v. 09/07/18

Land Test IDE

Язык LanD

Генерируемый по land-описанию парсер осуществляет LL(1) разбор.

Грамматика на языке LanD состоит из двух секций: секции правил и секции опций. В секции правил описываются терминальные и нетерминальные символы, в секции опций – различные опции, влияющие на разбор и превращение полученного дерева разбора в абстрактное синтаксическое дерево. Пример – грамматика для yacc-файла:

```
COMMENT      : COMMENT_L|COMMENT_ML
COMMENT_L    : '//' ~[\n\r]*
COMMENT_ML   : '/*' .*? '*/'
STRING       : STRING_STD|STRING_ESC
STRING_SKIP  : '\\"' | '\\\\'
STRING_STD   : '"' (STRING_SKIP|.)*? '"'
STRING_ESC   : '@'('"' ~["]* '"')+
LITERAL      : '\'' ('\\\''|'\\\\'|.)*? '\''
DECLARATION_CODE : '%{' (STRING|COMMENT|.)*? '%}'

ID : [_a-zA-Z][_0-9a-zA-Z]*
DECLARATION_NAME : '%'ID

grammar      =   declaration* '%%' rule* grammar_ending
grammar_ending   =   ('%%' Any)?
declaration      =   symbol_declaration | other_declaration
symbol_declaration = ('%token' | '%left' | '%nonassoc' | '%right' | '%type') ('<' ID '>')? (ID|LITERAL)+
other_declaration = DECLARATION_NAME Any
rule         =   ID ':' alternative ('|' alternative)* ';'
alternative      =   (ID | block | LITERAL | '%prec')*
block        =   '{' (Any|block)+ '}'

%%

%parsing ignoreundefined
%parsing start grammar
%parsing skip COMMENT STRING DECLARATION_CODE
%nodes ghost declaration
```

Описания символов и опций начинаются с начала строки.

Описание терминального символа начинается с идентификатора, после которого идёт двоеточие; регулярное выражение записывается в формате, применяемом в генераторе компиляторов ANTLR. Если некоторая последовательность входных

символов соответствует нескольким описанным терминалам, лексер возвращает парсеру тот терминал, который описан раньше (выше в грамматике).

Описание нетерминала начинается с идентификатора, за которым следует знак равенства. Правило разделяется на альтернативы при помощи символа |. Внутри правил разрешено использование не описанных отдельно терминальных символов в случае, если регулярное выражение для этих символов представляет собой обычную строку (в antlr строковые литералы записываются в одинарных кавычках). Пример - '%type', '{', '%%'. Также элементом альтернативы правила может являться набор элементов, заключённый в круглые скобки – группа. Группа также может быть разбита на несколько альтернатив (см. правило **alternative**). К любому элементу правила может быть применён квантификатор +, * или ? («один или более», «ноль или более», «ноль или один» соответственно). Также существуют конструкции специального вида *!, ?!, означающие, что в случае неоднозначности (в случае, когда очередной токен может быть трактован и как наличие конструкции, помеченной квантификатором, и как её отсутствие) нужно сделать выбор в пользу наличия. При помощи такой конструкции в грамматике PascalABC решается проблема висящего else:

```
if  =  'if' Any 'then' operator ('else' operator)?!
```

В результате else всегда относится к ближайшему if. Конец описания терминала и нетерминала не требуется обозначать дополнительным символом.

Символ Any - специальный терминальный символ, обозначающий место, где возможно сопоставление последовательности из 0 и более токенов, не описанной в явном виде в этом месте грамматики. Семантика Any подробно рассмотрена в статье «Tolerant parsing with a special kind of “Any” symbol». Говоря по-простому, при написании грамматики мы пишем Any вместо подробного описания тех мест, структура которых не важна для нас в рамках текущей задачи, или вместо части их описания. Если область реальной программы, соответствующая некоторому вхождению Any в грамматику, не содержит токенов, которые могут идти сразу после этого Any в строках, порождаемых данной грамматикой, Any гарантированно не захватит больший участок, чем требуется, и мы не потеряем релевантную информацию о структуре программы.

Пример правила для подбора перечислимого типа в C# файле, сформулированного с участием Any:

```
enum  =  'enum' name Any '{' Any '}' ';'?
```

Символ Any позволяет пропустить все токены до тех, которые могут следовать за ним в текущем месте в соответствии с грамматикой и ситуацией на стеке. В данном примере символы Any означают пропуск токенов до ‘{‘ и ‘}’ соответственно.

Следует отметить, что для правила

```
a  =  TOKEN1 TOKEN2 | Any TOKEN2
```

в ситуации, когда символ **a** находится на вершине стека, и текущий токен – это TOKEN1, разбор пойдёт по первой ветке, поскольку TOKEN1 может быть явным образом сопоставлен с её началом. Если же текущий токен – это TOKEN2 или TOKEN3, разбор пойдёт по второй ветке, так как в данной ситуации отсутствует возможность сопоставить эти токены с чем-либо явно указанным и допустим терминал Any. При этом для TOKEN2 символ Any будет соответствовать пустой последовательности токенов.

Использование символа Any предполагается в местах, структура которых неизвестна или неинтересна разработчику грамматики (в терминах Island Grammars данные места именуются водой в противоположность «островам» - местам, структура которых нам важна). Тем не менее, в случае, когда острова по своей структуре близки к воде, некоторое грубое описание структуры воды всё равно может потребоваться. Также требуется минимальное описание структуры воды, если известны её (воды) границы и допустимо самовложение. К примеру, в уасс-грамматике водой являются вставки кода на целевом языке программирования, блоки кода могут быть вложены один в другой, и эту вложенность необходимо учесть в грамматике. Паттерн «самовложение» описывается следующим образом:

```
block       =  '{' (Any|block)+ '}'
```

В случае необходимости можно наложить явные ограничения на содержимое области, соответствующей Any: для этих целей существуют конструкции AnyExcept(…), AnyAvoid(…), AnyInclude(…), также возможна запись в общей форме: Any(Except(…), Avoid(…), Include(…)). При записи общей формы порядок конструкций неважен. На месте многоточий может быть записан через запятую произвольный набор строковых литералов и имён терминалов. Аргументами для Except являются токены, при встрече которых необходимо завершить сопоставление Any, в случае использования Except автовыведение правой границы Any на основе состояния стека не происходит. Аргументы для Avoid – токены, встреча которых в процессе сопоставления Any означает ошибочную ситуацию и требует перехода к восстановлению. Аргументы для Include – токены, которые нужно трактовать как часть Any даже если они входят в автоматически вычисленную правую границу. Все три конструкции имеют равный приоритет, их совместное использование в одном Any(…) возможно и осмысленно только если множества аргументов для них не пересекаются.

Реализованное в LanD восстановление отчасти напоминает классический режим паники: благодаря тому, что в процессе разбора автоматически строится дерево,

существует возможность подняться от текущего разбираемого нетерминала к ближайшему его родителю A, для которого есть продукция A -> Any, далее с места возникновения ошибки осуществляется стандартное сопоставление Any, при этом началом области, соответствующей этому Any считается начало области, которую до ошибки начали разбирать как выводимую из нетерминала A. Данный подход хорошо работает в случае, когда на одном уровне могут быть острова и водные конструкции, начинающиеся одинаково, но с некоторого места разные. К примеру, правило ниже позволяет разобрать содержимое класса и выделить только методы, пропустив свойства и поля. Парсер пытается распознать как метод поля и свойства, так как они тоже начинаются с модификатора, типа и имени, потом происходит ошибка и откат до уровня нетерминала class_content, после чего эти конструкции распознаются как Any.

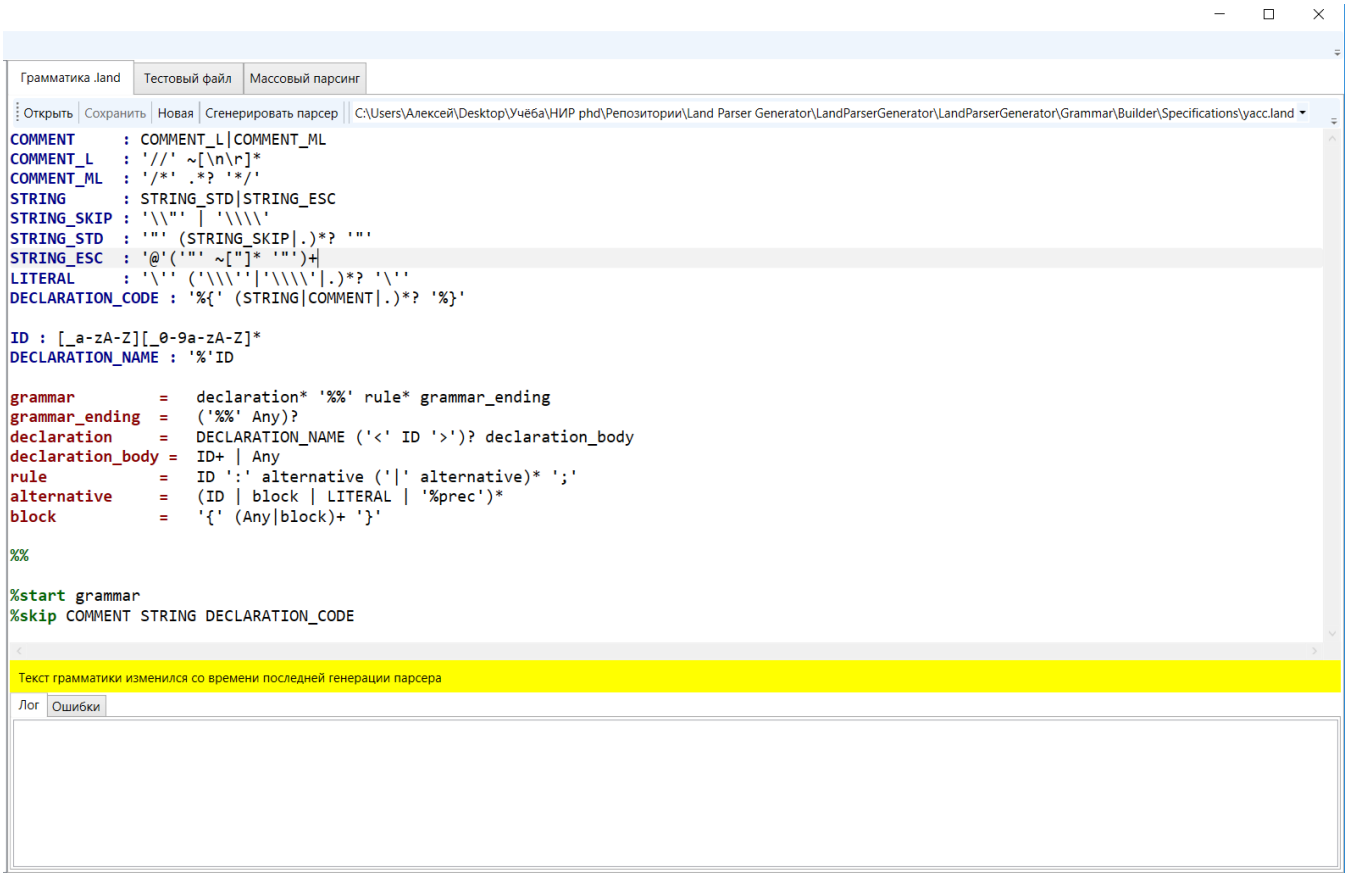
```
class_content = MODIFIER* type name arguments Any (block | ';')
              | Any
```

Опции разделены на несколько групп. К группе **parsing** относятся опции, влияющие на процесс разбора. Опция **start** служит для задания стартового нетерминала, опция **skip** – для задания терминалов, которые должны распознаваться лексером, но пропускаться синтаксическим анализатором. К таким терминалам относятся, например, комментарии и строки: внутри них может встретиться что угодно, поэтому важно подобрать их как монолитные лексемы. Терминалы, помеченные опцией **skip**, могут фигурировать в правилах грамматики; в процессе разбора в случае, когда такой терминал оказывается допустимым/ожидаемым в соответствии с ситуацией на стеке, он не отбрасывается, а обрабатывается стандартным образом. Опция **ignoreundefined** сообщает лексическому анализатору, что символы, которые не удалось склеить в какую-либо лексему, должны быть отброшены. В случае, если опция не указана, для каждого из таких символов лексер возвращает парсеру специальный предопределённый токен UNDEFINED. После опции **fragment** перечисляются имена вспомогательных токенов, описанных в грамматике только для того, чтобы впоследствии их можно было использовать в описаниях более сложных токенов. Передача парсеру этих токенов, распознанных в чистом виде, не осуществляется.

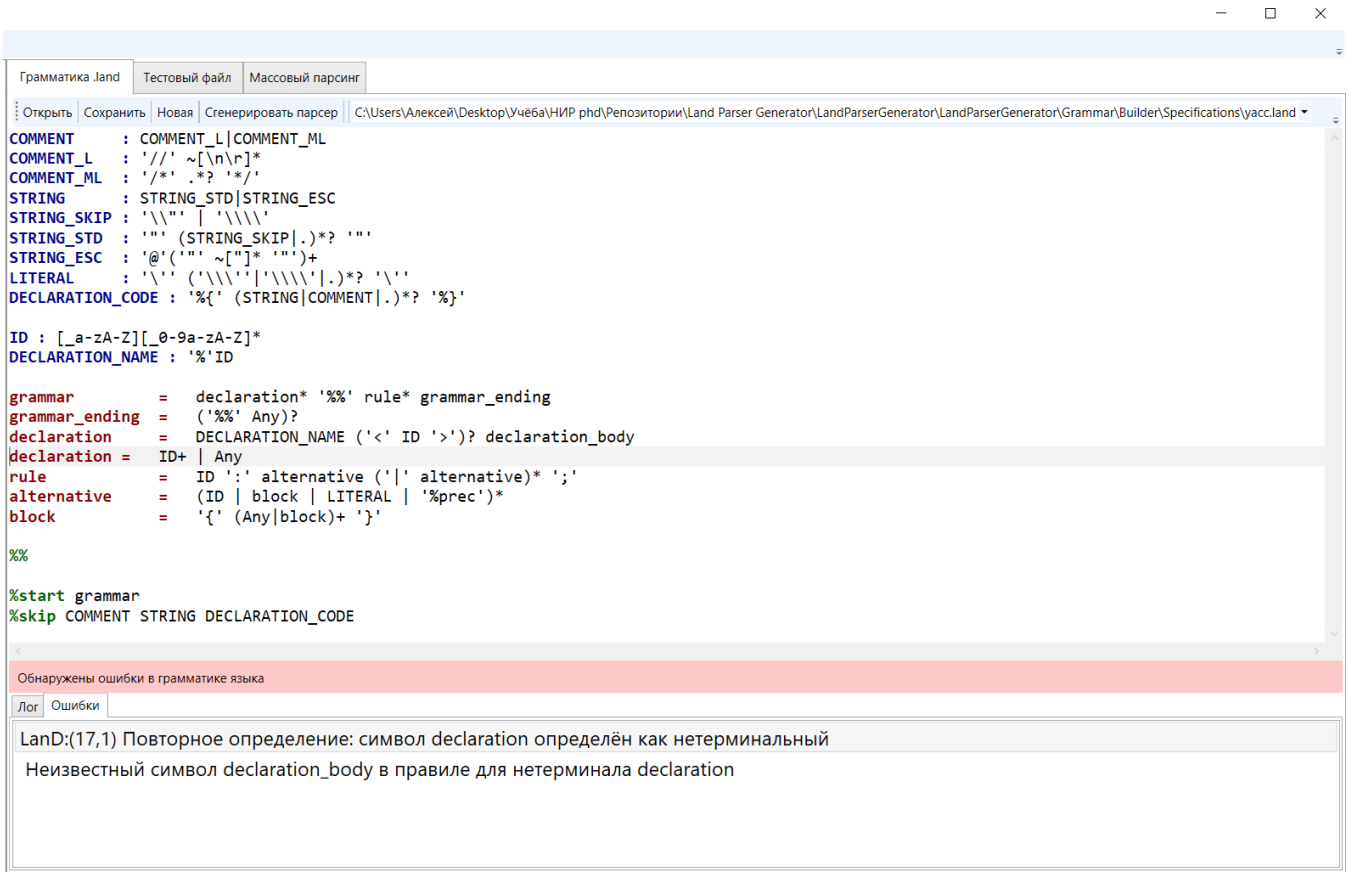
Опции группы **nodes** позволяют повлиять на построение AST. При помощи опции **ghost** можно задать перечень нетерминальных символов, узлы для которых не должны присутствовать в финальном дереве разбора. Эти узлы уничтожаются при преобразовании AST в дерево разбора, потомки такого узла становятся потомками его родителя. Опция **leaf** означает, что узел, соответствующий помеченному этой опцией нетерминалу, должен стать листовым. При этом все токены области программы, выведенной из этого нетерминала, помещаются в массив Value этого узла.

IDE

В IDE есть три основные вкладки.

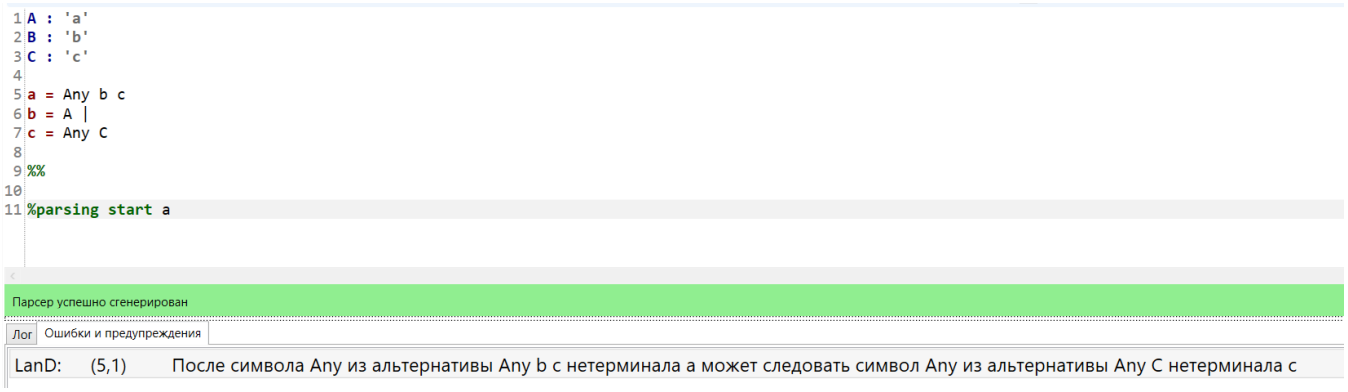


Вкладка **«Грамматика .land»** нужна для открытия имеющейся грамматики, создания новой, внесения изменений в грамматику и её сохранения. При помощи выпадающего списка можно быстро открыть одну из последних открытых land-грамматик. Кнопка «Сгенерировать парсер» запускает парсинг land-грамматики и генерацию по ней парсера для описанного формата. В случае успешной генерации строка статуса внизу окрашивается в зелёный, в случае неуспешного – в красный, на вкладке «Ошибки» внизу отображается перечень найденных в land-описании ошибок. Существует три источника ошибок: GPPG, Antlr, LanD. Источник ошибки отображается в начале сообщения. Ошибки LanD связаны с содержательной частью грамматики (наличие прямой или косвенной левой рекурсии, повторные описания символов, использование не описанных символов и т.п.), ошибки GPPG означают несоответствие описания land-формату с точки зрения синтаксиса. Ошибки Antlr возникают, если регулярные выражения были описаны в формате, не поддерживаемом Antlr.



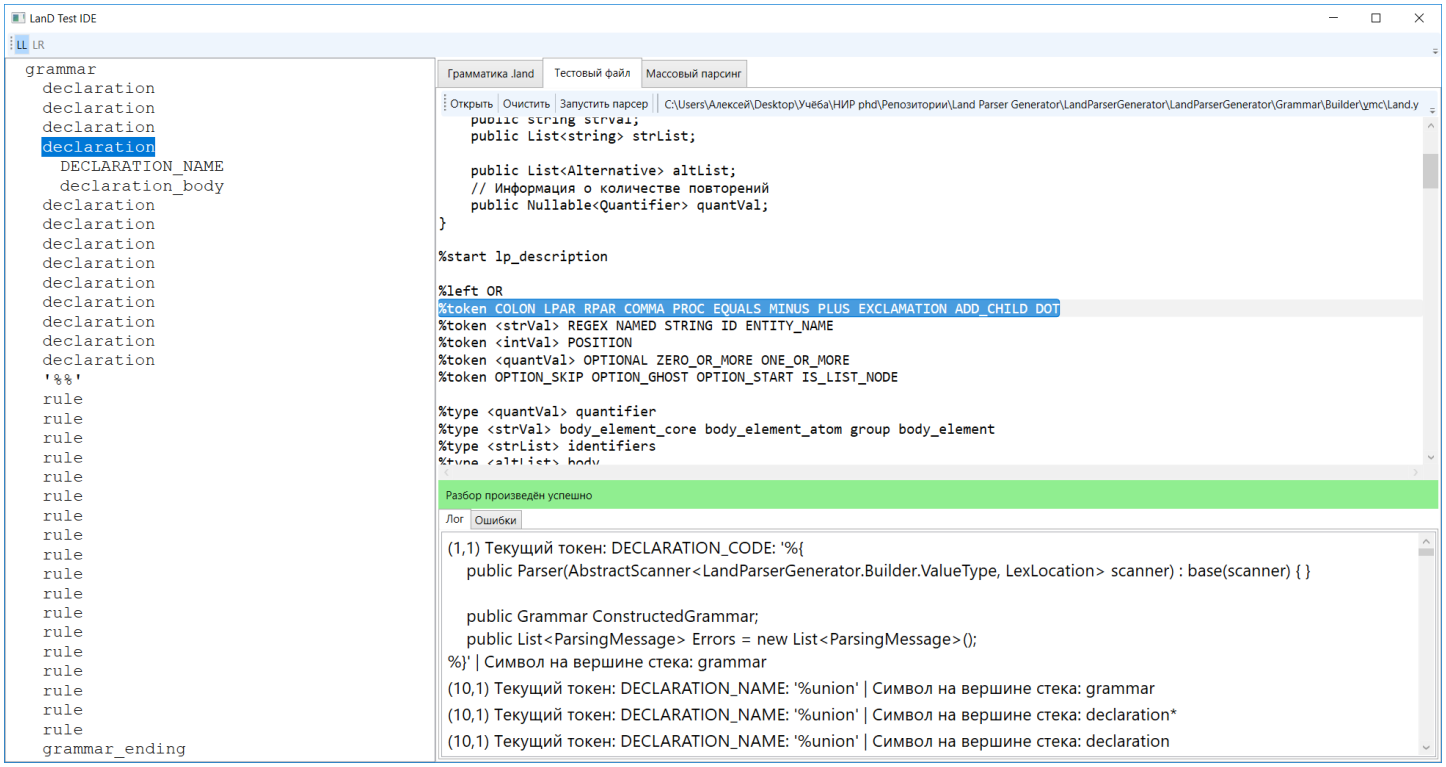
По двойному клику на те ошибки, для которых в скобках указаны номер строки и номер столбца, курсор в редакторе устанавливается в позицию ошибки. Ошибки, касающиеся формата регулярных выражений, выдаются не самим парсером .land-формата, а генератором ANTLR, служащим для создания лексического анализатора.

Среди сообщений LanD следует выделить предупреждение следующего вида:

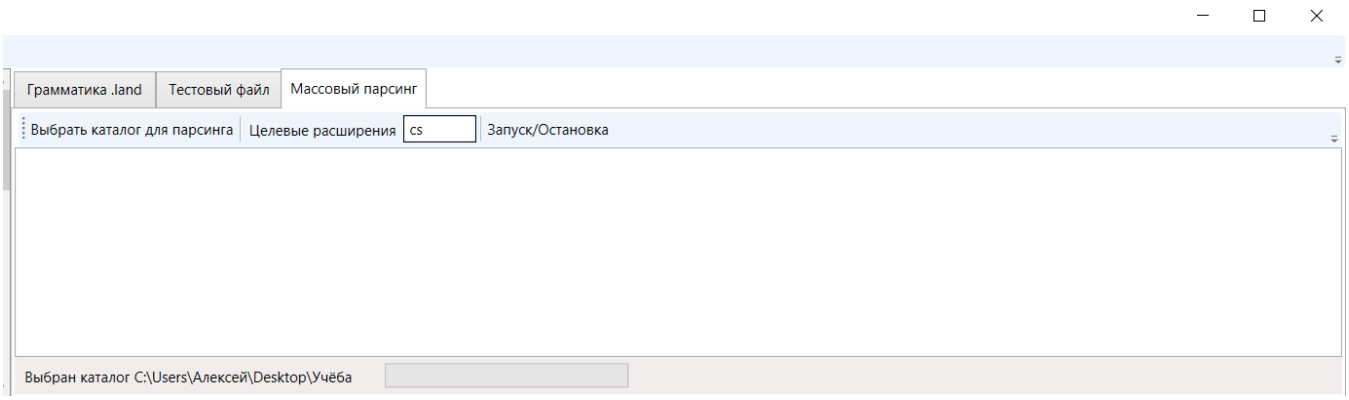


Наличие предупреждений не интерпретируется, как неуспешная генерация парсера, тем не менее, данное сообщение означает, что в процессе разбора может возникнуть ситуация с последовательно идущими Any, описанная в статье. Если при сопоставлении Any парсер обнаруживает, что после Any в соответствии с ситуацией на стеке может идти ещё одно или более Any, он рассматривает в качестве правой границы текущего Any не только те токены, которые могут идти непосредственно после него, но и те, которые могут идти после следующего за

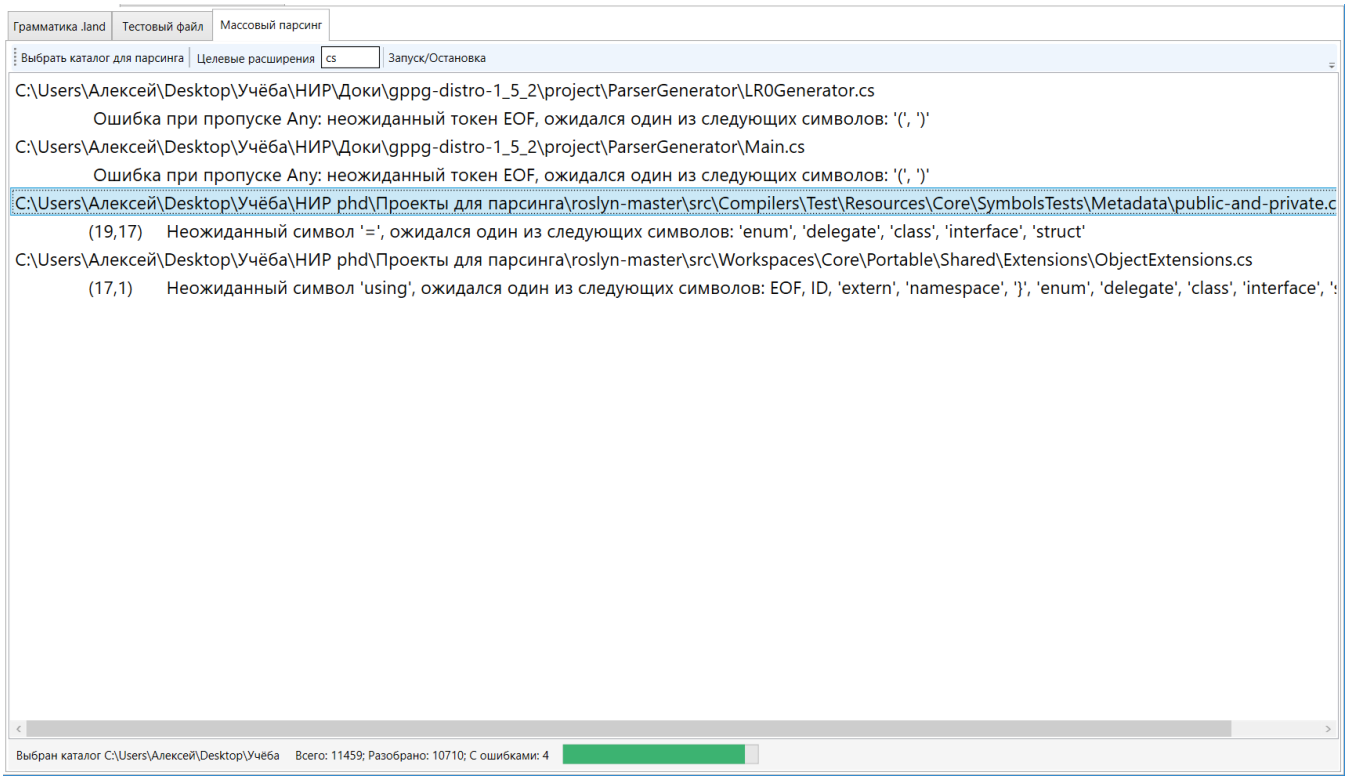
ним Any, следующего Any за следующим и так далее. Идейно таким образом мы хотим захватить при помощи первого Any токены, которые могут соответствовать всей этой цепочке. При этом, если цепочка порождена разными вхождениями Any в грамматику, возможны ситуации, когда это будет сделано не совсем корректно и пропуск Any окончится раньше, чем необходимо. Если данный эффект наблюдается, рекомендуется вместо символа Any, за которым может следовать другой Any, записать AnyExcept и явно указать границу для соответствующей ему области.



На вкладке **«Тестовый файл»** можно применить сгенерированный парсер к некоторому тексту, который этим парсером должен парситься. Текст можно как набрать в редакторе, так и прочесть из заданного файла и затем отредактировать в случае необходимости. К примеру, можно открыть cs-файл, если парсер сгенерирован по грамматике sharp.land. Кнопка «Запустить парсер» запускает парсер, в нижней вкладке «Лог» выводится лог разбора, во вкладке «Ошибки» выводятся ошибки (в эту вкладку имеет смысл заходить, если строка статуса по итогам разбора красная). В панели слева от вкладок отображается AST для разобранного файла, одинарный клик по элементу-узлу выделяет в тексте тестового файла участок, соответствующий этому узлу. Двойной клик раскрывает список дочерних узлов.



Вкладка **«Массовый парсинг»** позволяет совершить масштабную проверку корректности работы сгенерированного парсера. На данной вкладке можно выбрать каталог, в котором будут рекурсивно искаться файлы с расширением, указанным в поле «Целевые расширения» (можно указать несколько через запятую), по нажатию на «Запуск/Остановить» к каждому из этих файлов будет применён сгенерированный парсер. Поскольку процесс длительный, в статусной строке отображается полоса прогресса и счётчик файлов. Повторное нажатие приведёт к досрочному прекращению процесса парсинга. По мере разбора в основной области вкладки появляется информация о файлах, парсинг которых завершился с ошибкой. Двойной клик по имени файла позволяет открыть его во вкладке «Тестовый файл» и перейти к этой вкладке.



По окончании разбора в основной области также появляется список самых долго разбираемых файлов, отсортированный по уменьшению количества затраченного времени.

Грамматики

В папке «LanD Specifications» находятся тестовые LanD-грамматики для разных языков. В папке «sharp» находятся следующие файлы:

- sharp_initial, sharp_syrcose – грамматики C#, разработанные для выделения перечислений, классов и всех членов класса (в первом файле первая рабочая версия грамматики, во втором – немного подправленная для конференции), успешно использованы для разбора исходников PascalABC.NET и Roslyn;
- sharp_more_structured1, sharp_more_structured2 – вариации на тему более читабельной грамматики C# и более строгого структурирования имени типа и имени сущности, успешно использованы для разбора исходников PascalABC.NET и Roslyn;
- sharp_classes_methods – грамматика, трактующая как острова только классы и методы (пока не работает);
- sharp_if_switch – выделение сущностей на более низком уровне – островами являются операторы if и switch, - успешно использована для разбора исходников PascalABC.NET и Roslyn.

В папке «уасс» находится рабочая легковесная грамматика уасс-формата, в папке «pascal» - грамматики, применяемые для разбора ограниченного подмножества программ на PascalABC.NET (создавались для частной задачи разбора программ, написанных на занятиях в ВКШ).