Операции и выражения

Арифметические операции

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()

succ()

К вещественным типам эти функции применить невозможно.

Операции - стандартные действия, разрешенные для переменных того или иного типа данных.

Все операции (за исключением унарных и not), требуют двух операндов.

1. **Логические операции**

   (and, or, not, xor)

применимы только к значениям типа **boolean.**

результатом выполнения операций также являются величины типа boolean.

Для данных логического типа определены следующие логические операции

NOT - отрицание: **NOT(True)=false;**

OR - логическое сложение(ИЛИ): **True OR False = True;**

AND - логическое умножение(И): **False AND True = False;**

XOR - исключающее ИЛИ:

True XOR True = false;

True XOR False = True;

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

   a div b - деление a на b нацело (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.

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).

Стандартные арифметические функции

К арифметическим операциям примыкают и стандартные арифметические функции. Их список с кратким описанием приводится по схеме:

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 – 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. Возможно определение выражения через выражение.

Примеры арифметических выражений:

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

(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 and (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 обычном порядке (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".

Приоритеты операций языка Pascal

Операции	Приоритет
Унарные операции **+, -, not, @, ^, #**	**Первый(высший)**
Операции ***, /, div, mod, and, shl, shr**	**Второй**
эквивалентные умножению	
Операции **+,-, or, xor**	**Третий**
эквивалентные сложению	
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.

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 взаимозаменяемости различных типов данных

Эквивалентность - это наиболее высокий уровень соответствия типов. Она требуется при действиях с указателями, а также при вызовах подпрограмм.

Два типа - T1 и T2 - будут эквивалентными, если верно хотя бы одно из перечисленных ниже:

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

например:

```
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.

T2 и T3 - не эквивалентны!

Совместимость типов

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***

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;
- T1 - это тип pointer, а T2 - типизированный указатель
- T1 и T2 - оба процедурные, с одинаковым количеством попарно эквивалентных параметров, а для функций - с эквивалентными типами результатов

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 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;
- T1 и T2 совместимы, причем T2 - некоторое подмножество в T1;
- T1 - вещественный тип, а T2 - целый.

Приведение типов данных

Неявное приведение типов данных

Как известно, 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 результат будет

не -10, а 246 (246 = 256 - 10).

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 обычно считается 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

Список стандартных функций, аргумент и результат которых принадлежат к различным типам данных:

trunc: real -> integer;

round: real -> integer;

val: string -> byte/integer/real;

chr: byte -> char;

ord: <порядковый_тип> -> longint;