Алексей Канатов, [04.05.18 08:59]

Еще несколько мыслей, ну а далее от тебя зависит их принять или отмахнуться....

Итак, о типах опять. Пусть есть рутина

foo (T1, T2, ... Tn):Tn+1 do ... end

И тут в явном виде присутствуют три части.

1. Имя рутины. Оно может иметь алиасную часть

2. Сигнатура: перечень типов аргументов и тип возвращаемого результата. Такой перечень тоже является кортежем, точнее, двумя – первый кортеж описывает типы аргументов, а второй тип возвращаемого результата, если рутина – функция. Причем надо отметить, что оба этих кортежа опциональны.

Это выглядит разумно (правда, банально, но для руководства по языку сойдет). Единственное соображение (я тебе его высказывал): уподобление параметров рутины кортежу оставь для умных статей. Я против того, чтобы эта аналогия использовалась в определении языка. Это точно снизит привлекательность языка как инструмента общего назначения. А в статьях можно продолжать увлеченно строить классные схемы и красиво вокруг них плясать на радость тонким ценителям. Первым буду аплодировать.

3. Наш любимый супер-блок, который задает тело рутины, опционально с пред и пост условиями и с обработчиками исключительных ситуаций.

А теперь зададимся вопросом: а какого типа данная рутина? Ведь если мы хотим на базе рутины создать объект, он должен иметь тип. Так - какой он? Ответ прост - это рутинный тип, настроенный на опциональные кортежи типов аргументов и типа возвращаемого результата.

Первое замечание по существу. «На базе рутины создать объект» - что это такое??? Странная логика. (Разве что это у тебя скоропись.) **Мы хотим ввести в язык рутинный тип и, следовательно, мы хотим уметь создавать объекты рутинного типа** – вот как надо формулировать. Этот объект может не иметь никакого значения (дальше у тебя есть пример), а может в качестве значения получить объект – какую-нибудь рутину.

**Так вот рутинный объект – это не рутина. Это отдельная сущность, которая строится на базе рутины. Можно сказать, что рутинный объект – это дескриптор рутины.**

В нашем случае это тип

**routine** T1,T2... Tn Tn+1

Вот имя этого типа.

Следующее замечание. Рутины у нас задаются традиционно: параметры в скобках через запятую, тип результата за скобками после двоеточия. Для рутинного типа ты предлагаешь другой синтаксис, уже почему-то без скобок и двоеточий, но с замечательным словом **routine**. Зачем??

**Я не предлагаю другой синтаксис! Просто я определил, что такое рутинный тип – это признак рутинного типа routine а далее перечень типов аргументов и тип возвращаемого результата, что и записал**

По-хорошему вот как должна выглядеть запись рутинного типа:

(T1,T2,...,Tn): Tn+1

Отсюда следствие, что если есть 2 рутины с разными именами, но одинаковыми сигнатурами, то их типы совпадают.

Это НЕ следствие. Из предыдущих твоих слов ничего подобного не следует. На самом деле, утверждение, что у двух рутин с одинаковыми сигнатурами типы совпадают – это постулат языка. Через этот постулат мы вводим в язык понятие рутинного типа.

**Можем спорить до опупения, но если из данного мной определения рутинного типа следует, что имя рутины не является частью типа, то значит 2 рутины с одинаковыми сигнатурами и разными именами имеют один тип. И вводится понятие рутинного типа не через ‘постулат‘, а через то определение, что я дал.**

Второе утверждение, что если есть такая сигнатура, которой конформны две сигнатуры рутин с разными именами, то типы этих двух рутин конформны рутинному типу, построенном на базе такой общей сигнатуры.

Да, это очевидно. Согласен.

Для простоты тип рутины - это ее сигнатура, которая используется во всех тех местах, где используются типы. А так как сигнатура есть кортеж, то нам чисто синтаксически надо различать просто кортежный тип и рутинный тип на базе кортежа. Вот для чего и предлагается использовать ключевое слово **routine** или более короткое обозначение @.

Чтобы «чисто синтаксически» различать кортеж и рутинный тип, достаточно было бы твоего слова routine:

a: ?(x1: Integer, x2:Boolean) // тип кортеж  
b: ?**routine**(x1: Integer, x2:Boolean) // рутинный тип

и не нужно никаких закорючек.

**Неужели я так непонятно пишу :-( Я же черным по белому написал “** или **более короткое** обозначение **” – что тут непонятно? Никто не хочет писать идиотские длинные слова типа PROCEDURE, FUNCTION, IMPLEMENTATION MODULE, PACKAGE BODY, поэтому я и предлагаю рассмотреть вариант значка …**

Замечу, что я предпочел бы выкинуть слово **routine** вообще. Утверждаю, что без него можно обойтись. Для обозначения рутины без возвращаемого значения я бы предложил что-то вроде:

b: ?(x1: Integer, x2:Boolean):**none**  
 // рутинный тип без возвращ.значения

**Вместо ключевого слово или значка, который задает тип рутинный ввести ключевое слово none … И тогад заставить его писать для каждой процедуры, тогда уж лучше void, а то так Адой пахнет … Примеры на которые тебе надо ответить чтобы показать как обойтись без спецификатора рутинного типа у тебя в ленте Телеграмма**

Еще раз примеры

r1 (T1, T2) **do** ... **end**  
r2 (T1, T2) **do** ... **end**

r1 и r2 - это две разные рутины, рутинные типы которых совпадают, т.e. они имеют один тип. Отсюда практический вывод: рутинные объекты на базе r1 и r2 могут быть присвоены, одной переменной, которая имеет тип **routine** T1, T2.

Всё правильно. Только опять корявое «на базе». Есть объект рутинного типа. Ему присваивается значение (сущность) рутинного же типа. И все.

var: ? **routine** (T1,T2)

Так со скобками или без? ☺

**В коде программы я предлагаю со скобками так как это кортеж, в свободном тексте мне не всегда было удобно на клавиатуре мобильного телефона все время переключаться на другой экран чтобы набрать эти чертовы скобки**

Вот такая переменная

var := **routine** r1  
var := **routine** r2

Вот здесь наше расхождение и начинается. ЗАЧЕМ здесь слово **routine**? Сущности r1 и r2 и так имеют тот же тип, что и переменная var. Зачем добавлять это слово? (Впрочем, я знаю для чего: чтобы извести круглые скобки для одного частного случая, когда рутина не имеет параметров. Выглядит немного смешно, правда ☺.)

**Я это уже объяснял. Если есть рутина которая возвращает рутинный объект, то тогда не надо писать routine или @**

**Полная запись**

**var := new routine r1**

**var := new @ routine r2**

**Здесь создается рутинный объект на базе рутины r1 и r2. Я не знаю , что еще тут добавить. Да в сотый раз повторю – есть рутина – это то, что мы пишем в тексте программа, а есть рутинный объект, который создается на ее основе. Это две разные сущности.**

Еще раз: пусть есть рутина r1 с некоей сигнатурой. Мы (и ты, и я) трактуем эту сущность как имеющую определенный тип. Выше ты это пишешь. Такой же тип может иметь некоторая переменная. Чтобы задать для этой переменной такой же тип, предлагается некоторая нотация (забудем пока о наших с тобой синтаксических различиях в этой нотации). Если переменная и упомянутая сущность (рутина как рутинный объект) имеют один и тот же тип, то переменной можно присвоить эту сущность. ВСЁ!! Ничего больше не требуется.

В точности эта же самая логика используется в других случаях. Литерал 1 – это сущность (объект, константа) целочисленного типа. Есть способ задать переменную, имеющую тот же тип, что и литерал 1. Если и переменная, и литерал имеют один и тот же целочисленный тип, то переменной можно присвоить объект-литерал:

var: ?Integer  
…  
var := 1

(Мы же не нуждаемся в этом присваивании в каком-нибудь дополнительном крючке или служебном слове?)

**Потому что тип совпадает … И если бы мы все вызовы рутин записывали как** call **foo, то тогда да, а так как большинство операции это вызов рутины то удобно не писать лишнее call, а для гораздо более редкой операции создания рутинного объекта как раз использовать некоторый префикс …**

Идём дальше. Для объектов (констант вроде 1 или для переменных) целочисленного типа определено вполне конкретное множество операций. В некоторых языках они задаются изначально (если целочисленный тип считается встроенным), в СЛанге набор операций над целыми задается в виде рутин-операций в юните Integer.

Так или иначе, объекты типа Integer можно складывать, умножать и т.д. ***Точно так же***, для объектов рутинных типов (неважно, для рутин как таковых, то есть, рутин-констант, или для переменных рутинных типов) имеется определенный набор операций. Конкретно, для них определена единственная операция – ***вызов*** (активация). (И ты, кстати, прекрасно это выразил в последних строках своего бессмертного core.slang).

***Традиционно***, вызов рутин обозначается как имя рутины или имя переменной рутинного типа, за которым в скобках следует список аргументов вызова. Если аргументов нет, но остаются пустые скобки. Ну вот я и предлагаю следовать традиционному способу записи.

***Одной фразой:***Отличие foo от foo() заключается в том, что foo – это ***имя объекта*** (рутинного типа), а foo() – это обозначение ***действия*** над объектом foo, которое заключается в активации его тела.

(В некоторых языках для обозначения действия (вызова) используется специальное служебное слово: **call** foo. В этом случае скобки можно было бы опустить. Но в любом случае разделять объект как таковой и действие над ним – для меня это… не просто фундаментально, но и попросту очевидно.)

Напомню, что мы не можем написать

r1 := r2

Или наоборот, так как r1 и r2 не являются переменными, это рутины,…

Это правильно, хотя и вполне очевидно. В некоторых языках прямо говорится (и нам надо бы сказать), что подпрограмма подобна константе, и, следовательно, она не может изменять свое значение. Аналогично, массивы в C/C++ считаются константами, и объявив, скажем, **int** A[10], нельзя присвоить A что-то другое.

… а с рутинами у нас есть 2 операции - это вызвать/применить рутину или построить на ее основе рутинный объект.

Для рутин имеется только одна операция – вызов (см. выше), когда рутина как бы «вскрывается». (Впрочем, можно считать, что присваивание рутинного объекта переменной рутинного типа – это тоже операция… Но в этом отношении рутины ничем не отличаются, скажем, от целых: их тоже можно присваивать, передавать в качестве параметра, и т.д.) Но операция, специфичная именно для рутин, – только одна.

Пытливый читатель может задаться вопросом: а зачем писать routine r1, если я присваиваю такой объект переменной рутинного типа. Ответ прост, так как рутина может быть функцией, которая возвращает рутинный объект. Т.е.

var := func

Это вызов некоторой функции func, которая должна вернуть объект, тип которого конформен типу var.

Ну, «пытливый читатель» будет продолжать недоумевать: почему эти отличия нельзя выразить гораздо более понятным и привычным образом? var := func – это простое присваивание, var := func() – это вызов функции, которая возвращает объект, конформный типу var? И никаких служебных слов и закорючек.

**Наличие или отсутствие скобочек после имени – это понятный и наглядный способ?**

**А привычный для кого – в каких языках так сделано?**

И мы различаем синтаксически 2 операции над рутинами вызов и создание рутинного объекта.

Опять двадцать пять. Зачем повторять уже однажды сказанное? Свои возражения я уже высказал (только одна операция, специфичная для рутин).

Еще раз про целостность концепции формы создания объектов в СЛанг. В самом общем виде объект создается так

**new** Type.init(E1, E2, ... En)

Мы разрешаем сокращенные формы этой записи

Type  
Type (E1, E2, .. Еn)

А в качестве Type у нас могут быть все виды типов в СЛанге, а именно

1. UnitType тип на базе конкретного юнита  
2. Рутинный тип на базе конкретной рутины или на базе сигнатуры  
 Опять «на базе рутины». Ну нельзя так писать!

**Именно так и надо писать, потому что это так и есть.**

**(Отличный спор мужей – коммунистическое учение всесильно, потому что верно! :-))**

3. Якорный тип as this, например  
4. Кортежный тип (T1, ... Tn)

Надо еще углубиться в рутинный тип, в связи с тем, что сказано выше. Примеры рутинных типов

**routine** foo

Это рутинный тип, который такой же как у рутины foo

Зачем? У нас есть конструкция **as**. Почему не написать вот так:

foo(...): ... **do** ... **end**  
...  
v: ?**as** foo

**routine**(T1, .. Tn)

Это явное задание рутинного типа.

Уже писал выше: слово **routine** здесь совершенно не нужно.

**Не уловил ты ход моих мыслей тут. Да можно бы было в варианте routine foo написать as foo, но это не подходит для случая явного задания рутинного типа по сигнатуре – тут то нет якоря … А стоит иметь общую нотацию для рутинных типов … А не для каждого случая свою …**

Первый вариант по сути это якорный тип

**routine** **as** foo

Вот как можно его было бы написать, чтобы показать его якорную сущность. Может быть даже такую форму и оставить для большей наглядности.

Вот же опять: ну не нужно здесь routine, ни к чему она!!!

И, кстати, я бы извел это слово «якорь». Не нужно оно. Бертран увлекался придумыванием новых слов для всего на свете (я удивляюсь, почему он слово **class** оставил ☺), зачем нам преданно следовать его изобретениям?

**Ты опять забыл, что такое якорные типы и для чего они нужны?**

Когда я перечислял выше типы в СЛанге, возможно, я что-то упустил, в версии синтаксиса выписаны все виды типов ...

Поэтому из всего вышеизложенного у меня не получается принять концепцию есть скобки – вызов, а нет скобок – создание рутинного объекта.... По крайней мере, я сподобился дать развернутое описание ...

«Из всего вышеизложенного» твое желание отказаться от этой концепции ну уж никак не следует.

Алексей, в этом своем тексте ты, уж прости, не сказал ничего нового; более того, раньше ты приводил даже несколько более разумные причины (вроде следования принципу Лисков). Мне показалось, что в своих комментариях я предложил нормальный, традиционный, естественный, очевидный, удобный, компактный и какой угодно еще способ задания рутин, рутинных типов и операций над ними…

**А я не стал повторять все то, что приводил раньше, а иду дальше ….**

Алексей Канатов, [04.05.18 21:52]

Добавлю. Грамотный Бертран сознательно не стал использовать слово **routine** для создания рутинных объектов. Он сказал по-другому. Есть механизм для отложенного вызова рутины. Т.е. мы создаем некоторого агента, который позже может вызвать рутину. Поэтому Бертран пользуется ключевым словом **agent**. На первый взгляд непривычно и непонятно. Почему указатель на функцию называть агентом😉

У него под агентом и понимается вульгарный указатель на функцию. Да, «агент» - как бы своего рода представитель функции, посредством которого ее можно впоследствии активировать. Ничего нового за исключением нового (и явно не самого очевидного) служебного слова, кокетливо прикрывающего всем известную семантику.

Впрочем, в оправдание Бертрана могу заметить, что в Аде сделано ничуть не менее нелепо (правда, с б***о***льшим вкусом): ровно для этих целей было введено служебное слово **access**. (Но Майеру надо было непременно сказать ***своё*** слово. ☺)

Алексей Канатов, [04.05.18 21:56]

Я понимаю, что термин routine может путать так как я все время говорю о routine object.

new routine object (T1, ... Tn) Это конечно по смыслу точно но очень ДЛИННО .... Поэтому закорючка типа @ мне нравится больше ... Она бросается в глаза, ее нельзя поставить по ошибке, если ставишь значит разобрался что это ....

продолжаем

Доброе утро. Я быстро прочел твои комментарии зеленым. Быстро покажу, почему я так упорно пишу создавать рутинный объект на базе рутины.

Рутина - это имя, сигнатура и тело ( которое может быть супер блоком, абстрактным телом и внешним телом), т.е. по сути это структура внутри которой юнион, или в Паскалевских терминах запись с вариантами.

unit Routine

     name: String

     alias:?String

     signature:?Signature

     body:?RoutineBody

end

Так вот когда описываем атрибут рутинного типа, то по сути мы просто пишем

rtnObject :?Routine

Вот тебе 3 случая описания переменной  рутинного типа.

1. rtnObject is routine foo

2. rtnObject is routine goo (T1, ... Tn)

3. rtnObject : ?routine (T1, ... Tn)

1. Случай если перегрузки имен нет и переменная rtnObject будет иметь тип рутины foo с ее сигнатурой.

2. Перегрузка имен присутствует и тут явно задается какая сигнатура должна использоваться для определения типа.

3. Это просто задание рутинного типа.

В случаях 1 и 2 переменной сразу присваивается значение, а в 3ем случае переменная неинициализированная. И ей позже может быть присвоен любой рутинный объект с сигнатурой конформной заданной.

Напомню, что кортеж могут иметь как тегированные, так и не тегированные поля, поэтому нет никакого смысла тегировать  перечень типов.

Сейчас я попробую записать все тоже самое в твоей нотации

1. rtnObject is foo

2. rtnObject is goo ??? Что писать дальше или ты требуешь, что запись

goo (f1:T1; f2: T2 ... fn:Tn) есть рутинный объект, а

goo (T1, T2, ... Tn) это вызов?

3. rtnObject: ? Как тут записать имя рутинного типа, ключевого слова у тебя нет, рутины тоже, что тут писать?

Для большей наглядности попробуй в своей нотации написать рутину с параметром типа рутина ....

foo (rtnObject : routine (T1, T2, ... Tn)) do ... end, так пишу я….