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 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)+ '}'

%%

%start grammar
%skip COMMENT STRING DECLARATION_CODE
%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.

Описание нетерминала начинается с идентификатора, за которым следует знак равенства. 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). Конец описания терминала и нетерминала не требуется обозначать дополнительным символом.

Символ Any - специальный терминальный символ, обозначающий место, где возможен пропуск последовательности токенов (возможно, пустой), не являющихся допустимыми в текущем месте в соответствии с грамматикой. 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. 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 текущей 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 yacc-грамматике водой являются вставки кода на целевом языке программирования, блоки кода могут быть вложены один в другой, и эту

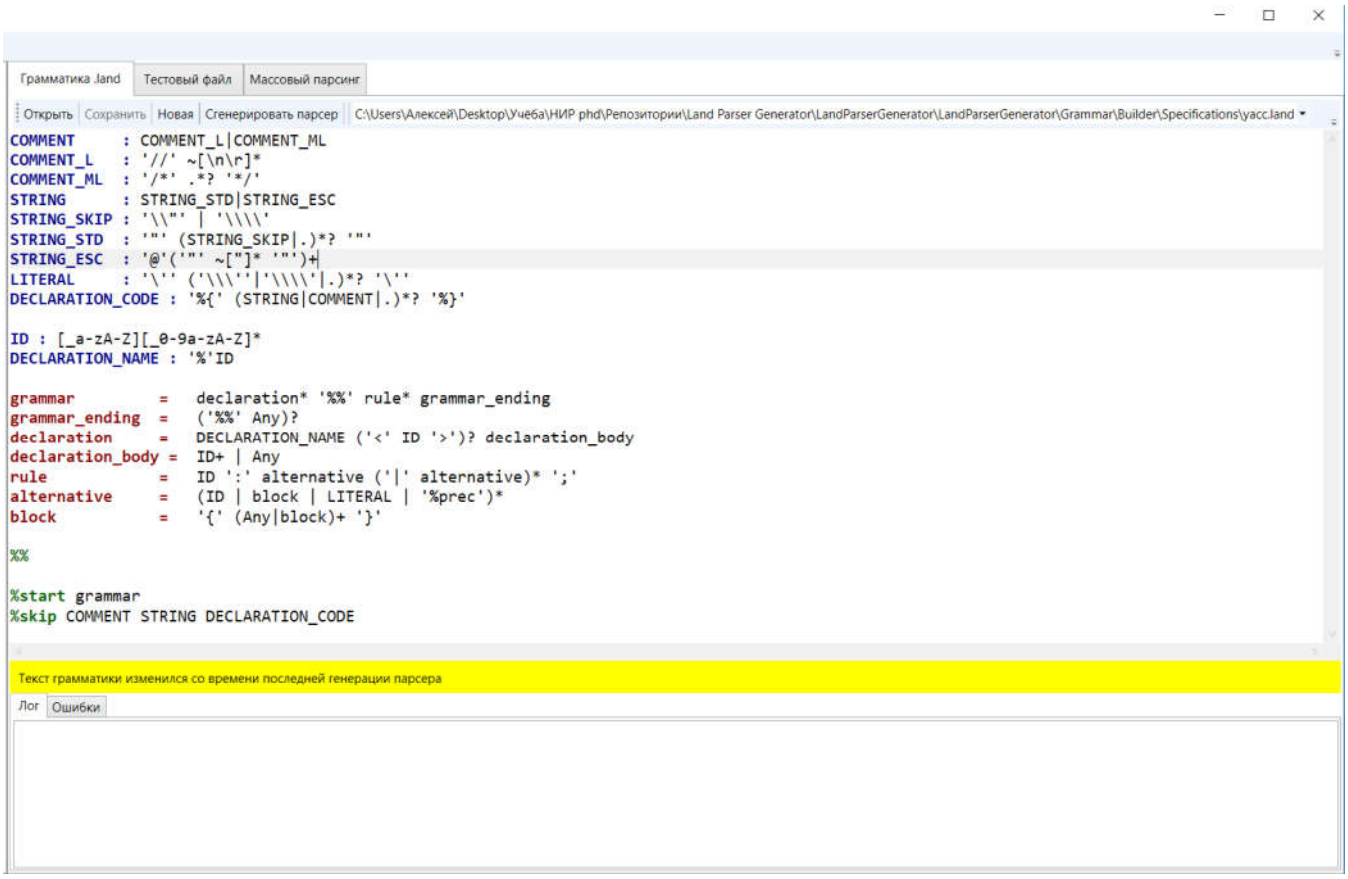
вложенность необходимо учесть в грамматике. Паттерн «самовложение» описывается следующим образом:

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

Опция %start служит для задания стартового нетерминала, опция %skip – для задания терминалов, которые должны распознаваться лексером, но не должны возвращаться синтаксическому анализатору. 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.

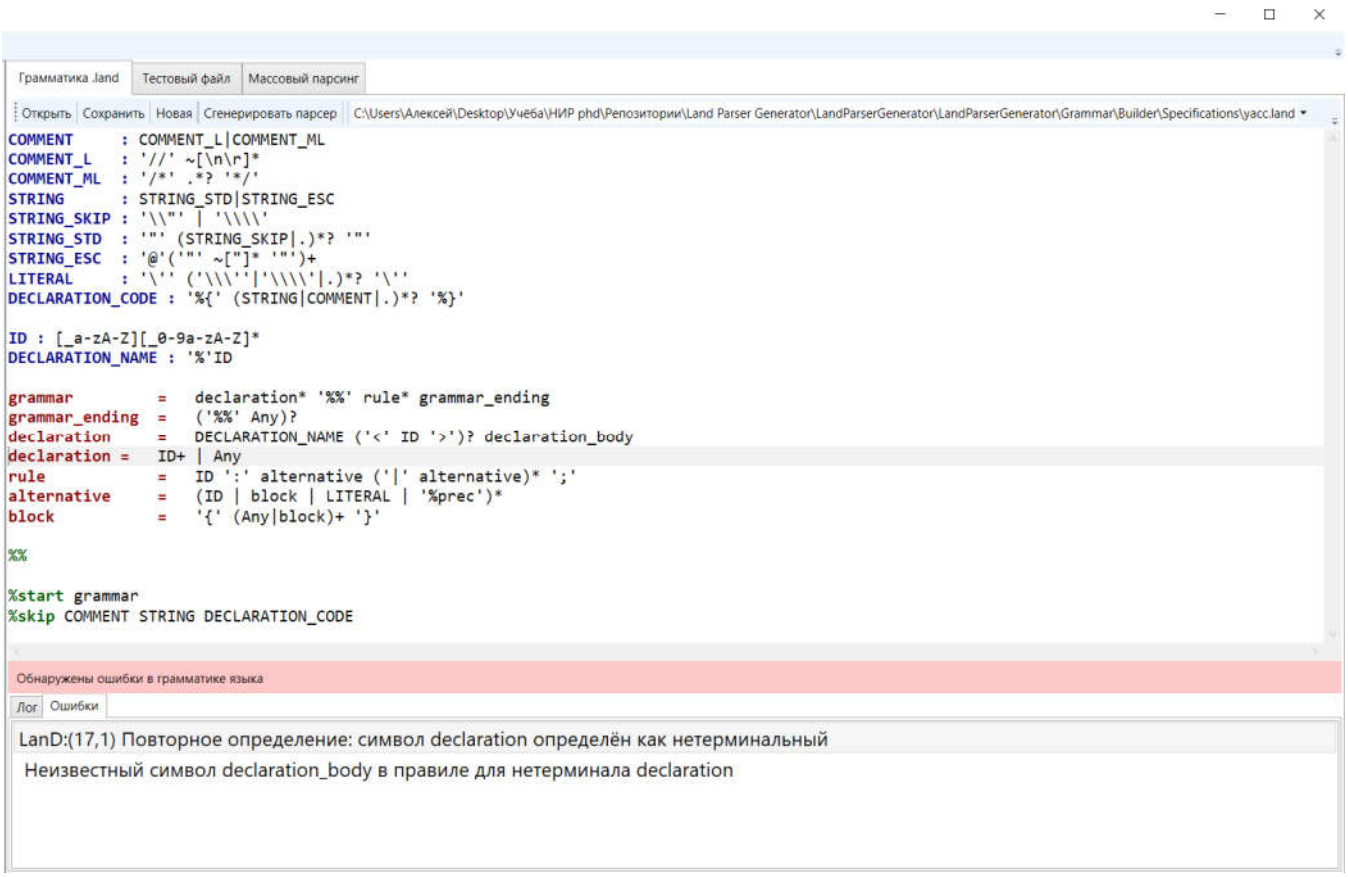
IDE

В IDE есть три основные вкладки.

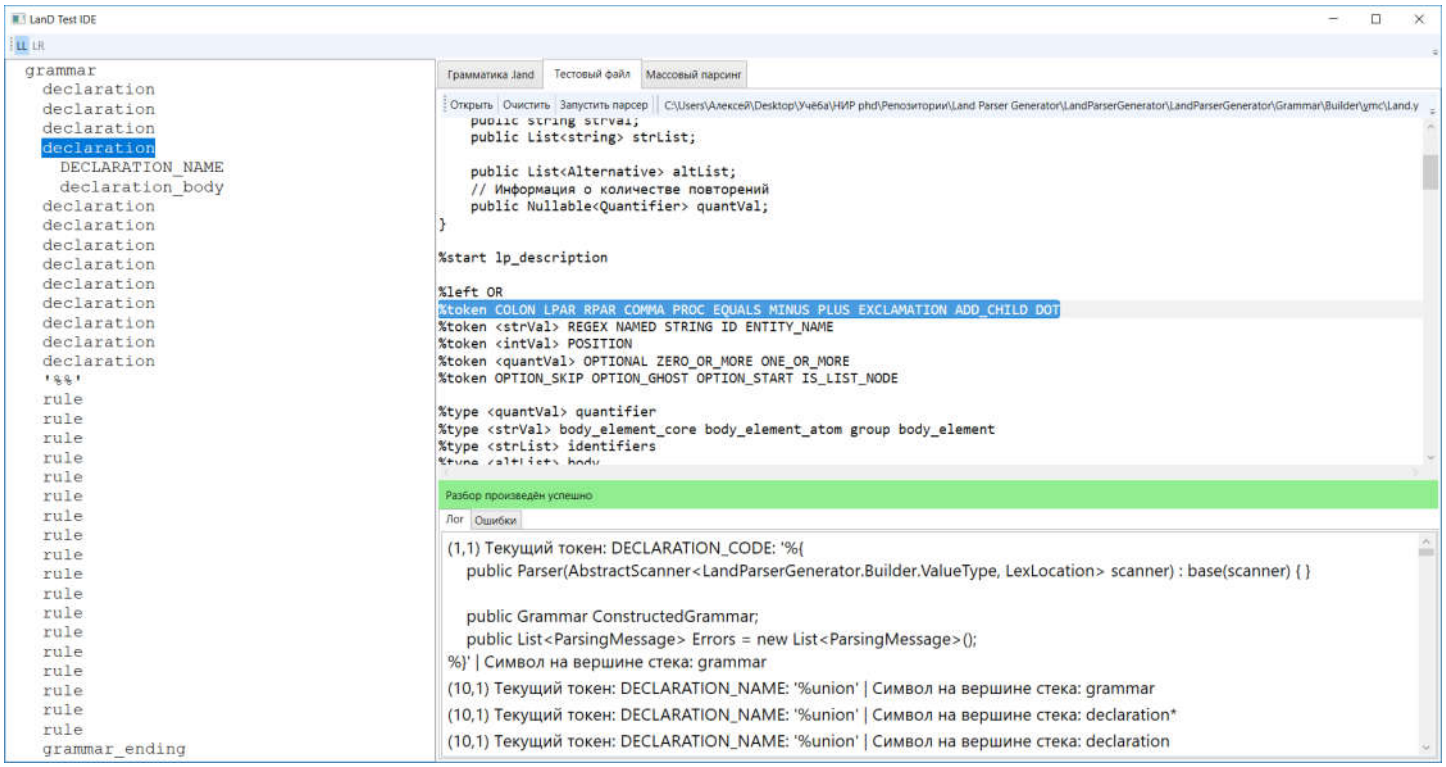


Вкладка **«Грамматика .land»** нужна для открытия имеющейся грамматики, создания новой, внесения изменений в грамматику и её сохранения. При помощи выпадающего списка можно быстро открыть одну из последних открытых land-грамматик. Кнопка «Сгенерировать парсер» запускает парсинг land-грамматики и генерацию 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

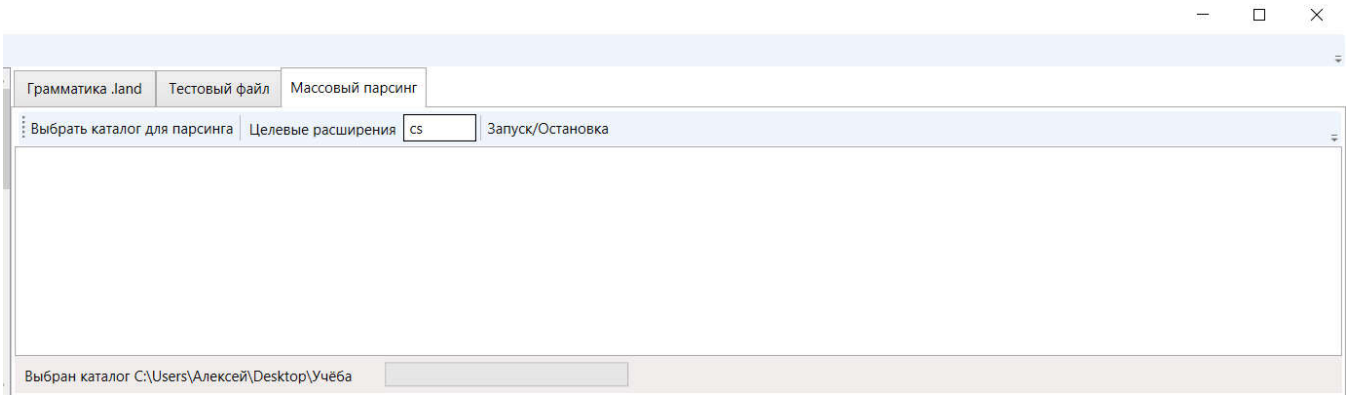
отображается в начале сообщения. Ошибки LanD связаны с содержательной частью грамматики (повторные описания символов, использование не описанных символов и т.п.), ошибки GPPG означают несоответствие описания land-формату с точки зрения синтаксиса. Ошибки Antlr возникают, если регулярные выражения были 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. Хотя они проходят некоторую предобработку ….



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.