Gem #7: Красота числовых литералов в языке Ada

Автор: Franco Gasperoni (AdaCore)

Краткое содержание: Gem Ada #7 – Известно ли вам, что волшебство языка Ada распространяется на целые и действительные числа (на самом деле, на числовые литералы). Читайте дальше, чтобы узнать больше о данном Gem Ada..

Давайте начнем…

В необычно жаркий день конца лета несколько лет назад я вошел в Институт математических наук Куранта. Это был мой первый класс в Нью-Йоркском университете и мой первый день с Ada. Духота в классе и кондиционер в ремонте.

Ветер облегчения в этот день исходил из чего-то простого и элегантного: числовые литералы в Ada. Мне повезло, что мой класс языков программирования начался с чего-то такого крутого. Первое, что поразило меня, – это возможность использовать символы подчеркивания для разделения групп цифр. Я всегда полагал что запись числа в виде.

```
3.14159_26535_89793_23846_26433_83279_50288_41971_69399_37510
```

более читаемое и менее подверженное ошибкам восприятия, чем

```
3.14159265358979323846264338327950288419716939937510
```

А какое Ваше мнение? (Я хочу, чтобы подчеркивания были разрешены при вводе номера кредитной карты в Интернете.)

Это - только холодное начало. Давайте говорить об основании счисления в литералах. Я никогда не понимал объяснения странного и не интуитивного метода записи числового литерала, для определения основания счисления литералы в языке программирования C, где у Вас есть только 3 возможных основы: 8, 10, и 16 (почему нет основания 2?). Кроме того, требование, чтобы числа в основании 8 предшествовал ноль, воспринималось мной подобно плохой шутке над нами программистами, кажется, что мы плохо программируем (какие значения принимают числовые литералы 0210 и 210 в C?).

Приятным сюрпризом в этот душный день конца лета, я узнал, что Ada позволяет любую основу от 2 до 16 и что мы можем записать десятичное число 136 в любой из следующих нотаций

```
2#1000_1000#     8#210#     10#136#     16#88#
```

Исходя из аппаратной платформы микроконтроллера, где мои устройства ввода / вывода были отображены в память, мне понравилась возможность писать:

```
Lights_On  : constant := 2#1000_1000#;
Lights_Off : constant := 2#0111_0111#;
```

и иметь возможность включать / выключать свет следующим образом:

```
Output_Devices := Output_Devices  or   Lights_On;
Output_Devices := Output_Devices  and  Lights_Off;
```

Конечно, мы можем также использовать записи, чтобы сделать вышеупомянутое еще более изящно, и я оставляю это будущим статьям в GEM (добровольцы есть?).

Назад к основанным литералам. Понятие основы в Ada допускается при записи экспоненты и это особенно приятно. Например, мы можем записать:

```
Kilobinary  : constant := 2#1#e+10;
Megabinary  : constant := 2#1#e+20;
Gigabinary  : constant := 2#1#e+30;
Terabinary  : constant := 2#1#e+40;
Petabinary  : constant := 2#1#e+50;
Exabinary   : constant := 2#1#e+60;
Zettabinary : constant := 2#1#e+70;
Yottabinary : constant := 2#1#e+80;
```

В записи системы счисления литерала на основе экспонента, так и базовая часть, использует стандартную десятичную нотацию и определяет порядок базы, которую должен умножить основанный литерал, чтобы получить окончательное значение. Например: 2#1#e+10 = 1 x 210 = 1_024 (в десятичной системе счисления), тогда как 16#F#e+2 = 15 x 162= 15 x 256 = 3_840 (в десятичной системе счисления).

Числа системы счисления одинаково хорошо применяются к записи литералов в формате real. Мы можем, например, записать:

```
One_Third : constant := 3#0.1#;  --  same as 1.0/3
```

Пишем ли мы 3#0.1# или 1.0/3, или даже 3#1.0#e-1, Ada позволяет нам определять точно рациональные числа, для которых не могут быть записаны десятичные литералы.

Это подводит нас к последней приятной особенности Ada для этой статьи в GEM. Как сказал бы Боб Дафф: Ada имеет открытый набор целочисленных и реальных типов.

В результате числовые литералы в Ada не несут с собой их тип, как в C. Фактический тип литерала определяется из контекста. Это особенно полезно для предотвращения переполнений, недочетов и потери точности (подумайте об ошибочной записи **32l** в C, что сильно отличается от **321**).

И это еще не все: все постоянные вычисления, выполненные во время компиляции, выполняются бесконечной точностью, будь они целыми или реальными. Это позволяет нам писать константы с любыми размерами и точностью, не беспокоясь о переполнении или потере значимости. Мы можем, например, написать:

```
Zero : constant := 1.0 - 3.0 * One_Third;
```

и быть гарантировано, что постоянная ноль действительно имеет значение ноль. Это очень отличается от написания:

```
One_Third_Approx : constant := 0.33333333333333333333333333333;
Zero_Approx      : constant := 1.0 - 3.0 * One_Third_Approx;
```

где Zero_Approx действительно 1.0e-29 (и это обнаружится в Ваших числовых вычислениях.) Вышеупомянутое довольно удобно, когда мы хотим записать части без любой потери точности. С учетом предыдущих строк, мы можем записать:

```
Big_Sum : constant := 1           +
                      Kilobinary  +
                      Megabinary  +
                      Gigabinary  +
                      Terabinary  +
                      Petabinary  +
                      Exabinary   +
                      Zettabinary;
```

```
Result : constant := (Yottabinary - 1) / (Kilobinary - 1);
Nil : constant := Result - Big_Sum;
```

И будет гарантировано, что Nil будет равен нулю.

Но я увлекшись элегантностью числовых литералов Ada и почти забыл о дате нашей следующей статьи GEM, которая выйдет 2#10_10# сентября (извините, я не мог сопротивляться :).

Счастливого лета коллеги разработчики.

Обсуждение…

1. 28 июня 2007 года в 7:32
Кристоф Грин сказал:
Франко,
Сегодня, в день Господа, 7 # 40.6.5565 #, я должен сказать вам печальную истину, что ваши цифры, подобные Килобиту, ошибочны.
Kilobyte = 1 kB = 10 # 1 # e3 B, всегда,
видеть .
Вы имеете в виду Kibibyte = 1 KiB = 2 # 1 # e10 B.
Обратите внимание на капитализацию: k kilo, Ki kilobinary.
Это действительно имеет значение, если у вас 1 гигабайт или 1 ГБ. Разница составляет 7 # 1_553_536_512 # B.
Кристоф ;-)

2. 28 июня 2007 года в 7:38
Кристоф Грин сказал:
К сожалению, ваша программа съела адрес NIST http://physics.nist.gov/cuu/Units/prefixes.html
Потому что я включил его в «меньшие» - «большие» скобки. См. Там бинарные и десятичные префиксы.

3. 28 июня 2007 года в 18:33
Франко Гасперони сказал:
Когда я узнал все это - в конце 70-х - килобайт составлял 1_024 байта. В то время я нашел такой симпатичный, а также немного запутанный.
Спасибо, Кристоф, за то, что вы меня свели на этот счет.
Интересно, является ли это обычной практикой в эти дни
Говорить о килобайтных байтах (или кибибытах :)
Кстати, для всеобщего блага на странице:
http://physics.nist.gov/cuu/Units/prefixes.html
NIST пишет:
Поскольку префиксы СИ строго представляют степени
Из 10, они не должны использоваться для представления полномочий
Из 2. Таким образом, один килобит, или 1 кбит, составляет 1_000 бит и
Не 210 бит = 1_024 бит. Чтобы облегчить эту двусмысленность,
Префиксы для двоичных мультипликаторов были приняты
Международная электротехническая комиссия (МЭК)
Для использования в информационных технологиях.
Видеть
http://physics.nist.gov/cuu/Units/binary.html
Для деталей.

Я изменил имена переменных, так что теперь вместо Kilobyte я использую Kilobinary :)

4. 29 июня 2007 года в 8:39 утра

Кристоф Грин сказал:

«Когда я узнал все это - в конце 70-х - килобайт составлял 1_024 байта. В то время я нашел такой симпатичный, а также немного запутанный.

...

Интересно, является ли это обычной практикой в эти дни

Говорить о килобайтных байтах (или кибибытах :) "

Нет, я никогда не слышал, чтобы кто-то использовал эти префиксы, что печально из-за путаницы, которую вы упоминаете.

Поэтому я начинаю распространять (старые) новости :-)

5. 29 июня 2007 года в 12:44

Мэтью Хейни сказал:

Конвенция, которую мы используем в США:

Kb = 1000

Kb = 1024

Mb = 1000 ^ 2

Mb = 1024 ^ 2

и т.д

6. 29 июня 2007 года в 13:21

Кристоф Грин сказал:

Не так хорошо, если вы считаете, что капитализация обычно рассматривается как нечто необязательное.

4 mS - Millisiemens, а не миллисекунды (что подразумевается, и я часто вижу).

И на некоторых устройствах есть (все еще) символы нижнего регистра.

Итак, что означало бы 10 МБ?

7. 1 августа 2007 года в 16:37

Франсиско Дж. Монтойя сказал:

Привет, Франко.

Я не уверен, правильно ли я понял «бесконечную точность» во время вычислений, связанных с литералами.

В линии:

Zero: constant: = 1.0 - 3.0 * One_Third;

, Zero гарантируется равным 0.0 из-за характера операндов правой стороны «: =» или характера левого аргумента (поскольку Zero - это именованный номер, а не типизированная константа) или По обоим причинам?

Я имею в виду, было бы так же 0,0, если бы Zero были константой с плавающей запятой, а не именованным номером?

Спасибо за вашу статью в GEM :-)

-

Франсиско Дж. Монтойя

8. 4 сентября 2007 года в 1:28 утра

Вилле Витт сказал:

Уважаемый господин Франсиско Дж. Монтойя:

Нуль гарантированно равен 0, потому что 3.0 * One_Third - 1 точно. Это должно показать, что One_Third действительно 1/3, а не (с конечными десятичными знаками 3) 0,333

Что такое 0.333 + 0.333 + 0.333? Это имеет значение 0.999.

Но если вы имеете в виду 1/3 + 1/3 + 1/3, вы подозреваете, что результатом будет 1. Ада позволит вам перестать интересоваться количеством бит целевой архитектуры и вместо этого объяснить компиляции того, что вы на самом деле имеете в виду.

Искренне Вилле Витт (.net)

P.S .: Извините, если я ошибался в отношении вышеизложенного - я сам совершенно новичок в Ada и восхищен этим. Нам нужны люди, чтобы обновить их учебники Ada-on-Linux!