Compiladores 2023-2
Facultad de Ciencias UNAM
Práctica 3: Parser.

Lourdes del Carmen Gonzáles Huesca Juan Alfonso Garduño Solís
Fausto David Hernández Jasso Juan Pablo Yamamoto Zazueta

21 de febrero
Fecha de Entrega: 8 de marzo

*"Una vez que ha sido revisado y limpiado de caracteres innecesarios para su ejecución, el código es analizado bajo la estructura del lenguaje de alto nivel para obtener una representación de alto nivel mediante un **parse tree**."*

Prelimiares

El objetivo del análisis sintáctico (parseo) es generar un árbol de sintáxis concreta a partir del flujo de tokens obtenido del análisis léxico, también es conocido como parse tree y el hecho de poder generarlo nos va a asegurar que el código que buscamos compilar es correcto sintácticamente porque puede ser generado por un lenguaje formal.
El analizador que vamos a utilizar para construir esta representación intermedia usa la técnica Look-Ahead para ver 1 token por adelantado comenzando de izquierda (L) a derecha (R), por eso pertenece a la clase LALR(1), estos analizadores son los más implementados por su alta eficiencia aunque no son los más poderosos. Estos analizadores hacen uso de las acciones Shift para avanzar en la entrada y Reduce para aplicar una regla de producción en forma inversa, cuando tenemos más de una posibilidad para reducir decimos que tenemos un conflicto reduce/reduce y cuando tenemos la posibilidad de reducir y hacer shift al mismo tiempo existe un conflicto shift/reduce, la implementación de racket permite definir precedencia y asociación de operadores para evitar estos conflictos, ten en cuenta esta tabla para tu implementación:

Operador	Asociación
−, !	-
∗, /, %	Izquierda
+, −	Izquierda
´ =>=>	Izquierda
==! =	Izquierda
&	Izquierda
|	Izquierda

Nota: *En la clase del 21 de febrero no alcanzamos a ver la definición de la asociación y presedencia de los operadores, en la documentación se explica pero de igual manera en la siguiente clase de laboratorio lo veremos con ejemplos.*

Implementación

Para comenzar con esta práctica debes asegurarte que el lexer que resultó de tu primer práctica pueda reconocer correctamente qué tokens generar para cada caso y proveer de ese módulo el lexer y ambos grupos de tokens:

```
(provide contenedores
         vacios
         jelly-lexer)
```

O en su defencto:

```
(provide (all-defined-out))
```

Para utilizar el analizador LALR(1) utilizaremos requeriremos del módulo parser-tools/yacc, al final del documento se encuentra el enlace a la documentación oficial pero si tienes dudas concretas puedes contactarme por Telegram.

Ejercicios

- ***(5 puntos)*** Deine una gramática libre de contexto que sea capaz de generar consistentemente a *jelly*.
- ***(5 puntos)*** A partir de la gramática que definiste en el ejercicio anterior implementa *jelly-parser*, un parser con la estructura vista en clase.
- ***(2 puntos extra)*** Da una implementación para los errores de manera que cuando un token no pueda ser procesado, informe el nombre del token el caso de los tokens vacíos o su valor para los tokens contenedores.

Notas

- Haz push antes de preguntar por una duda que requiera revisar tu código.
- Para dudas rápidas puedes encontrarme en Telegram.
- Si vas a hacer cambio de integrantes en tu equipo avisame antes.
- Documentación del módulo utilizado: parser-tools/yacc.