ТУБ №22:
Выражения и приоритеты операций

Редкая программа обходится без вычислений. Даже чтобы окно программы при запуске всегда оказывалось в центре экрана, иной раз приходится изрядно попотеть: всё-таки мониторы у пользователей разные, разрешения экрана тоже, а ещё некоторые масштабирование включают — так и вообще беда. Что уж говорить о 3D-графике, где вообще приходится и с матрицами повозиться, и вектора поперемножать, и ещё потом для анимации немного арифметики задействовать.

Для организации вычислений в большинстве языков программирования используются выражения.

Выражение — это формула для вычисления некоторого значения, состоящая из операндов, знаков операций и круглых скобок.

Возьмём, скажем, вычисление дискриминанта квадратного трёхчлена: $D = b^2 - 4ac$. В математике обычно рассматривают взаимосвязи между некоторыми величинами и формулы соответственно задаются в виде отношений этих величин, но нам для программирования хорошо бы переложить эту формулу на язык конкретных действий, которые необходимо выполнить для получения результата. Например, так:

```
D := B * B - 4 * A * C
```

Вот та часть, которая после знака присваивания `:=` оказалась записана, — это и есть выражение. Знаки операций — это `*` и `-`, без круглых скобок удалось обойтись. А где же тогда операнды?

Операнд — величина, участвующая в выражении, к которой применяются операции.

Соответственно операнды в нашем примере — это `A`, `B` и `C`. Обычно в роли операндов выступают идентификаторы переменных и констант, а также литералы. В нашем примере, кстати, и литерал имеется — число 4.

Как и в математике, в языках программирования операции могут иметь различные приоритеты. Количество уровней приоритетов зависит от языка программирования: например, в языке C (1972 г.) операции образуют 16 уровней приоритетов, в C++ (1983 г.) их уже 18 или 19, смотря как считать. Как видим, создатели этих языков изрядно потрудились, чтобы допустить ошибку в программе было легко не только начинающим.

Посмотрим на языки поновее. Python (1991 г.) — 18 уровней. Появившиеся чуть позже Java (1995 г.) и C# (2000 г.) имеют 14 и 16 уровней приоритетов соответственно.

Pascal и Delphi смотрят на это безобразие с недоумением и 4 уровнями приоритетов, которые к тому же элементарно запоминаются:

1. Операции `not` и `@`
2. Операции группы умножения: `*  /  div  mod  and  shl  shr`
3. Операции группы сложения: `+  -  or  xor`
4. Операции группы сравнения: `in  =  <>  >  >=  <  <=`

Принцип запоминания прост. Самый высокий приоритет имеют две унарные операции. Их, как мы сейчас увидим, даже не придётся запоминать.

Дальше вспоминаем школьные уроки математики: умножение приоритетнее сложения.

Какие операции относятся к группе умножения? Само умножение — логично, деление — это **умножение** на обратное число. Целочисленное деление `div` и взятие остатка `mod` — это тоже разновидность **деления**. Операция `and` по-другому ещё называется как? Логическое **умножение**. Операции сдвига эквивалентны **умножению/делению** на степень двойки.

Ниже приоритетом сложение. Вычитание — это **сложение** с противоположным по знаку числом. Операцию `or` ещё называют логическим **сложением**. А `xor` — это же exclusive OR, разновидность логического сложения, ещё по-другому называется **сложением** по модулю 2.

И вот теперь, когда все вычисления выполнены, можно оценить полученные результаты, **сравнить**, например, и принять решение в зависимости от того, как они соотносятся с нашими ожиданиями и между собой.

В Delphi появилось ещё две операции, которые немного не вписываются в эту картину мира: это операции `as` и `is`, но они используются только в определённых видах выражений, где кроме них встретятся разве что операции сравнения = и <>, и то вряд ли, да и выражения эти потребуются только после набора значительного опыта программирования, так что беспокоиться не о чем.

Что окончательно развеять все сомнения, дополним эту картину мира *правилами написания и вычисления выражений*.

1. Первая группа операций имеет самый высокий приоритет, т.е. операции этой группы вычисляются в первую очередь.

2. В каждой из групп операции имеют одинаковый приоритет.

3. Операции одного приоритета обычно вычисляются в порядке их следования в выражении (слева направо). В этом случае компилятор может поменять порядок вычисления для повышения быстродействия программы, но только если это не повлияет на результаты.

4. Операнд, находящийся между двумя операциями с различными приоритетами, связывается с операцией, имеющей более высокий приоритет.

5. Операнд, находящийся между двумя операциями с равными приоритетами, связывается с операцией, которая находится слева от него.

6. Для уточнения последовательности действий используются круглые скобки. Действия в скобках выполняются в первую очередь.

7. Последовательная запись двух знаков операций запрещена. Исключение — операция `not`.

8. Многоэтажные формулы записываются в одну строку.

9. Знак умножения `*` опускать нельзя.

Так почему же не придётся запоминать две операции с самым высоким приоритетом? Да потому что в здравом уме невозможно, глядя на выражение с ними, подумать, что они вычисляются позже других операций. Они ведь унарные (с одним операндом). Ну вот, например:

```
X + not Y
```

Можно ли здесь решить, что сложение выполнится раньше?

Для операции `@` всё ещё проще: она унарная, у неё один операнд. А значит, с другой стороны от неё может оказаться только знак операции. Два знака операций подряд записывать нельзя, нужны скобки. А скобки уже однозначно дают понять, какая операция отработает раньше:

```
X + (@Y)
```

И это не говоря уже о том, что сама необходимость в подобных выражениях в Pascal/Delphi возникает крайне редко.

Ещё раз кратко подведём итоги. Четыре уровня приоритетов: две унарные операции, группа умножения, группа сложения, группа сравнения. Операции с равным приоритетом вычисляются слева направо. Всё.

Виды выражений

Среди всех выражений особое место занимают выражения, которые можно вычислить, не запуская программу. Например:

```
5 * Abs(9 - 16 * 8) shl 4
```

Такие выражения называются *константными*, и компилятор вычисляет их значение во время компиляции программы. В саму программу вместо вычислений по такой формуле идёт уже вычисленное значение. Это позволяет повысить скорость работы программы.

Напоследок вспомним о том, что выражение — это формула вычисления некоторого значения. Это является характерной особенностью выражения: выражение всегда чему-то равно.

Типом выражения называют тип значения, получившегося в результате вычисления этого выражения.

Так, например, *арифметические выражения* — это выражения, результатом вычисления которых является целое или вещественное число. Аналогично *символьным выражением* считается выражение, дающее в результате значение символьного типа, а *логическим выражением* — имеющее результат логического типа.

Дополнительные вопросы

1. Можно ли определить тип выражения, не вычисляя его значение? Как?

2. Даны координаты точки на плоскости `x` и `y`. Почему в выражении, проверяющем, находится ли точка с такими координатами в третьем квадранте (четверти) координатной плоскости, придётся использовать скобки?

3. Даны координаты точки на плоскости `x` и `y`. Запишите выражение, проверяющее находится ли точка с такими координатами на одной из координатных осей.

4. Как записать то же самое выражение, не используя скобок?

5. Почему в ответе на вопрос 4 нежелательно использовать операцию умножения?