# Спорные моменты для дальнейшей дискуссии

1. hidden - нет такого ключевого слова - используем [“{” [this| UnitTypeName {“,” UnitTypeName} ] “}”]  для управление видимостью члена юнита.

Поясни на примерах

**unit** A

foo /\* read-only(for attribute)/call-only(for routine) visibility for clients,

read-write(for attribute)/call-only(for routine) visibility for descendants \*/

{} goo /\* hidden from all clients,

read-write(for attribute)/call-only(for routine) visibility for descendants \*/

{A, B, C} too /\* hidden from all clients except A, B, C (exported to A, B, C),

read-write(for attribute)/call-only(for routine) visibility for descendants \*/

// NOTE: I do not insistent on this – it is just so natural in terms of syntax

{**this**} boo /\* hidden from all clients,

hidden from all descendants \*/

/\* Current unit accessible thru **this** has all the access to all its members except inherited as {this} form its parents\*/

**end** // A

EZ: Первое. Мне эти идиотские фигурные скобки активно не нравятся. У Бертрана не слишком хороший вкус в этом отношении.

Второе (главное). Задавать, кому этот класс может что-то предоставлять, а кому-то нет - это кажется методологически порочным. Когда ты проектируешь библиотечный класс, ты вообще не можешь предполагать кто этот класс будет юзать. Он должен быть спроектирован без знания о своих клиентах и потомках.

AK: Про управление областями видимости членов. Мы же обсуждали, что либо нам надо 2 ключевых слова, а может и больше, что управлять этим, и идея использовать {} была ведь принята. Мне нравится вариант, что нет ключевых слов для этого. Логично. Спорить в 100й раз, что это неправильно, что юнит перечисляет для кого экспортирует члены, я считаю бессмысленным, тем более, что я ЯВНО писал, что не настаиваю, просто легко поддержать. У меня есть доводы для чего restricted export может работать, но просто не хочу спорить. В Америке в свое время были надписи White only - это классический пример restricted export. Staff Only, Authorized Personal Only, видел такие таблички ... Спорь дальше 😊

**Основной вопрос – есть 4 теоретических варианта видимости члена юнита (они ВСЕ представлены в примере выше), поэтому первый момент согласен ли ты что их 4? Второй сколько из них надо иметь в Сланге? И третий какую нотацию ТЫ для этого предлагаешь?**

2. Ключевое слово routine используется для задания рутинного типа, а для создания рутинного объекта ты предлагаешь использовать имя рутины без круглых скобок, но как тогда различать рутины если они различаются только сигнатурами, Как взять рутинный объект?

foo (T1, T2) **do** ... **end**

foo (T1, T2, T3) **do** ... **end**

rtn **is** foo // неоднозначность - ошибка компиляции или как?

Это не просто неоднозначность – по моему мнению, это семантически неправильно **(Ты же сам хотел – со скобками вызов, без скокбок рутинный объект).**

Я так никогда не хотел, ты меня с кем-то путаешь. Если угодно, наличие круглых скобок – это характеристический признак рутин вообще. **{AK: Это неверно! Рутина – это имя, сигнатура (возможно пустая) и тело (3 варианта do…end, abstract, foreign)}** Объявление со скобками, и вызов со скобками. То есть, если рутина foo в данном контексте единственная, тогда, скрепя сердце, можно записать ее имя без скобок. Но если их несколько, то чтобы их различить, приходится писать ее имя с типами параметров.

И вообще: у нас после is всегда идет инициализация, то есть выражение. foo(T1,T2) – все что угодно, но только не выражение. **{AK: Это неверно! Это вызов рутины foo с параметрами T1 и T2– можно написать new T1.init(), new T2.init()**

**Я понимаю в чем проблема. Вот смотри**

a **is** foo (e1, e2, … en) /\* ты интепретируешь это как вызов с аргументами n выражений\*/

b : **routine** foo (T1, T2) **is** … /\* я не знаю как ты \то интерпретируешь, для меня однозначно – b – это объект рутинного типа сигнатура (tuple) этой рутины имеет 2 параметра с типами T1 и T2, вызова тут никакого нет\*/

Отсюда вывод, что раз Зуев путается, то и другие могут путаться и надо разводить синтаксически кортеж и вызов .... Сейчас и для первого и для второго идут ()…. Честно говоря очень не хочется ... Вот и дай мне ответ, что легче понять или вводить что-то типа <T1, T2, T3, e1> - это кортеж. Почти как у Мейера ... Одна из идей СЛанга, что у нас есть многозначность ... () – это кортеж, foo() – это вызов рутины foo, **routine** (T1, T2) – это рутинный тип, **routine** foo (T1) – это рутинный объект (частный случай выражения) И везде есть чертовы круглые скобки, они ведь просто образуют группу и не более того, а пустая группа () эквивалента пустому месту ... поэтому можно () не писать ... Ну и конечно примеры на все это

Да, я пропускаю в описании рутин иногда имена аргументов, а оставляю просто типы тогад когад мне имена аргументов не важны – возможно это тоже путает но оно однозначно

rtn (T1; a: T2; T3; b: T4) **do** … **end**

Рутина rtn имеет 4 (ЧЕТЫРЕ) аргумента, типы соответсвтенно T1, T2, T3, T4. У первого и третьего имен нет, обращаться можно тоолько по позиции ну типа \_1 и \_3, у второго и четвертого имена соответственно a и b. После такого объяснения надеюсь непонимания будет меньше ...

Да у меня в голове идет процесс и не все что есть в голове отражено на бумаге, но видишь я стараюсь ... Итак примеры про кортежи и рутины

foo: T0 **do** … **end**

foo (T1): T0 **do** … **end**

a **is** foo // это вызов функции foo

b **is routine** foo /\* это означивание атрибута b рутинным объектом, который ассоциирован с foo. Т.е. взять указатель на функцию и присвоить переменной, но все типизировано. Т.е. тип переменной b – это **routine** ():T0, так как именно такой рутинный тип имеет foo без аргументов \*/

c: **routine** foo **is** **routine** foo /\* так как routine foo стоит после : - это тип! После is выражение этого типа\*/

d: **routine** (): T0 **is** **routine** foo

e: ?**routine** (): T0 // Это локал который неинициализирован, но тип означен

e := c

Другими словами наличие ключевого слова routine четко показывает с чем мы имеем дело – с рутинным типом или с сущностью (переменной, атрибутом) рутинного типа. Я понимаю, что момент сложный для понимания, и поэтмоу мне кажется, что именно наличие ключевого слова четко позволяет показать семантику ....

Я не хочу спорить что ты говорил, а ты будешь говоить что нет. Скажу так – некто мне расскзаал, что в некоторых языках программирования разница между функциональным типом и вызовом опрееделеятся по наличию (), на что я тут же привел примеры когад возникает неоднозначноть в условия перегрзуки имен. Поэтому если Зуев (а не некто ☺) предложит вариант, как ввести рутинные типы без неоднозначностей и ключевого слова, то Канатов с удовольствием такое предлжение прочитает и даже постарается его понять. А пока есть вариант с ключевым словом routine, которое в зависимости от контекста (как многое в Сланге) участвует как в задании рутинного типа, так и рутинных выраженеий, что было проиллюстрировано примерами выше.

**}**

Объявляя объект rtn, ты хочешь, очевидно, задать его нач. значение? (**НЕТ! НЕ ОБЯЗАТЕЛЬНО. Например a: ?routine foo(T1, T2) – т.е. я прост описал a как рутинную сущность с типом как у рутины foo (T1, T2))**

**Что такое вопросительный знак впереди?**

– ну так пиши rtn is foo(T1,T2). Какие проблемы? **(ЭТО ВЫЗОВ foo)**

**Какой же это вызов foo, если ты ТИПЫ задал в позиции параметров????**

(А, кстати, для задания типа можно было БЫ написать rtn: (T1, T2), ЕСЛИ БЫ эта запись не была синтаксически тождественной кортежу…)

Но так или иначе, объявление с инициализацией, как я написал, - вполне ОК.

rtn ( <сколько параметров передавать и что будет вызвано????> )

Здесь не проблема: ты хочешь ВЫЗВАТЬ **(НЕ ХОЧУ Я ВЫЗЫВАТЬ, по сути я беру указательн на функцию в терминах С, а в термниах СЛанга делаю рутинный объект ассоциированный с конкретной рутиной – rtn1 is new routine foo (T1, T2) – вот полная форма)** определенную foo, которой была инициализирована rtn – ну так подбрасывай ей аргументы подходящих типов и в подходящем количестве…

у меня пока это однозначно работает через ключевое слово ...

rtn1 is routine foo (T1, T2) rtn1 is foo(T1,T2)

rtn2 is routine foo (T1, T2, T3) rtn2 is foo(T1,T2,T3)

Нет проблем с однозначностью.

**Еще как есть!**

**rtn1 is foo(T1,T2) – это вызов foo c двумя параметрами**

**rtn2 is foo(T1,T2,T3) – и это в другой foo у которй 3 параметра**

ТАК… ПОХОЖЕ, У НАС НЕ ПОЛУЧАЕТСЯ НОРМАЛЬНОГО ДИАЛОГА. СИСТЕМА СО СПОНТАННЫМИ НАБРОСКАМИ, ИЗОБИЛУЮЩИМИ СОКРАЩЕНИЯМИ, ПОНЯТНЫМИ ТОЛЬКО ИХ АВТОРУ – И ПОПЫТКИ ОТВЕЧАЮЩЕГО РАЗГАДАТЬ НАМЕРЕНИЯ ОРИГИНАЛЬНОГО АВТОРА – **НЕ РАБОТАЕТ**.

ЕСЛИ ТЫ ХОЧЕШЬ НОРОМАЛЬНОГО ПРОЦЕССА ПРОЕКТИРОВАНИЯ, ИЗЛОЖИ СВОИ ПРЕДЛОЖЕНИЯ ПО ТЕМ ИЛИ ИНЫМ РАЗДЕЛАМ СИСТЕМАТИЧЕСКИ, В ВИДЕ **СВЯЗНОГО ТЕКСТА** С ПРИМЕРАМИ (НАПИСАННЫМИ МОНОШИРИННЫМ ШРИФТОМ).

Например, оказывается, у тебя запись foo(T1,T2) – это вызов (????), а Т1 и Т2 – это, оказывается обозначения аргументов???

В общем, изложи свои взгляды на рутины, рутинные типы, рутинные объекты, вызовы рутин – **в систематическом виде**. Прочитаю с великой радостью.

Т.е. получается что слово routine используется как для указание на рутинный тип, так и на создание-получение объекта рутинного типа ...

Не все твои нюансы я схватил, но мнение мое – **допиши какое твое мнение! Еще раз сформулирую, что у меня не получается рутинные типы без неоднозначностей ввести без дополнительного ключевого слова .... С дополнительным ключевым словом неоднозначностей нет ...**

3. Что до сих не решено - так это подходящее слово вместо super, так как смысл - это обращение к версии данной рутины из непосредственного родителя юнита.

Не обязательно рутины – любого члена родительского юнита.  
Вообще, слово super мне не очень понятно в условиях множественного наследования. Какой именно super имеется в виду?

**Еще раз, внутри рутины, которая является переопределением, можно вызвать ее ближайщую предшествующую версию, если таких версий много то надо указать из какого ближайщего родителя эта версия приходит.**

Так как по сыслу мы хотим вызвать предыдущую версию – т.е. former version, синонимами к слову former – будет prior, old. Вот между этими двумя словаим и идет выбор ... Пока в примере ниже используется old чтоыб не увеличивать число ключевых слов. В постусловии рутин old применяется чтоыб взять “старое” значение некотрого атриюута, а в теле рутины, чтобы вызвать “старую” версию этой рутины

**Пример**

**unit** Z

foo (…) **do** … **end**

**abstract** goo ()

**end**

**unit** A **extend** Z

**end**

**unit** B

foo (…) **do** … **end**

goo () **do** … **end**

**end**

**unit** C **extend** A, B

**override** foo (…) **do**

/\* old (…) – if foo comes from both A and B then such call is ambiguous in this context \*/

A.**old** (…) EZ: А ПОЧЕМУ НЕ НАПИСАТЬ ПРОСТО **A.foo(…)**

**AK: А потому что это обращение к модулю А**

B. **old** (…)

/\* Z.**old** (…) – is incorrect as C has only A and B as parents and it knows nothing about Z \*/

**end**

**override** goo () **do**

**old** (…) /\* this call is Ok as we have only one non-abstract version of goo\*/

**end**

**end**

~~Меня, конечно, подмывает просто заменить super на precursor и закрыть тему, уж очень давно идет процесс поиска короткого, но понятного ключевого слова ...~~

~~Precursor – отвратительное слово; не понимаю, как у Майера хватило вкуса вставить в язык эту фармацевтику.~~

~~Так как есть люди, для которых семантика слова super связана с понятием класс, умница Бертран и выбрал 100% правильный английский термин – предшественник – вызови мне предшественника, а в аптеке провизор ... ☺ Именно поэтому я и хочу избавиться от super, так как само слово означает класс, а не член класса в других языках ... Но, я не смог найти слова короче, чем precursor со значением предшественник, ancestor – опять же используется для классов, а не для рутин ...~~

**AK: По-моему все уже разложено по полочкам про обращение к предыдущей версии из переопределяющей ее. И весь вопрос в ключевом слово для этого. Так как super имеет определенную семантику, которая отличается от того что тут, то предлагаю использовать old или prior**

4. Так как ты любишь вложенности, у нас есть такая вещь StatementsList: { Statement[“;”]|UnitRoutineDeclaration} - т.е. среди операторов рутины могут быть описания вложенных рутин и далее рекурсивно. Но пока в таком виде это не дает возможность внутри отдельно-стоящей рутины иметь вложенные рутины также параметризуемые - надо?

[G **init**] standAloneRtn (arg: G) **do**

         foo () **do** // Так можно

goo [G] (arg)

goo [Integer] (444)

local: G **is new** G /\* Вот тут да можно использовать Тип, котрый задается при настройке охватывающей рутины \*/

**end**

         [G] goo (arg: G) **do**

goo [Integer] () /\* Именно потому, что goo – это настраиваемая рутина ее даже внутри самой себя можно настроить на конкретный тип, даже на G тоже можно\*/

**if** false **do** goo [G] () **end**

**//**А вот настроить foo – нельзя!!! Она ненастраиваемая исходя их декларации

**end**

// А вот так пока нельзя ... Надо?

**end**

Вообще, в С++ такое возможно, только пример неудачен: для вложенной рутины, которая тоже параметризуется, надо, очевидно, выбрать для параметра другоой идентификатор – иначе просто синт.ошибка будет.

Как раз никакой ошибки не будет! Вложенная рутина может вполне настраиваться на тот же самый параметр, что и охватывающая, а может и на другой. В примере выше добавил аргументы, чтобы показать всю полноту картины ... Т.е. внутри настраиваемой рутины standaloneRtn есть 2 вложенные рутины foo и goo и в теле foo стоят разные настройки-вызовы goo…

Для полноты и стройности надо бы такое допустить, но реализовывать (да и использовать) будет довольно сложно.

Т.е. я это интерпретирую как ответ Да ☺

Если внутри рутины-шаблона есть влож.рутина, то она по умолчанию тоже шаблонная ТЕМ ЖЕ параметром, что что и объемлющая. Если же ты хочешь рассматривать влож.рутину как параметризуемую ЕЩЕ И ДРУГИМ параметром – просто напиши имя этого параметра.

Именно поэтому повторять одно и то же имя для объемлющей и для вложенной – бессмысленно

**AK: Мой ответ в теле примера выше, который я расширил...**

5. AK: Inheriting initialization procedures from parents. If one of parent initialization procedures can be used to initialize current unit there is no need to create code of init while just calls the version of init from parent. So, instead of

**unit** A **extend** B

**init** (…) **do**

**old**(…)

**end**

**end**

one may just use the form

**unit** A **extend** B

**use** B.**init**()

**end**

Более полное описание: после use можно указать все процедуры инициализации родительских юнитов, которые могут быть использованы для инициализации текущего с учётом их сигнатур. Например

**unit** A **extend** B, C

**use** B.**init**(T1, T2), B.**init**(T3), C.**init**(T4)

**end**

При этом конечно же не должно быть идентичных сигнатур.

Да, конечно, я должен писать все мысли, что есть в моей голове – пустые круглые скобки можно пропускать ... И именно наличие ключевого слова init говорит о том что это наследование инициализатора, а не выбор версии при конфликте версии при множественном наследовании ... (Или опять надо описывать что такое конфликт версий?)

**unit** A **extend** B

**use** B.**init**

**end**

Ниже полный перечень вопросов , включая те, которые уже были закрыты

Несколько  вопросов тебе на эту неделю, по которым я хотел бы получить твое ДА или НЕТ или мысли

1. **unit** A[G->Type] to be replaced with **unit** A[G **extend** Type]

Согласен.

2. **unit** A [ rtn: **routine** (T1, T2) ] - routine types as formal generic parameters

С идеей согласен. Синтаксис нужно оттачивать.

3. **while** <condition> **do** ... **end** and **do** ... **while** <condition> end instead of loop-based, to make the concept of super block consistent across the whole language. Minus 1 keyword plus **do** … **end** block everywhere. Example

**while** <condition>

**require** /\* This is in fact loop invariant, which is checked before each iteration starts\*/

<predicate>

**do**

<statements>

**when** <exception> **do**

**ensure** /\* This is the predicate which describes validity of the context after loop id fully done. For example loop was sorting an array and fater the end fo the loop array must be sorted (for any i in array.lower … array.upper implies array(i) <= array(i+1))\*/

<predicate>

**end**

Хрен с тобой, пусть будет так. Только не пойму, чем тебе слово loop не угодило…

4. **when** <exceptionType> => to be replaced with **when** <exceptionType> **do**

Согласен.

5. then - не является ключевым словом

Ладно.

6. invariant - не является ключевым словом - используем **require** вместо этого.

Поясни на паре примеров.

There is an example above about loop – we use keyword require instead. Just reducing the number of keywords. And here is the unit

**unit** A

**require**

<predicate>

**end** // A

So, terminology to be used in documentation – routines have preconditions and postconditions, units and loops have invariants. To highlight these assertions in the program code we use keywords require and ensure as they have semantics which is adequate. Require states which predicates must be true before we do something while ensure states which predicates must be true after we did something (routine, block, loop – in fact all of them are just different forms of blocks ☺)

Для управления включением-выключением инвариантов юнитов и циклов поддерживаем раздельный контроль для предусловий рутин и инвариантов и все. Т.е. языки управления проектами и управления рантайм конфигурациями используют понятия предусловие, постусловие и инвариант.

Не поял: так есть инварианты или нет? Примеры?

7. условный оператор использует усеченный блок для then & else ветвей без require секции в начале if <condition> do <statements> [else <statements>] end

Тоже не понял. Надо соотнести это с тем, что написано в описании языка.

**if** <condition>

**do** <statements>

[**else** <statements>]

**end**

In other words – there is no ‘then’ as a keyword

8. hidden - нет такого ключевого слова - используем [“{” [this| UnitTypeName {“,” UnitTypeName} ] “}”]  для управление видимостью члена юнита.

Поясни на примерах

**unit** A

foo /\* read-only(for attribute)/call-only(for routine) visibility for clients,

read-write(for attribute)/call-only(for routine) visibility for descendants \*/

{} goo /\* hidden from all clients,

read-write(for attribute)/call-only(for routine) visibility for descendants \*/

{A, B, C} too /\* hidden from all clients except A, B, C (exported to A, B, C),

read-write(for attribute)/call-only(for routine) visibility for descendants \*/

// NOTE: I do not insistent on this – it is just so natural in terms of syntax

{**this**} boo /\* hidden from all clients,

hidden from all descendants \*/

/\* Current unit accessible thru **this** has all the access to all its members except inherited as {this} form its parents\*/

**end** // A

Первое. Мне эти идиотские фигурные скобки активно не нравятся. У Бертрана не слишком хороший вкус в этом отношении.

Второе (главное). Задавать, кому этот класс может что-то предоставлять, а кому-то нет - это кажется методологически порочным. Когда ты проектируешь библиотечный класс, ты вообще не можешь предполагать кто этот класс будет юзать. Он должен быть спроектирован без знания о своих клиентах и потомках.

Да кстати - если есть анонимная рутина, то все атрибуты , объявленные в ней не являются глобальными переменными - они не доступны в других рутинах - так как они локальные переменные-константы анонимной рутины.

А это разве не очевидно? Все, что объявлено в теле рутины – локально…

9. Это не поправка в просто просьба - я послал свои предложение по тексту книги и хотелось бы ... ну какой-то реакции ...

10. По мелочи - может все-таки разрешить что операции (инфиксные, префиксные) начинаются с набора символов и все , а не жестко фиксируются

Ты как-то более систематично формулируй, что ли… а то я ничо не понимаю и чувствую себя тупым-тупым… Вообще, я против подобного рода лексических ограничений. Набор предопределенных операций расширять нельзя – это мое мнение вслед за Страуструпом. И тем более вводить какие-то правила, как называть новые операции…

11. Целочисленные константы - а может разрешить их записывать в ДОБАВОК  к указанию системы счисления в начале с возможностью указания системы счисления в конце вида 010101b  FFFx 777о. О каким сложным станет сканер 😊 Пусть люди пишут как им удобно ...

Сложность сканера ни при чем (и так просто, и эдак несложно). Но мне очень не нравится, когда в желании осчастливить всех, угодить привычкам разных групп программеров мы «размываем» язык. Я за нотацию 0x, 0b и т.д.

12. Ключевое слово routine используется для задания рутинного типа, а для создания рутинного объекта ты предлагаешь использовать имя рутины без круглых скобок, но как тогда различать рутины если они различаются только сигнатурами, Как взять рутинный объект?

foo (T1, T2) **do** ... **end**

foo (T1, T2, T3) **do** ... **end**

rtn **is** foo // неоднозначность - ошибка компиляции или как?

Это не просто неоднозначность – по моему мнению, это семантически неправильно **(Ты же сам хотел – со скобками вызов, без скокбок рутинный объект).**

Я так никогда не хотел, ты меня с кем-то путаешь. Если угодно, наличие круглых скобок – это характеристический признак рутин вообще. Объявление со скобками, и вызов со скобками. То есть, если рутина foo в данном контексте единственная, тогда, скрепя сердце, можно записать ее имя без скобок. Но если их несколько, то чтобы их различить, приходится писать ее имя с типами параметров.

И вообще: у нас после is всегда идет инициализация, то есть выражение. foo(T1,T2) – все что угодно, но только не выражение.

Объявляя объект rtn, ты хочешь, очевидно, задать его нач. значение? (**НЕТ! НЕ ОБЯЗАТЕЛЬНО. Например a: ?routine foo(T1, T2) – т.е. я просто описал a, как рутинную сущность с типом как у рутины foo (T1, T2))**

**Что такое вопросительный знак впереди?**

– ну так пиши rtn is foo(T1,T2). Какие проблемы? **(ЭТО ВЫЗОВ foo)**

**Какой же это вызов foo, если ты ТИПЫ задал в позиции параметров????**

(А, кстати, для задания типа можно было БЫ написать rtn: (T1, T2), ЕСЛИ БЫ эта запись не была синтаксически тождественной кортежу…)

Но так или иначе, объявление с инициализацией, как я написал, - вполне ОК.

rtn ( <сколько параметров передавать и что будет вызвано????> )

Здесь не проблема: ты хочешь ВЫЗВАТЬ **(НЕ ХОЧУ Я ВЫЗЫВАТЬ, по сути я беру указательн на функцию в терминах С, а в термниах СЛанга делаю рутинный объект ассоциированный с конкретной рутиной – rtn1 is new routine foo (T1, T2) – вот полная форма)** определенную foo, которой была инициализирована rtn – ну так подбрасывай ей аргументы подходящих типов и в подходящем количестве…

у меня пока это однозначно работает через ключевое слово ...

rtn1 is routine foo (T1, T2) rtn1 is foo(T1,T2)

rtn2 is routine foo (T1, T2, T3) rtn2 is foo(T1,T2,T3)

Нет проблем с однозначностью.

**Да все однозначно – это вызовы рутин foo а не инициализация атрибутов ссылками на рутины**

**rtn1 is foo(T1,T2) – это вызов foo c двумя параметрами**

**rtn2 is foo(T1,T2,T3) – и это в другой foo у которй 3 параметра**

ТАК… ПОХОЖЕ, У НАС НЕ ПОЛУЧАЕТСЯ НОРМАЛЬНОГО ДИАЛОГА. СИСТЕМА СО СПОНТАННЫМИ НАБРОСКАМИ, ИЗОБИЛУЮЩИМИ СОКРАЩЕНИЯМИ, ПОНЯТНЫМИ ТОЛЬКО ИХ АВТОРУ – И ПОПЫТКИ ОТВЕЧАЮЩЕГО РАЗГАДАТЬ НАМЕРЕНИЯ ОРИГИНАЛЬНОГО АВТОРА – **НЕ РАБОТАЕТ**.

ЕСЛИ ТЫ ХОЧЕШЬ НОРОМАЛЬНОГО ПРОЦЕССА ПРОЕКТИРОВАНИЯ, ИЗЛОЖИ СВОИ ПРЕДЛОЖЕНИЯ ПО ТЕМ ИЛИ ИНЫМ РАЗДЕЛАМ СИСТЕМАТИЧЕСКИ, В ВИДЕ **СВЯЗНОГО ТЕКСТА** С ПРИМЕРАМИ (НАПИСАННЫМИ МОНОШИРИННЫМ ШРИФТОМ).

Например, оказывается, у тебя запись foo(T1,T2) – это вызов (????), а Т1 и Т2 – это, оказывается обозначения аргументов???

В общем, изложи свои взгляды на рутины, рутинные типы, рутинные объекты, вызовы рутин – **в систематическом виде**. Прочитаю с великой радостью.

Т.е. получается что слово routine используется как для указание на рутинный тип, так и на создание-получение объекта рутинного типа ...

Не все твои нюансы я схватил, но мнение мое – **допиши какое твое мнение! Еще раз сформулирую, что у меня не получается рутинные типы без неоднозначностей ввести без дополнительного ключевого слова .... С дополнительным ключевым словом неоднозначностей нет ...**

13. Что до сих не решено - так это подходящее слово вместо super, так как смысл - это обращение к версии данной рутины из непосредственного родителя юнита.

Не обязательно рутины – любого члена родительского юнита.  
Вообще, слово super мне не очень понятно в условиях множественного наследования. Какой именно super имеется в виду?

**Еще раз, внутри рутины, которая является переопределением, можно вызвать ее ближайщую предшествующую версию, если таких версий много то надо указать из какого ближайщего рожителя эта версия приходитю Пример**

**unit** Z

foo (…) **do** … **end**

**abstract** goo ()

**end**

**unit** A **extend** Z

**end**

**unit** B

foo (…) **do** … **end**

goo () **do** … **end**

**end**

**unit** C **extend** A, B

**override** foo (…) **do**

// super (…) – such call is ambiguous in this context

A.**super** (…) А ПОЧЕМУ НЕ НАПИСАТЬ ПРОСТО **A.foo(…)**

B.**super** (…)

/\* Z.super (…) – is incorrect as C has only A and B as parents and it knows nothing about Z \*/

**end**

**override** goo () **do**

**super** (…) /\* this call is Ok as we have only one non-abstract version of goo\*/

**end**

**end**

Меня, конечно, подмывает просто заменить super на precursor и закрыть тему, уж очень давно идет процесс поиска короткого, но понятного ключевого слова ...

Precursor – отвратительное слово; не понимаю, как у Майера хватило вкуса вставить в язык эту фармацевтику.

Так как есть люди, для которых семантика слова super связана с понятием класс, умница Бертран и выбрал 100% правильный английский термин – предшественник – вызови мне предшественника, а в аптеке провизор ... ☺ Именно поэтому я и хочу избавиться от super, так как само слово означает класс, а не член класса в других языках ... Но, я не смог найти слова короче, чем precursor со значением предшественник, ancestor – опять же используется для классов, а не для рутин ...

14. Так как ты любишь вложенности, у нас есть такая вещь StatementsList: { Statement[“;”]|UnitRoutineDeclaration} - т.е. среди операторов рутины могут быть описания вложенных рутин и далее рекурсивно. Но пока в таком виде это не дает возможность внутри отдельно-стоящей рутины иметь вложенные рутины также параметризуемые - надо?

[G] standAloneRtn (arg: G) do

         foo () do // Так можно

goo [G] (arg)

goo [Integer] (444)

         end

         [G] goo (arg: G) do ... end // А вот так пока нельзя ... Надо?

end

Вообще, в С++ такое возможно, только пример неудачен: для вложенной рутины, которая тоже параметризуется, надо, очевидно, выбрать для параметра другоой идентификатор – иначе просто синт.ошибка будет.

Как раз никакой ошибки не будет! Вложенная рутина может вполне настраиваться на тот же самый параметр, что и охватывающая, а может и на другой. В примере выше добавил аргументы, чтоыб показать всю полноту картины ... Т.е. внутри настраиваемой рутины standaloneRtn есть 2 вложенные рутины foo и goo и в теле foo стоят расзные настройки-вызовы goo…

Для полноты и стройности надо бы такое допустить, но реализовывать (да и использовать) будет довольно сложно.

Т.е. я это интерпретирую как ответ Да ☺

Если внутри рутины-шаблона есть влож.рутина, то она по умолчанию тоже шаблонная ТЕМ ЖЕ параметром, что что и объемлющая. Если же ты хочешь рассматривать влож.рутину как параметризуемую ЕЩЕ И ДРУГИМ параметром – просто напиши имя этого параметра.

Именно поэтому повторять одно и то же имя для объемлющей и для вложенной - бессмысленно

15. Еще одна тонкость о которой не думал раньше

 a is T

в этом случае вид а будет определятся стоит или нет ref или val у юнита Т, а если мне надо сменить вид объекта при его создании, получается что надо поддержать синтаксис

a is ref T

b is val T

Диаграммы то я подправил - это несложно. Но момент важный ...

Да это же у нас было с самого начала!! Вот в точности как ты написал!

Ну просто пакет поправок в конституцию СЛанга 😊 Голосуй, а то 😊

16. Из него (#15) сразу следует, что константные объекты тоже должны поддерживать синтаксис вида

**unit** X

**const is**

             a, // вид объекта а определяется видом объектов по умолчанию юнита Х

             b is ref, // в явном виде указан вид объекта а имя юнита известно

Неправильно, кмк: после ‘is’ мы пишем инициализатор, а здесь у тебя слово, относящееся к типу. Почему не b: ref?

Еще проще !

**ref** b,

             с is val, // симметрично

**val** c,

**ref** d ~~is ref~~.**init** (T1, T2), /\* Т.е. если надо указать какую процедуру инициализации вызывать то просто пишем ее после ref/val \*/

А почему не просто d: ref is X(arg1,arg2) ?

**val** e ~~is va~~l.**init** (T1, T2)

**end**

**init do** ... **end**

**init** (T1, T2) **do** ... **end**

**end** // Х