ТУБ №17:
Вещественные типы данных в Delphi

Вещественные типы данных используются для представления значений тех величин, которые могут в процессе вычислений принимать нецелые значения.

Конечно, проблемы начинаются сразу же, как только мы задумываемся о том, как записывать такие числа в памяти компьютера, ведь в одних задачах приходится работать с очень большими вещественными числами, в других — с очень маленькими, в третьих — вообще и с теми, и с другими. Тут ещё не очень понятно, как вообще их записывать, а уже очевидно, что будет непросто угодить всем.

К счастью, умные математики всё уже придумали, а инженеры реализовали в аппаратуре, поэтому нам остаётся только с благодарностью пользоваться плодами их работы. А чтобы приходящим в профессию новичкам было проще понять, как работают вещественные числа, идеи этих людей были оформлены в стандарт IEEE 754.

Как взрослые дяди вещественные числа придумывали

Со стандартом, кстати, тоже интересная история вышла [1]. Компьютеры изначально создавались для решения преимущественно научных задач, а значит, попытки вести в компьютерных программах вычисления над вещественными числами начались почти сразу же, как только первая ЭВМ подала признаки жизни. Поскольку на отсутствие серьёзного математического образования первые программисты не жаловались, идей того, как заставить спагетти из проводов и логических элементов работать с дробями, было в избытке. В результате к 60–70-м годам XX века этих самых идей накопилось столько, что разобраться в этом многообразии могли только самые стойкие.

В 1976 году ещё сравнительно молодая, но уже весьма успешная компания Intel озадачилась разработкой математического сопроцессора (вспомогательного вычислительного устройства) для своих процессоров 8086 и 8088. Задача ставилась серьёзная: в сопроцессорах Intel должна была использоваться «самая лучшая вещественная арифметика», для чего к разработке были привлечены ведущие специалисты в этой области.

Другие крупные компании того времени того времени, понимая важность момента, не могли оставаться в стороне, тем более, что у некоторых из них уже были свои аппаратные решения для работы с вещественными числами: для них было особенно важно знать, что придумали в Intel, чтобы не упустить конкурентное преимущество. Возникший ажиотаж привёл к созданию комитета, результатом работы которого вскоре стал стандарт IEEE 754.

Среди возможных реализаций поддержки вещественных чисел двумя наиболее перспективными и обсуждаемыми стали реализация компании Intel под условным названием K-C-S (по именам авторов — William Kahan, Jerome Coonen и Harold Stone) и реализация компании DEC, незадолго до этого приступившей к выпуску процессоров архитектуры VAX с поддержкой вещественной арифметики. Основными преимуществами реализации VAX были простота и быстродействие, а также наличие уже работающих решений, основанных на этой реализации. Предложение Intel выглядело громоздким, а возможность его эффективной аппаратной реализации была совершенно неочевидной (хотя, разумеется, в Intel знали, как это сделать), но важным преимуществом K-C-S была поддержка многих полезных возможностей, которых не было у DEC VAX, в особенности в части работы с очень малыми числами.

Пока происходили бурные обсуждения с привлечением видных учёных, затрагивавшие вопросы точности представления вещественных чисел и другие возможные проблемы конкурирующих реализаций, произошло то, что, вероятно, и должно было произойти: были предприняты попытки реализовать K-C-S в аппаратуре и они оказались весьма успешными, в том числе не оправдались опасения, что K-C-S окажется медленной. В результате к 1984 году K-C-S нашла воплощение в продукции Intel, AMD, Apple, IBM, Motorola, National Semiconductor, Weitek, Zilog, AT&T и многих других крупных игроков рынка того времени.

K-C-S стала стандартом де-факто. Комитету не оставалось ничего, кроме как в 1985 году стандартизировать реализацию, получившую широчайшее распространение. На сегодняшний день крайне сложно найти устройство, которое бы позволяло работать с вещественными числами, но не следовало при этом IEEE 754, а значит, и реализации от Intel.

Вещественные типы данных

Стандарт IEEE 754-1985 определил 4 основных способа записи вещественных чисел в памяти компьютера:

* одинарной точности (Single Precision) — 32 бита;

* двойной точности (Double Precision) — 64 бита;

* расширенной одинарной точности (Single-Extended Precision) — 43 и более бита;

* расширенной двойной точности (Double-Extended Precision) — 79 и более бит.

Single-Extended Precision широкого применения не нашёл, отчасти потому что не был реализован самой Intel в математических сопроцессорах. Остальные три способа записи выглядят следующим образом (в реализации Intel):

Для вещественных типов данных используется представление с плавающей точкой (floating-point). Оно хорошо знакомо каждому по экспоненциальной форме записи, которая применяется в школе на уроках физики. В этой форме записи число представляется в виде мантиссы и порядка. При этом мантисса содержит в себе все значащие цифры числа, а порядок задаёт степень, в которую нужно возвести основание системы счисления, чтобы после домножения мантиссы на такую степень получалось реальное значение числа. Например:

$2.5 \cdot 10^{-3} \qquad -13.967551 \cdot 10^{12} \qquad 0.0023 \cdot 10^{7}$

Основное отличие заключается в том, что в вещественных типах данных используется, конечно же, двоичная система счисления.

Наблюдательный читатель может заметить, что иллюстрация к типу Extended несколько отличается от двух других — единицей. Чтобы понять назначение этой единицы, необходимо обратить внимание на то, что одно и то же число можно записать в экспоненциальной форме бесконечным количеством способов:

$1.3 \cdot 10^{0} = 13 \cdot 10^{-1} = 0.13 \cdot 10^{1} = 130 \cdot 10^{-2} = 0.013 \cdot 10^{2} = \ldots$

Аналогичная ситуация наблюдается и при записи в двоичной системе счисления. Получается, несколько различных комбинаций мантиссы и порядка могут задавать одно и то же число. При записи вещественных чисел в ограниченное количество бит это означало бы, что мы попусту тратим комбинации этих битов. Во избежание этой проблемы из всех возможных способов записи выбирается такой, чтобы значение мантиссы находилось в диапазоне $[1;2)$. При соблюдении этого условия запись мантиссы в двоичной системе счисления будет иметь следующий вид:

1.xxxxxxxxx

где x — двоичные цифры дробной части.

Поскольку нормализованная мантисса всегда имеет 1 в целой части, эту цифру можно не записывать, а лишь подразумевать, что она имеется в записи числа, и сэкономить этим 1 бит, который лучше использовать для хранения большего количества цифр дробной части.

Однако особенностью типа Extended в реализации Intel является то, что именно он применяется для вычислений в математическом сопроцессоре (в наше время математический сопроцессор из отдельного устройства превратился в одну из частей процессора). Это, в частности, означает, что для вычислений значения всех остальных типов конвертируются в Extended, а затем результат при необходимости преобразуется обратно в требуемый вещественный тип. Ну а при вычислениях удобно иметь все значащие цифры числа, поэтому-то в Extended старшая единица присутствует явно.

Следует иметь в виду важную особенность вещественных типов данных: представление чисел при использовании этих типов данных может быть неточным. Возьмём, например, какое-нибудь не очень громоздкое десятичное число. Скажем, 0.2. В двоичной системе счисления оно выглядит так:

0.00110011001100110011001100110011…

т.е. является бесконечной периодической дробью! Но для записи мантиссы, т.е. значащих цифр, у нас есть ограниченное количество бит. Значит, какими-то из них придётся жертвовать, и очевидно, что жертвовать придётся младшими цифрами. Но это означает, что фактически в переменной вещественного типа будет храниться не 0.2, а число, которое будет чуть больше или чуть меньше.

Погрешности представления вещественных чисел при выполнении вычислений могут как компенсировать друг друга, так и накапливаться. Поэтому вполне возможна ситуация, когда математически равные величины, но вычисленные разными способами, окажутся неравными. По этой причине существует правило:

Проверять вещественные числа на точное равенство нельзя.

На самом деле, конечно, такое сравнение никто не запретит, но его результат может отличаться от ожидаемого, поэтому новичкам, как правило, запрещают это делать.

На практике сравнение вещественных чисел может быть вполне безопасным, если программист хорошо понимает, что делает. Насколько безопасным? Настолько, что, например, в JavaScript для представления даже целых чисел используется тип Double! В этом типе данных погрешности при вычислениях в целых числах начинают проявлять себя только на очень больших числах. Следует, однако, понимать, что это касается ТОЛЬКО работы с целыми числами. Если в процессе вычислений появляются дробные числа, эти гарантии могут не соблюдаться.

Вещественные типы данных в Delphi

Чтобы понимать назначение имеющихся в современном Delphi вещественных типов данных, полезно знать историю их появления, поэтому начнём с типов данных, которые имелись в наличии у предшественника Delphi — Borland Pascal.

Тип данных	Размер, байт	Примерный диапазон	Точность, десятичных цифр
Real	6	$10^{\pm 38}$	11÷12
Single	4	$10^{\pm 38}$	7÷8
Double	8	$10^{\pm 308}$	15÷16
Extended	10	$10^{\pm 4932}$	19÷20
Comp	8	$-2^{63}+1 \ldots 2^{63}-1$	19÷20

Нетрудно заметить, что три типа в точности совпадают со стандартными вещественными типами, реализованными Intel в математическом сопроцессоре. Зачем же тогда понадобились два других? И почему один из них записан в таблице в самом начале, а другой — в самом конце?

Дело в том, что математический сопроцессор хоть и решал достаточно востребованную задачу, даже самой Intel изначально рассматривался как роскошь для учёных, выполняющих сложные математические вычисления. Собственно, именно поэтому он изначально был реализован как отдельное устройство, а не включён в состав процессора. Кроме того, стоимость такого устройства была довольно высокой.

Такое положение дел означало, что программист при написании программы не мог быть уверенным в том, что в компьютерах, где его программа будет выполняться, будет установлен математический сопроцессор. А значит, не мог и использовать его возможности. Получается, существовал единственный способ написать программу, работающую с вещественными числами, и быть уверенным, что она заработает везде: не использовать возможности сопроцессора, а выполнять все вычисления только командами самого процессора: побитовыми операциями, операциями сдвига и т.д.

В сущности, до появления аппаратной поддержки вещественной арифметики все так и делали. Но у такого подхода был очевидный недостаток: такие программы выполняли вычисления значительно медленнее. Это не проблема, если сопроцессора в компьютере нет. Но если сопроцессор есть, то такая программа просто не использовала бы его! Какое-то бесполезное устройство получается, что ли? К тому же, программная реализация была достаточно сложной, а значит, программист мог бы легко допустить десяток-другой ошибок, которые к тому же могли бы оставаться незамеченными довольно долго.

В Pascal, а затем и в Delphi всегда уделяли особое внимание надёжности программ. Поэтому-то тип Real появился в Pascal примерно в каменном веке и с тех самых пор обеспечивал работу с вещественными числами даже в отсутствие аппаратной поддержки. Именно так, используя команды процессора для работы с целыми числами. Благодаря наличию этого типа программиста избавляли от необходимости реализовывать работу с вещественными числами самостоятельно.

С появлением стандарта IEEE 754 и сопроцессоров от Intel появились и новые типы — Single, Double и Extended — поддерживаемые сопроцессором. На случай, если программа разрабатывалась для компьютеров без сопроцессора, в Borland Pascal предусмотрели специальную настройку, которая влияла на то, будут ли в машинном коде использоваться команды для сопроцессора или же новые типы будут эмулироваться программно.

Тип Comp — особенный. Он использовался для хранения исключительно целых чисел, но в намного более широком диапазоне, чем позволяли целочисленные типы данных. Дело в том, что сопроцессор умел работать не только с вещественными, но и с целыми числами: это требовалось, например, когда требовалось выполнить операцию над двумя числами, одно из которых было вещественным, а второе — целым. Результат сопроцессор, опять же, позволял записать не только как вещественное число, но и как целое.

Во времена Borland Pascal, когда самый большой целочисленный тип во всех языках программирования был 32-битным, тип Comp был настоящей находкой и огромным конкурентным преимуществом. Единственным его недостатком было то, что по внутреннему устройству он всё ещё оставался вещественным. Например, ему можно было попытаться присвоить нецелое число — при этом оно округлялось до целого. По этой причине программисту следовало соблюдать некоторые меры предосторожности, если предполагалась работа с действительно большими целыми числами, т.к. могла происходить потеря точности, однако в пределах заявленного диапазона результаты получались корректными и это было математически доказуемо.

Теперь перейдём к вещественным типам данных в Delphi.

Тип данных	**Размер, байт**	**Примерный диапазон**	**Точность, десятичных цифр**
Real48	6	$10^{\pm 38}$	11÷12
Single	4	$10^{\pm 38}$	7÷8
Double	8	$10^{\pm 308}$	15÷16
Extended	10	$10^{\pm 4932}$	19÷20
Comp	8	$-2^{63}+1 \ldots 2^{63}-1$	19÷20
Currency	8	$-9 \cdot 10^{15} \ldots 9 \cdot 10^{15}$	19÷20
Real	Эквивалентен либо Double, либо Real48.		

Появление Delphi совпало с переходом от 16-битных к 32-битным процессорам.

Во-первых, это позволило реализовать полноценный 64-битный целочисленный тип средствами самого процессора, поэтому надобность в типе Comp отпала. Тем не менее, поскольку с Borland Pascal было написано огромное количество программного обеспечения и вспомогательных библиотек, этот тип данных и по сей день поддерживается в Delphi — чтобы крупные проекты, в которых замена этого типа данных на другой потребует больших трудозатрат, в том числе на тестирование, могли продолжать компилироваться в Delphi без изменений в исходном коде. Обратная совместимость.

Во-вторых, теперь сопроцессор перестал быть отдельным устройством, а стал частью самого процессора. По этой причине классический тип данных Real был переименован в Real48 и также поддерживается только для обратной совместимости. Но поскольку в мире уже существует большое количество кода, использующего этот тип под именем Real, название Real также сохранилось.

В Delphi по умолчанию Real эквивалентен Double. Для подавляющего большинства программ эта подмена совершенно незаметна и выражается лишь в повышении быстродействия. Однако для тех случаев, когда конкретные размеры типа или особенности вычислений важны для программы, имеется настройка компилятора, позволяющая заставить его под именем Real понимать старый-добрый (но более медленный!) Real48.

Как написать программу и не стать должником

Во многих программах приходится иметь дело с подсчётом денег. Представим себе, что Вы разрабатываете программу для выполнения банковских операций. Со школы каждый из нас знает, что сумма «12 рублей, 3 копейки» — это дробное число 12.03, так ведь?

Если Ваши руки уже потянулись написать название одного из вещественных типов, задумайтесь! Погрешность представления — совершенно реальная проблема вещественных типов. А ещё погрешности иногда чудесным образом комбинируются так, что из совсем небольших, копеечных (такой вот каламбур), превращаются в очень значительные. Здесь сложили, там перемножили — и вот Ваша программа уже перечисляет хитрому злоумышленнику погрешность вычислений в размере нескольких миллиардов долларов. Отгадайте, кто потом будет выплачивать эти деньги?

Для вычислений с деньгами вещественные типы использовать нельзя!

В большинстве случаев простым решением проблемы будет просто хранение денежных сумм в виде целых чисел. Например, не 12.03 рубля, а 1203 копейки. Целочисленные типы данных не имеют проблем с погрешностями, поэтому подведут, только если суммы станут совсем уж астрономическими. 64-битных типов в наше время пока ещё хватает.

Тем не менее, иногда вычисления с деньгами становятся сложнее и выкручиваться с использованием целочисленных типов становится неудобно. В этом случае в Delphi на помощь приходит тип данных Currency. Как можно догадаться из вышеприведённой таблицы, его внутреннее представление — 64-битное целое число со знаком, однако с точки зрения программы младшие 4 десятичные цифры этого числа находятся в дробной части. Т.е. переменные этого типа ведут себя, как дробные числа с 4 десятичными цифрами после запятой, но без погрешностей. Красота!

[1] https://people.eecs.berkeley.edu/~wkahan/ieee754status/754story.html