Я продолжаю писать текст статьи и, написав некоторое описание, вдруг с ужасом понял, а зачем нам init. Каков в нем смысл - мы же разрешаем написать

a **is** A

что эквивалентно

a **is new** A.init ()

Так зачем нам нужен init???? Получается только для того, чтобы отметить процедуры инициализации в теле юнита …

Я решил еще раз струкутрно подойти к дизайну Итак, есть 3 варианта

1. Init как ключевое слово, которое обозначает процедуры инициализации
2. Процедура инициализации - это просто метка некотрой процедуры, т.е. любая процедура может быть помечена как init. Аналогично тому, как любая отдельно-стоящая процедура без параметров или с параметром массив строк может быть выбрана как точка входа - вот такая аналогия.
3. Имя юнита используется как имя процедуры инициализации
4. Зуев может предложить другие варианты? :-)

Примеры, чтобы было видно

1. **Текущий вариант**

**unit** A

attr: A

// Рутины инициализации

**init do**

**init** (**this**)

**end**

**init** (a: A) **do**

attr :=a

**end**

// Обычные рутины

foo **do**

**new** attr // создание атрибута юнита 1й вариант (короткий)

**new** attr.**init** () // создание атрибута юнита 1й вариант (полный)

**new** attr(**this**) // создание атрибута юнита 2й вариант (короткий)

**new** attr.**init** (**this**) // создание атрибута юнита 2й вариант (полный)

**end**

**end**

a **is** A // короткая запись. Выглядит круто, но понять … ох …

a **is new** A // промежуточная(средняя)запись, понятнее, но умолчания …

a **is new** A.**init**() // полная запись, все явно

a **is** A(A) // короткая запись – выглядит круто, но понять … ох …

a **is new** A(**new** A) // промежуточная(средняя)запись

a **is new** A.**init** (**new** A.**init**()) // полная запись

Минусы:

- слишком много вариантов написания одного и того же, поэтому трудно понимать. Умолчания – new и init можно пропускать …

Плюсы:

- Оригинально ☺

2. **Схема с пометкой процедур как процедур инициализации**

**unit** A

attr: A

// Рутины инициализации

**new** create **do**

init(**this**)

**end**

**new** init(a: A) **do**

attr :=a

**end**

// Обычные рутины

init: A **do**

**new** attr.create // создание атрибута юнита 1й вариант

**new** attr.init (**this**) // создание атрибута юнита 2й вариант

**end**

init(a: A): A **do return this end**

**end**

a **is new** A.create // Только полная запись

a **is new** A.init (**new** A.create) // Только полная запись

/\* Только полная запись так как A.init (…) без new – это будет вызов из модуля А функции init в данном примере …

Ну и еще надо отметить что pure и safe процедуры не могут быть процедурами инициализации

Важно – здесь init не есть ключевое слово!!! Специально на это обращаю внимание!

\*/

Т.е. вариантр x **is** **new** X рабоатет только если у юнита Х нет процедуры инициализации

Минусы:

- слишком длинно, много надо писать при создании объектов

- непривычно для С++/Java программистов

Плюсы:

- все очень наглядно и однозначно

- можно обойтись один ключевым словом **new**

3. **Имя процедур инициализации - это имя юнита**

**unit** A

attr: A

// Рутины инициализации

A **do**

A(**this**)

**end**

A(a: A) **do**

attr := a

**end**

// Обычные рутины

foo **do**

A // О ужас – это вызов процедуры инициализации без параметров

A () // Тоже самое

**new** attr

**new** attr ()

**new** attr(**this**) // создание атрибута юнита 2й вариант

**end**

**end**

a **is** A // короткая запись

a **is** A()// промежуточная(средняя)запись

a **is new** A() // полная запись

a **is** A(A) // короткая запись

a **is new** A(**new** A)// промежуточная(средняя)запись

a **is new** A(**new** A()) // полная запись

Минусы:

- провоцирует делать имена юнитов короткими – слишком много писать ☺

- непосредственный вызов процедры инициализации это использование имени юнита - путает

Плюсы:

- схоже с решением в других распространненых языках – всем известно

Еще раз напомню про связь между процедурами инициализации и проверками инвариантов юнитов – ибо это и есть основной смысл процедур инициализации. Процедураа инициализации должна привести объект в состояние, удовлетворяющее инварианту юнита. Отсюда простой вывод – до вызова процедуры инициализации объект может быть в неконсистентном состоянии. Т.е. есть два сценария - объект только что создан – выделена под него память – надо вызвать процедуру инициализации. Второй сценарий: объект уже существует и мы хотим его вернуть в начальное состояние или произошла исключительная ситуация – объект разрушен во время работы и мы хотим его восстановить – вызвать процедуру инициализации, обработав такую исключительную ситуацию. Т.е. до вызвова процедуры инициализации не надо проверять инвариант юнита, а вот после надо.

Object life cycle:

Runtime creates an object: memory for the object is allocated

Call to the particular initialization procedure

Check invariant of the object

Access any visible member of the object (call any member-fucntions or read data-member )

Invariant is valid

Invariant is violated

Exception Invariant violation is raised

Object is safe to be destroyed - destroy

Для статьи, я склонен к выбору варианта 2, хотя вариант 3 мне тоже понятен … Но если ты приведешь убедительные доводы в защиту другого варианта, то я готов с превеликим интересом тебя выслушать (скорей всего в письменом виде, чтобы интроверт был в своей тарелке). Итак – от меня покая нет и шайба на твоей половине поля …

Вполне возможно у тебя возникент вопрос про форму new процедурного вида new attr. Вот тебе пример

**unit** A

attr: **as this**

foo **do**

**new** attr /\* Тип создаваемого объекта определяется динамически \*/

end

**end**

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

**end**

**unit** C **extend** A

**end**

a **is new** A

b **is new** B

c **is new** C

a.foo // a.attr – это объект типа A

b.foo // b.attr – это объект типа B

c.foo // c.attr – это объект типа C