Язык чувствительный к регистру

Описание языка генерации парсеров состоит из последовательности директив или опций.

Строковые литералы могут заключаться в одинарные или двойные кавычки.

## Опции

**%CaseInsensitive** – описываемый язык является нечувствительным к регистру

**%CaseSensitive** – описываемый язык является чувствительным к регистру. Опция по-умолчанию.

**%Extension <StringList>**

Задает расширения файлов, обрабатываемые этим парсером. StringList – список расширений без точки, заключенных в кавычки, через пробел.

**%Preprocessor <PreprocessorOptions>**

Задает ключевые слова для директив препроцессора.

PreprocessorOptions:

* Define = <string>
* Undef = <string>
* IfDef = <string>
* IfNDef = <string>
* Else = <string>
* ElIf = <string>
* End = <string>

<string> - строка, заключенная в кавычки. Опции данной директивы могут идти в произвольном порядке, разделенные пробелом. Некоторые опции могут отсутствовать.

**%Namespace <ID>** - пространство имен для сгенерированных парсеров. Значение по-умолчанию: «LWParser». <ID> - идентификатор, может быть как в кавычках, так и без них.

## Директива Token

Описывает лексическую категорию с заданным именем через регулярное выражение в формате Lex.

Синтаксис:

Token <ID> <RegExp>

Пример

Token ID [A-Za-z\_][A-Za-z\_0-9]\*

Если нужно описать несколько равнозначных ключевых слов, нужно использовать директиву Rule.

Rule tkBegin : @"begin" | @"case" | @"try"

## Директива Skip

Данная директива используется для объявления участка пропускаемого текста. Текст пропускается на уровне лексера, как комментарий.

После ключевые слова Skip записываются одна или несколько опций. Опция начинается с ключевого слова, за которым идут параметры опции (если есть), разделенные пробелом.

Список опций:

**Begin**. Обязательная опция. Параметры: одна или более лексем, с которых начинается пропускаемый фрагмент.

**End.** Параметры: ноль или более лексем, с которых начинается пропускаемый фрагмент.

**BeginEnd.** Параметры: одна или более лексем, с которых начинается и которыми заканчивается пропускаемый фрагмент.

**Escape**. Параметры: одна или более Escape-лексем. Если во входном файле будет встречены подряд идущие Escape-лексема и лексема из секции End (BeginEnd), то такая последовательность символов не будет считаться признаком конца пропускаемого участка. Например, для строк C# Escape-лексемой является символ \.

**Nested**. Параметров нет. Данное ключевое слово обозначает, что нужно отслеживать вложенность пропускаемых конструкций, например, скобочную структуру. Не допускается, если лексемы начала и конца конструкции совпадают (например, объявлены через BeginEnd).

**Preprocessor**. Параметров нет. Означает, что данная пропускаемая конструкция может быть директивой препроцессора.

Пример:

Skip BeginEnd '\"' Escape "\\"

Skip Begin '//'

Skip Begin '#' Preprocessor

Skip Begin '{' End '}' nested

## Директива Rule

Описывает правила грамматики языка.

Имеет следующий вид: после ключевого слова Rule следует идентификатор - имя правила, затем двоеточие и затем ветки правила, разделенные символом '|'.

Ветка правила - последовательность терминалов или нетерминалов, перед которыми могут использоваться символы **@** и **#**, а после - символы **?** **\*** **+**.

Пример:

Rule ClassOrNamespace : @Token\* @tkClassOrNamespace @Token\* "{" #Program "}" ;

Rule Field : @Token\* ";" ;

Rule Method : @Token\* MethodBody ;

Символ **?** означает, что данная лексема или правило необязательно (встречается 0 или 1 раз), **\*** означает, что в данном месте может быть последовательность из нуля или более, а **+** значит из одного или более терминалов или нетерминалов.

Несколько вхождений терминалов/нетерминалов можно сгруппировать скобками **[]**, всю группу можно затем пометить одним из символов ? + \*. Внутри скобок можно использовать символ | для описания альтернатив:

Rule R: A [B | C] D

Вместо описания

Rule R: A B D | A C D

В узел дерева, генерируемый данным правилом в качестве имени будут добавлены имена сущностей, помеченных символом **@**, а в качестве подузлов — все сущности, помеченные символом **#**. Символом **@** может быть помечены не только вхождения терминалов/нетерминалов в правой части, но и имя правила.

Если в описании правила нет ни одного вхождения, помеченного символом **@**, то для данного правила не будет генерироваться узел дерева. Подузлы такого узла будут напрямую добавляться в родительский узел.

Стартовым правилом считается правило с именем Program. Если оно не определено, то оно генерируется автоматически и включает в себя список из всех правил, ни разу не встречающиеся в правой части какого-либо другого правила.

В данном примере оно будет иметь вид:

Program: ProgramNode\*

ProgramNode: ClassOrNamespace | Field | Method

## Нетерминалы Any и AnyExcept

Нетерминал Any представляет любую лексему, кроме тех, с которых может начинаться следующий символ в текущем правиле.

Нетерминал AnyExcept по сравнению с Any позволяет расширить список исключаемых лексем за счет параметров: перечисления имен лексем в круглых скобках через запятую.

Определение множества лексем, входящих в Any/AnyExcept осуществляется в пределах правила, в котором этот нетерминал встречается.

Примеры:

Rule R1: Any ";"

Здесь Any представляет любую лексему кроме точки с запятой

Rule R2: Any\* ";"

Данное правило соответствует любому тексту, заканчивающемуся точкой с запятой.

Rule R3: AnyExcept(tkClass, tkMethod) ";"

Здесь AnyExcept представляет любую лексему, кроме точки с запятой, tkClass и tkMethod.

Rule Params : "(" [Any| Params]\* ")"

Здесь Any представляет любую лексему, кроме закрывающей круглой скобки, так как она идет после Any, а также кроме открывающей скобки, так как группа [Any| Params] может повторяться 0 или более раз, а значит после Any может встречаться нетерминал Params, начинающийся с открывающей круглой скобки. Таким образом, данное правило описывает текст, начинающийся с открывающей скобки и заканчивающийся соответствующей закрывающей скобкой.

Rule R4\_1: Any "A" "B"

Rule R4\_2: Any "A"? "B"

В первом случае Any соответствует любой лексеме, кроме "A". Лексема "B" не может встречаться непосредственно после Any и поэтому не влияет на определение Any.

Во втором случае Any соответствует любой лексеме, кроме "A" и "B". Лексема "A" является необязательной, поэтому после Any может встречаться как "A", так и "B".

Rule R5: Any

Rule R6: R5\* ";"

Здесь Any представляет любую лексему, без ограничений, так как нетерминал Any определяется в рамках того правила, в котором он встречается.

Поэтому правило R5 ошибочно (приведет к конфликтам?) – R5\* соответствует всему тексту до конца файла, в том числе всем точкам с запятыми.

# Использование

1. Исходный код описания парсера на языке LightParse подать на вход генератору парсеров ParserGenerator.exe

Параметры командной строки:

ParserGenerator.exe <input> [/l <lexer>] [/y <parser>] [/cs <code>]

<input> - имя входного файла

<lexer> - имя для сгенерированного файла лексера

<parser> - имя для сгенерированного файла парсера

<code> - имя для сгенерированного файла со вспомогательным кодом

В качестве имени по-умолчанию для сгенерированных файлов используется имя входного файла с измененным расширением: .lex – для лексера, .y – для парсера и .cs – для вспомогательного кода.

2. Сгенерированный файл лексера подать на вход генератору лексических анализаторов GPLex (В данный момент в дистрибутив не входит, так как используется оригинальная, немодифицированная версия). Использовать параметр **/stack**

3. Сгенерированный файл парсера подать на вход генератору синтаксических анализаторов Gppg из данного дистрибутива. Использовать параметр **/gplex**

В результате двух предыдущих действий будут получены cs-файлы для лексера и парсера.

4. В Visual Studio создать проект (тип – Class Library). В проект добавить три .cs-файла, полученные на шагах 1-3: вспомогательный код, сгенерированный ParserGenerator(1), описания лексера(2) и парсера(3).

В References для проекта с парсером добавить ссылку на AspectCore.dll

5. Скомпилировать проект.