и в предыдущем случае, определяется типами операндов, а 0 является запрещенным значением для b. В отличие от математической операции mod, результатом которой всегда является неотрицательное число, знак результата "программистской"

операции mod определяется знаком ее первого операнда. Таким образом, если в математике (-2 mod 5)=3, то у нас (-2 mod 5)= -2.

`a shl k` - сдвиг значения a на k битов влево (это эквивалентно умножению значения переменной a на 2^k). Результат операции будет иметь тот же тип, что и первый ее операнд (a).

`a shr k` - сдвиг значения a на k битов вправо (это эквивалентно делению значения переменной a на 2^k нацело). Результат операции будет иметь тот же тип, что и первый ее операнд (a).

`and,or,not,xor` - операции двоичной арифметики, работающие с битами *двоичного представления* целых чисел, по тем же правилам, что и соответствующие им логические операции.

4. Операции общей арифметики (+, -, *, /) применимы ко всем арифметическим типам. Их результат принадлежит к типу данных, общему для обоих операндов (исключение составляет только операция дробного деления /, результат которой всегда относится к вещественному типу данных).

Другие операции

Помимо арифметических, существуют и другие операции, специфичные для значений некоторых стандартных типов данных языка Pascal. Эти операции мы рассмотрим в соответствующих разделах:

```
#, in, +, *, [] : см. лекцию 5
@, ^            : см. лекцию 10
```

Стандартные арифметические функции

К арифметическим операциям примыкают и стандартные ***арифметические функции***. Их список с кратким описанием мы приводим в таблице.

	Описание	Тип аргумента	Тип результата
`abs(x)`	Абсолютное значение (модуль) числа	Арифметический	Совпадает с типом аргумента
`arctan(x)`	Арктангенс (в радианах)	Арифметический	Вещественный
`cos(x)`	Косинус (в радианах)	Арифметический	Вещественный
`exp(x)`	*Экспонента* (e^x)	Арифметический	Вещественный
`frac(x)`	Взятие дробной части числа	Арифметический	Вещественный
`int(x)`	Взятие целой части числа	Арифметический	Вещественный
`ln(x)`	*Натуральный логарифм* (по основанию e)	Арифметический	Вещественный
`odd(x)`	Проверка нечетности числа	Целый	`boolean`
`pi`	Значение числа π	-	Вещественный
`round(x)`	Округление к ближайшему целому	Арифметический	Целый
`trunc(x)`	Округление "вниз" - к ближайшему меньшему целому	Арифметический	Целый
`sin(x)`	Синус (в радианах)	Арифметический	Вещественный
`sqr(x)`	Возведение в квадрат	Арифметический	Вещественный
`sqrt(x)`	Извлечение квадратного корня	Арифметический	Вещественный

Арифметические выражения

Все арифметические операции можно сочетать друг с другом - конечно, с учетом допустимых для их операндов типов данных.

В роли операндов любой операции могут выступать переменные, константы, вызовы функций или выражения, построенные на основе других операций. Все

вместе и называется выражением. Определение выражения через выражение не должно вас смущать, ведь рекурсивное задание конструкций вообще свойственно программированию (см. лекцию 9).

Примеры ***арифметических выражений***:

`(x<0) and (y>0)` - выражение, результат которого принадлежит к *типу boolean*;

`z shl abs(k)` - вторым операндом является вызов стандартной функции;

`(x mod k) + min(a,b) + trunc(z)` - сочетание арифметических операций и вызовов функций;

`odd(round(x/abs(x)))` - "многоэтажное" выражение.

Полнота вычислений

В общем случае вычисление сложного логического выражения прекращается в тот момент, когда его окончательное значение становится понятным (например, true or (b<0)). Зачастую такой подход позволяет заметно сэкономить на выполнении "лишних" действий. Скажем, если есть некоторая сложно вычислимая функция my_func, вызов которой входит в состав выражения

```
if (x<=0) and my_func(z+12),
```

то для случая, когда x положительно, этих сложных вычислений можно избежать.

Однако включение директивы {$B+} принудит компилятор завершить эти вычисления даже в таком случае. Ее выключение {$B-} вернет обычную схему вычислений.

Порядок вычислений

Если в выражении расставлены скобки, то вычисления производятся в порядке, известном всем еще с начальной школы: чем меньше глубина вложенности скобок, тем позже вычисляется заключенная в них операция. Если же скобок нет, то сначала вычисляются значения операций с более высоким приоритетом, затем - с менее высоким. Несколько подряд идущих операций одного ***приоритета*** вычисляются в последовательности "слева направо".

Таблица 2.1. Приоритеты (для всех) операций языка Pascal

	Операции	Приоритет
Унарные[2)] операции	+, -, not, @, ^, #	Первый(высший)
Операции, эквивалентные умножению	*, /, div, mod, and, *shl*, shr	Второй
Операции, эквивалентные сложению	+,-, or, *xor*	Третий
Операции сравнения	=, <>, >, <, <=, >=, in	Четвертый

Замечание: Вызов любой функции имеет более высокий приоритет, чем все внешние относительно этого вызова операции. Выражения, являющиеся аргументами вызываемой функции, вычисляются в момент вызова (см. лекцию 8).

Примеры выражений (с указанием последовательности вычислений) для целых чисел:

```
a + b * c / d                (результат принадлежит к вещест-
  3   1   2                  венному типу данных);

a * not b or c * d = 0       (результат принадлежит к логиче-
  2  1     4   3   5          скому типу данных);

-min(a + b, 0) * (a + 1)     (результат принадлежит к целочис-
3 2    1       5    4         ленному типу данных).
```

Совместимость типов данных

В общем случае при выполнении арифметических (и любых других) операций компилятору требуется, чтобы типы операндов совпадали: нельзя, например, сложить массив и множество, нельзя передать *вещественное число*функции, ожидающей целый аргумент, и т.п.

В то же время, любая процедура или функция, написанная в расчете на вещественные значения, сможет работать и с целыми числами.

Правила, по которым различные типы данных считаются взаимозаменяемыми, мы приводим ниже.

Эквивалентность

Эквивалентность - это наиболее высокий уровень соответствия типов. Она требуется при действиях с указателями (см. лекцию 10), а также при вызовах подпрограмм. "А как же тогда быть с оговоркой, сделанной двумя абзацами выше?" - спросите вы. Мы не станем сейчас описывать механизм передачи аргументов процедурам и функциям, поясним лишь, что эквивалентность типов требуется только для *параметров-переменных* (см. лекцию 8).

Итак, два типа - T1 и T2 - будут эквивалентными, если верен хотя бы один вариант из перечисленных ниже:

- T1 и T2 совпадают;
- T1 и T2 определены в одном объявлении типа;
- T1 эквивалентен некоторому типу T3, который эквивалентен типу T2.

Поясним это на примере:

```
type T2 = T1;
          T3 = T1;
          T4,T5 = T2;
```

Здесь эквивалентными будут T1 и T2; T1 и T3; T1 и T4; T1 и T5; T4 и T5. А вот T2 и T3 - не эквивалентны!

Совместимость

Совместимость типов требуется при конструировании выражений, а также при вызовах подпрограмм (для параметров-значений). Совместимость означает, что для переменных этих типов возможна операция присваивания - хотя во время этой операции присваиваемое значение может измениться: произойдет ***неявное приведение типов данных*** (см. п. "Приведение типов данных" ниже).

Два типа T1 и T2 будут совместимыми, если верен хотя бы один вариант из перечисленных ниже:

- T1 и T2 эквивалентны (в том числе совпадают);
- T1 и T2 - оба целочисленные или оба вещественные;
- T1 и T2 являются подмножествами одного типа;
- T1 является некоторым подмножеством T2;
- T1 - строка, а T2 - символ (см. лекцию 5);
- T1 - это тип pointer, а T2 - *типизированный указатель* (см. лекцию 10);
- T1 и T2 - оба процедурные, с одинаковым количеством попарно эквивалентных параметров, а для функций - с эквивалентными типами результатов (см. лекцию 8).

Совместимость по присваиванию

В отличие от простой совместимости, совместимость по присваиванию гарантирует, что в тех случаях, когда производится какое-либо присваивание (используется запись вида a:=b; или происходит передача значений в

подпрограмму[1] или из нее и т.п.), не произойдет никаких изменений присваиваемого значения.

Два типа данных T1 и T2 называются совместимыми по присваиванию, если выполняется хотя бы один вариант из перечисленных ниже:

- T1 и T2 эквивалентны, но не файлы[2] ;
- T1 и T2 совместимы, причем T2 - некоторое подмножество в T1;
- T1 - вещественный тип, а T2 - целый.

Приведение типов данных

Неявное приведение типов данных

Как мы упомянули в п. "Арифметические операции" выше, тип результата арифметических операций (а следовательно, и выражений) может отличаться от типов исходных операндов. Например, при "дробном" делении (/) одного целого числа на другое целое в ответе все равно получается вещественное. Такое изменение типа данных называется неявным приведением типов.

Если в некоторой операции присваивания участвуют два типа данных совместимых, но не совместимых по присваиванию, то тип присваиваемого выражения автоматически заменяется на подходящий. Это тоже неявное приведение. Причем в этих случаях могут возникать изменения значений. Скажем, если выполнить такую последовательность операторов

```
a:= 10;                    {a: byte}
a:= -a;
writeln(a);
```

то на экране мы увидим не -10, а 246 (246 = 256 - 10).

Неявным образом осуществляется и приведение при несоответствии типов переменной-счетчика и границ в циклах for (см. лекцию 3).

Неявное приведение типов данных можно отключить, если указать директиву компилятора {$R+}, которая принуждает компилятор всегда проверять границы и диапазоны. Если эта директива включена, то во всех ситуациях, в которых по умолчанию достаточно совместимости типов данных, будет необходима их эквивалентность.

По умолчанию такая проверка отключена, поэтому во всем дальнейшем изложении (если, конечно, явно не оговорено противное) мы будем считать, что эта директива находится в выключенном состоянии {$R-}.

Явное приведение типов данных

Тип значения можно изменить и ***явным*** способом: просто указав новый тип выражения, например: a:= byte(b). В этом случае переменной a будет присвоено значение, полученное новой интерпретацией значения переменной b. Скажем, если b имеет тип shortint и значение -23, то в a запишется 233 (= 256 - 23).

Приводить явным образом можно и типы, различающиеся по длине. Тогда значение может измениться в соответствии с новым типом. Скажем, если преобразовать тип longint в тип integer, то возможны потери из-за отсечения первых двух байтов исходного числа. Например, результатом попытки преобразовать число 100 000 к типу integer станет число 31 072, а к типу word - число 34 464.

Функции, изменяющие тип данных

В заключение мы приведем список стандартных функций, аргумент и результат которых принадлежат к совершенно различным типам данных:

```
trunc[3]: real -> integer;
round: real -> integer;
val[4]: string -> byte/integer/real;
chr[5]: byte -> char;
ord: <порядковый_тип> -> longint;
```