ТУБ №16:
Работа с целочисленными типами данных в Delphi

Определившись с множествами значений различных целочисленных типов (а заодно и представлением значений этих типов в памяти компьютера), недурно было бы разобраться и с тем, что с этими значениями можно делать.

Важным свойством целочисленных типов данных в Delphi является свойство перенумерованности. Его наличие не только означает, что все значения целочисленных типов могут быть упорядочены по возрастанию единственным способом, но и позволяет предсказать наличие у этих типов нескольких определённых над ними встроенных функций. Тем не менее, намного более важной характеристикой типов данных является множество операций.

Операции над целочисленными типами данных

Для целочисленных типов данных может быть выделено несколько групп операций. Начнём с самых простых для понимания — арифметических.

Операция	Вид	Описание	Тип результата
+	Одноместная	Сохранение знака	Целый
–	Одноместная	Изменение знака	Целый
+	Двухместная	Сложение	Целый
–	Двухместная	Вычитание	Целый
*	Двухместная	Умножение	Целый
/	Двухместная	Деление	Вещественный
div	Двухместная	Целочисленное деление	Целый
mod	Двухместная	Остаток целочисленного деления	Целый

Начнём с одноместных операций. Кажущаяся самой нелогичной одноместная операция — операция сохранения знака. Действительно, результат её применения к любому целому числу — само это число. Вдвойне удивительным выглядит то, что она есть в большинстве языков программирования. Зачем нужна такая операция?

Как ни странно, введение такой операции в языки программирования — по большей части решение уровня математической эстетики: если есть «унарный минус», то почему не должно быть «унарного плюса»? На практике же это позволяет иметь чуть более свободный синтаксис. Например, нередко программистам приходится задавать целые наборы чисел, причём в виде матриц, и эти числа имеют различные знаки. Сравните две записи:

```
..., -42, 28, -193, -115,  1025, 2019, ...
...,  15, 43,  214, -215, -1613, 3511, ...
```

и

```
..., -42, +28, -193, -115, +1025, +2019, ...
..., +15, +43, +214, -215, -1613, +3511, ...
```

В ряде случаев такая запись может оказываться более наглядной, особенно если знаки должны чередоваться определённым образом независимо от модулей самих чисел. Аналогичный случай — вычисление нескольких величин по схожим формулам. Например,

```
x1 := 2 * x + 5;
x2 := -2 * x + 5;
x3 := 2 * x - 5;
x4 := -2 * x - 5;
```

выглядит явно хуже, чем

```
x1 := +2 * x + 5;
x2 := -2 * x + 5;
x3 := +2 * x - 5;
x4 := -2 * x - 5;
```

В других языках программирования операция сохранения знака также нашла себе применение как более компактный способ манипуляции типами данных для некоторых случаев. Впрочем, это применение скорее следует считать «костылём» и «грязным хаком», т.к. может быть неочевидно, если ранее Вы не сталкивались с такими приёмами.

Операции изменения знака, сложения, вычитания, умножения и деления в Delphi работают в строгом соответствии с одноимёнными математическими операциями. Кстати, в так называемых C-подобных языках программирования операция деления работает несколько иначе, что явно не помогает начинающим в изучении программирования, а в ряде случаев приводит к необходимости писать более громоздкий код. К счастью, автор языка Pascal, от которого произошла Delphi, Никлаус Вирт подошёл к разработке своего языка более ответственно и подобных нюансов в этих языках не наблюдается.

Кроме обычного деления в математике существует также понятие деления с остатком, или целочисленного деления. При таком делении кроме частного (которое называется неполным частным) образуется ещё и некоторый остаток. Сами по себе эти операции работают достаточно очевидно:

```
d := 7 div 2; // Результат — 3
r := 7 mod 2; // Результат — 1
```

Отдельного внимания заслуживает случай, когда делимое или делитель — отрицательные. В этом случае у различных языков программирования нет договорённости о том, какой знак должен иметь остаток. Например, в таких языках, как современные C, C++, C# и Java знак результата эквивалентной операции совпадает со знаком делимого. В то же время, например, Lua, Perl, Python и Ruby выбирают для остатка знак делителя.

Разработчики Delphi придерживаются первого подхода, т.е. знак остатка совпадает со знаком делимого. Такое поведение соответствует тому, как работают команды процессоров Intel/AMD, виртуальная машина платформы Android и т.д. Другими словами, именно такая реализация наиболее эффективна. Кроме того, при такой реализации соблюдается математическое правило целочисленного деления:

$$X = dq + r,$$

где X — делимое, q — делитель, d — неполное частное, а r — остаток.

Следующая группа операций — побитовые операции.

Операция	Вид	Описание	Тип результата
not	Одноместная	Поразрядное дополнение	Целый
and	Двухместная	Поразрядное логическое умножение (И)	Целый
or	Двухместная	Поразрядное логическое сложение (ИЛИ)	Целый
xor	Двухместная	Поразрядное логическое исключающее ИЛИ	Целый

Наиболее широко эти операции применяются при реализации алгоритмов шифрования и проверки целостности, но этим их применение не ограничивается. Принцип их работы заключается в том, что к двоичному представлению операндов применяется одноимённая логическая операция (к разрядам, занимающим одинаковые позиции в записи обоих чисел). Например, для операции and:

$$\begin{array}{l} 12 = 0000\ 1100_{(2)} \\ 5 = 0000\ 0101_{(2)} \\ \text{----------} \\ 0000\ 0100_{(2)} = 4_{(10)} \end{array}$$

Таким образом, 12 and 5 = 4. Аналогично при использовании операции or:

$$\begin{array}{l} 12 = 0000\ 1100_{(2)} \\ 5 = 0000\ 0101_{(2)} \\ \text{----------} \\ 0000\ 1101_{(2)} = 13_{(10)} \end{array}$$

Операция xor вычислится следующим образом:

$$\begin{array}{l} 12 = 0000\ 1100_{(2)} \\ 5 = 0000\ 0101_{(2)} \\ \text{----------} \\ 0000\ 1001_{(2)} = 9_{(10)} \end{array}$$

Как правило, побитовые операции применяются к значениям беззнаковых типов или в тех случаях, когда заранее известно, что операция не будет затрагивать знаковый бит. Сами операции работают исключительно с двоичным представлением, но то, как полученный результат будет интерпретироваться дальше в программе, может зависеть от используемого кода.

Возьмём, например, число 12 и вычислим not 12:

$$\begin{array}{l} 12 = 0000\ 1100_{(2)} \\ \text{----------} \\ 1111\ 0011_{(2)} \end{array}$$

Полученная комбинация битов может соответствовать разным числам. Если интерпретировать её, как запись числа со знаком в дополнительном коде, это будет число –13. Если же считать число беззнаковым, такой комбинации будет соответствовать число 243.

Операции, работающие похожим образом — с побитовым представлением числа, — операции сдвига.

Операция	Вид	Описание	Тип результата
shl	Двухместная	i shl j — побитовый сдвиг влево значения i на j позиций	Целый
shr	Двухместная	i shr j — побитовый сдвиг вправо значения i на j позиций	Целый

Обозначения этих операций являются сокращениями от SHift Left и SHift Right. Характерным свойством этих операций является их эквивалентность для беззнаковых типов соответственно умножению и целочисленному делению на степень числа 2. Например,

```
7 shl 1 = 4
9 shl 2 = 36
24 shr 2 = 6
19 shr 1 = 9
```

Чтобы понять, почему это так, необходимо записать исходные числа в двоичной системе счисления:

```
 7 = <- 0000 0111 <-
        0000 1110

 9 = <- 0000 1001 <-
        0001 0010
        0010 0100

24 = -> 0001 1000 ->
        0000 1100
        0000 0110

19 = -> 0001 0011 ->
        0000 1001
```

Следующая группа операций — операции сравнения.

Операция	Вид	Описание	Тип результата
=	Двухместная	Равно	Логический
<>	Двухместная	Не равно	Логический
<	Двухместная	Меньше	Логический
>	Двухместная	Больше	Логический
<=	Двухместная	Меньше или равно	Логический
>=	Двухместная	Больше или равно	Логический

Отличительной особенностью Pascal-подобных языков (к которым относится и Delphi) является использование для обозначения операций сравнения исключительно математических знаков. При этом для тех операций, для которых отсутствует отдельный символ на клавиатуре, используется комбинация из двух соответствующих знаков: <= (меньше или равно), >= (больше или равно), <> (меньше или больше, т.е. не равно).

Все операции сравнения возвращают результат так называемого логического типа. Это означает, что результатом может быть одно из двух значений — истина или ложь.

Встроенные процедуры и функции над целочисленными типами данных

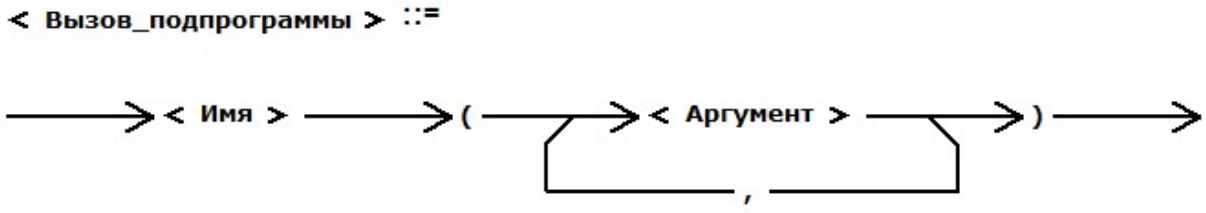
Помимо операций в языках программирования обычно выделяют встроенные процедуры и функции, предназначенные для работы с данными тех или иных типов. Это позволяет избежать усложнения синтаксиса языка.

Функции в программировании похожи на функции в математике: как правило, у каждой из них есть один или несколько аргументов и значение. Однако в отличие от математики, где функция просто задаёт зависимость между некоторыми величинами, в программировании под функцией в общем случае понимают алгоритм, который на основании значений аргументов выполняет некоторые действия, обеспечивающие вычисление значения функции. При этом действия могут быть совершенно произвольными, необязательно вычислительного характера.

В отношении значения функции принято говорить, что функция «возвращает значение».

Процедуры от функций отличаются отсутствием вычисляемого результата: они лишь выполняют те или иные действия в зависимости от значений аргументов. Собирательное название для процедур и функций — подпрограммы.

Для вызова подпрограмм в Delphi используется следующий синтаксис:



Для работы с целыми числами имеются следующие встроенные подпрограммы:

Имя	Вид	Описание	Тип результата
Chr(x)	Функция	Возвращает символ, код которого задан аргументом	Символьный
Abs(x)	Функция	Возвращает модуль числа	Тип x
Sqr(x)	Функция	Возвращает квадрат числа x	Тип x
Odd(x)	Функция	Определяет, является ли аргумент нечётным числом	Логический
Dec(x[, n])	Процедура	Уменьшает значение переменной x на число n (или на 1, если n не указано)	—
Inc(x[, n])	Процедура	Увеличивает значение переменной x на число n (или на 1, если n не указано)	—
Random(x)	Функция	Возвращает псевдослучайное целое число в диапазоне от 0 до x–1	
Оставленные для совместимости с Turbo Pascal			
Lo(x)	Функция	Возвращает значение младшего байта аргумента x (*)	Byte
Hi(x)	Функция	Возвращает значение старшего байта аргумента x (*)	Byte
Swap(x)	Функция	Возвращает значение, полученное путём обмена старшего и младшего байтов аргумента (*)	Тип x
Определённые для всех перенумерованных типов			
Ord(x)	Функция	Возвращает порядковый номер значения аргумента во множестве значений типа	Integer
Pred(x)	Функция	Возвращает значение, следующее во множестве значений типа перед значением аргумента	Тип x
Succ(x)	Функция	Возвращает значение, следующее во множестве значений типа после значения аргумента	Тип x
Определённые для всех типов			
SizeOf(x)	Функция	Возвращает размер типа в байтах	Integer

(*) Применяются только к двухбайтовым типам данных. При использовании для больших типов работают только с двумя младшими байтами.

Следует обратить особое внимание на принцип работы функции Random. Свойства правильного алгоритма, в особенности такие, как дискретность и определённость, лежат в основе устройства современных компьютеров, поэтому осуществить генерацию по-настоящему случайных чисел с использованием только команд процессора технически невозможно: требуется внешний источник случайности.

Тем не менее, на практике истинной случайности и не требуется, более того — по-настоящему случайные величины могут плохо подходить для конкретных практических задач. Поэтому в большинстве случаев вместо истинно случайных значений используют псевдослучайные — такие, которые выглядят, как случайные, но в действительности получены по определённому алгоритму.

Каждое следующее псевдослучайное число получается по определённой формуле из значений предыдущих (зачастую — одного). По этой причине, если не принять дополнительных мер, при каждом запуске программы будут выдаваться одни и те же псевдослучайные числа в одном и том же порядке. Чтобы этого избежать, необходимо при каждом запуске изменять «отправную точку» для генерации псевдослучайных чисел — так называемый random seed. Для этого в Delphi предусмотрена процедура Randomize. За редким исключением, её необходимо вызывать ровно один раз за время выполнения программы. Чаще всего это делают сразу после запуска программы.

Функции Ord, Pred и Succ определены в Delphi для всех типов, обладающих свойством перенумерованности. Для целочисленных типов Ord(x) возвращает значение x, Pred(x) — число на 1 меньшее x, Succ(x) — число на 1 большее x. Нетрудно заметить, что большой практической пользы от таких функций нет. Тем не менее, эти функции отражают способ работы некоторых конструкций языка с данными перенумерованных типов. С точки зрения производительности Pred и Succ создают минимальные накладные расходы (эквивалентно прибавлению или вычитанию 1), Ord не стоит ничего (не требует времени на выполнение).

Дополнительные вопросы

1. Как, используя только операции и встроенные подпрограммы, определённые над целочисленными типами, реализовать вычисление остатка от деления в стиле Lua, Perl, Python и Ruby, т.е. чтобы знак остатка совпадал со знаком делителя, а не делимого?
2. Почему в Lua, Perl, Python и Ruby операция взятия остатка от деления выполняется медленнее?
3. Как определить, является ли число нечётным, не используя функцию Odd и операцию mod?
4. Как заменить функцию Swap операциями, определёнными над целочисленными типами данных?
5. Почему потребовалось добавление встроенной функции Swap и почему в настоящее время можно обойтись без неё?
6. Зачем может понадобиться обмен местами значений отдельных байтов целочисленной переменной?