1.5 старнички про многоточковые имена. Пусть имеется 4 исходника , компилируемые раздельно … 1, 2, 3 прекрасно компилируются в объектный код или синатксический анализатор сохраняет для них только AST …

Source #1:

**unit** A  
b: B

**end**

**unit** B

c: C  
**end**

**unit** C

foo **do end**  
**end**

Source #2:

**unit** a.b.c

foo **do end**  
  
**end**

Source #3:

a.b.c(): **rtn** () **do**

**return rtn** () **do end**

**end**

// Это фукнция которая возвращает процедуру

Source $4:

a.b.c.foo

Тема по-моему очень интересная – как синтаксическому анализатору разбирать подчеркнутый фрагмент.

Подход 1.

Синанал отслеживает и запоминает все описания всех атрибутов (юниов и локальных атрибутов рутин, как отдельно стоящих, так и анонимных ) и тогда он по очереди в таблице таких имен ищет сначала a, потом a.b, потом a.b.c если ничего не было найдено то он вынужден воспринимать многоточковое имя как имя юнита – в данном случае используемого как модуль … В нашем случае – если было описание вида a **is** A то строим одно AST, а если не было, то другое … Именно это меня и смущает, что для одной и той же синтаксической конструкции может быть разное AST и следователньо семантика … Вот в чем как мне кажется избыточная сложность …

Подход 2.

Синанал бьет многоточковую конструкция на две части последнее имя и все что до него и откладыыает рещение с разбором передней части до семантического анализатора, котрый точно будет знать что такое a.b.c Точнее можно сделать два варианта – один что это строка – многоточковое имя, а вотрое AST для многоточкового вызова-взятия

Подход 3. Запрещается называть юниты с маленькой буквы, но тогда как быть со отдельно-стоящими функциями – ты их тоже хотел назыыать такими многоточковыми именами. Ну запретить их так назыыать например …

Подход 4. ЕА мождет предложить свое решение.

Для заключения – **мне очень нравится твоя идея с точками** – она дает еще один уровень гибкости, и если ты знаешь как правильно написать синтаксический анализатор (не весь фронтенд!), то отлично научи меня!

И еще один момент расширение юнитов из той же оперы

Source #1:

**unit** A  
a: A

**end**

Source #2:

**extend unit** A

foo **do end**  
  
**end**

Source #3:

a is A

a.foo

Что получается , что синатксический анлизатор однозначно моедт построить AST для 3его исходника, а вот семантический анализатор либо выдаст ошибку если вконтексте только исходник 3 и 1 и не выдаст ошибку если в контексте все 3 исходника. Так что момент что надо разделять во времени просто построение AST и семантика с кодогенерацией имеет под собой логику если мы говорми о повотрном испоьзовании компонент (Недоря плохо спит ☺)

Другими словами можно породить некий псевдокод для 3его исходника а когда делается сброка проверть семантику и проводить догенерацию кода . Другим словами лексические и синтаксические ошибки не зависят от окружения – контекста проекта …

Что виит человек, когад смотрит на 3й исхолдник – вот создается объект Типа А а потмо из этого объекта вызывается процеду с именем foo – вот такой код вполен модно генерить … А при сбороке проверять его валидность и оптимизировать …