Construcción de Compiladores - Laboratorio Guiado No. 2

Gabriel Brolo, Bidkar Pojoy

Julio de 2024

1. Introducción

En este laboratorio, aprenderemos a usar ANTLR para compilar un lenguaje de programación. ANTLR (ANother Tool for Language Recognition) es una herramienta poderosa para generar analizadores léxicos y sintácticos. Vamos a crear un lenguaje simple y compilaremos un programa usando ANTLR.

2. Configuración del Entorno

2.1. Docker y Docker Compose

Para evitar la instalación local de ANTLR, utilizaremos Docker. Asegúrese de tener Docker Desktop o Rancher Desktop instalado (si no desea usar Docker, puede instalar ANTLR localmente). También puede instalar todo localmente, pero se recomienda que use Docker/Rancher Desktop. Utilice este código como referencia:

2.1.1. Dockerfile

```
FROM openjdk:11
RUN apt-get update && apt-get install -y antlr4
WORKDIR /usr/src/app
COPY . .
ENTRYPOINT ["antlr4"]
```

2.1.2. docker-compose.yml

```
version: '3'
services:
  antlr:
    build: .
    volumes:
      - .:/usr/src/app
    command: ["bash"]
```

2.2. Instrucciones para Ejecutar el Entorno

1. Construir la imagen de Docker:

   ```
   docker-compose build
   ```

   o

```
docker compose up --build -d
```

2. Iniciar un contenedor interactivo:

```
docker-compose run antlr
```

3. La intención de estas hojas no es explicarle cómo usar Docker. No es tampoco *rocket science*, por lo que se incentiva a que usted investigue como correr los contenedores. No obstante, cualquier duda, estoy a la orden.

3. Crear el Lenguaje

Vamos a definir un lenguaje simple llamado "MiniLang". Aquí está el código para el archivo de gramática 'MiniLang.g4':

3.0.1. MiniLang.g4

```
grammar MiniLang;

prog:   stat+ ;

stat:   expr NEWLINE                # printExpr
    |   ID '=' expr NEWLINE         # assign
    |   NEWLINE                     # blank
    ;

expr:   expr ('*'|'/') expr         # MulDiv
    |   expr ('+'|'-') expr         # AddSub
    |   INT                         # int
    |   ID                          # id
    |   '(' expr ')'                # parens
    ;

MUL : '*' ; // define token for multiplication
DIV : '/' ; // define token for division
ADD : '+' ; // define token for addition
SUB : '-' ; // define token for subtraction
ID  : [a-zA-Z]+ ; // match identifiers
INT : [0-9]+ ; // match integers
NEWLINE:'\r'? '\n' ; // return newlines to parser (is end-statement signal)
WS  : [ \t]+ -> skip ; // toss out whitespace
```

4. Compilando el Lenguaje

4.1. Generando el Analizador Léxico y Sintáctico

Ejecute el siguiente comando dentro del contenedor Docker (o en donde tenga su ambiente) para generar el código del analizador:

```
antlr4 MiniLang.g4 -o gen
```

4.2. Compilando el Código Generado

Utilice el compilador de Java para compilar el código generado:

```
javac gen/*.java
```

4.3. Probando el Lenguaje

Cree un archivo de prueba 'test.minilang' con el siguiente contenido:

```
a = 5
b = 6
a + b * 2
```

Ejecute el siguiente comando para interpretar el archivo de prueba:

```
grun MiniLang prog -gui test.minilang
```

5. Actividades con ANTLR

1. Cree un programa que asigne un valor a una variable.
2. Cree un programa que realice una operación aritmética simple.
3. Experimente con expresiones más complejas.
4. Modifique el lenguaje para incluir la asignación de variables con expresiones aritméticas.
5. Agregue manejo de errores al compilador para detectar tokens inválidos en el programa fuente.
6. Cree un programa que utilice paréntesis para cambiar la precedencia de operadores.
7. Extienda el lenguaje para soportar comentarios de una sola línea.
8. Agregue operadores de comparación (==, !=, ¡, ¿, ¡=, ¿=) al lenguaje.
9. Cree un programa que utilice operadores de comparación.
10. Extienda el lenguaje para soportar estructuras de control como 'if' y 'while'.
11. Cree un programa que utilice una estructura 'if'.
12. Cree un programa que utilice una estructura 'while'.
13. Agregue soporte para funciones definidas por el usuario.
14. Cree un programa que defina y llame a una función.
15. Implemente un sistema de tipos básico que, además de incluir enteros, también incluya cadenas (yo sé que todavía no sabe que es un sistema de tipos, de manera formal, pero para este laboratorio, busque, investigue, haga algo simple, no necesitamos *rocket science todavía* y la respuesta es bastante trivial y a nivel de producción en la gramática).

6. Refrescando la memoria... y atando cabos con ANTLR

1. ¿Qué es un analizador léxico?
   Respuesta: Un analizador léxico, o lexer, es un componente del compilador que convierte una secuencia de caracteres de entrada en una secuencia de tokens.

2. ¿Qué es un analizador sintáctico?
   Respuesta: Un analizador sintáctico, o parser, es un componente del compilador que analiza una secuencia de tokens para determinar su estructura gramatical de acuerdo con una gramática formal.

3. ¿Qué son las reglas léxicas en ANTLR?
   Respuesta: Las reglas léxicas, o lexer rules, definen cómo los caracteres de entrada se agrupan en tokens.

4. ¿Qué son las reglas sintácticas en ANTLR?
   Respuesta: Las reglas sintácticas, o parser rules, definen cómo se estructuran los tokens para formar construcciones sintácticas válidas en el lenguaje de programación.

5. ¿Cómo maneja ANTLR los errores léxicos?
   Respuesta: ANTLR puede personalizar el manejo de errores léxicos sobreescribiendo los métodos de la clase base del lexer para reportar y recuperar errores.

6. ¿Cómo maneja ANTLR los errores sintácticos?
   Respuesta: ANTLR permite manejar errores sintácticos definiendo un 'error handler' personalizado o utilizando el manejo de errores por defecto que proporciona mensajes de error detallados.

7. ¿Qué es un árbol de sintaxis abstracta (AST)?
   Respuesta: Un AST es una representación jerárquica de la estructura sintáctica de un programa, que abstrae detalles de la sintaxis concreta y se centra en la estructura lógica del código.

8. ¿Cómo se puede visualizar un AST generado por ANTLR?
   Respuesta: Utilizando la opción '-gui' de 'grun', se puede visualizar el AST generado por ANTLR en una interfaz gráfica.

9. ¿Qué es la precedencia de operadores y cómo se maneja en ANTLR?
   Respuesta: La precedencia de operadores determina el orden en que se evalúan las operaciones. En ANTLR, se maneja definiendo las reglas sintácticas de manera que las operaciones con mayor precedencia se evalúan primero.

10. ¿Qué es una gramática libre de contexto?
    Respuesta: Una gramática libre de contexto es un tipo de gramática formal en la que las reglas de producción se pueden aplicar independientemente del contexto de los símbolos no terminales.

7. Entregables

- Documento PDF con capturas de pantalla de la ejecución del ambiente en Docker, o de su propio entorno, para realizar las actividades solicitadas.
- El documento debe de incluir el enlace a un repositorio privado de Github con todo el código utilizado para la realización de este laboratorio guiado. Recuerde que debe de compartir todos los repositorios utilizados en este curso con el catedrático y los auxiliares. La información de los usuarios se encuentra en Canvas.

8. Rúbrica de Evaluación

Tarea	**Puntos**
Crear un programa que asigne un valor a una variable	5
Crear un programa que realice una operación aritmética simple	5
Experimentar con expresiones más complejas	10
Modificar el lenguaje para incluir la asignación de variables con expresiones aritméticas	10
Agregar manejo de errores al compilador para detectar tokens inválidos	10
Crear un programa que utilice paréntesis para cambiar la precedencia	5
Extender el lenguaje para soportar comentarios	10
Agregar operadores de comparación al lenguaje	10
Crear un programa que utilice operadores de comparación	5
Extender el lenguaje para soportar estructuras de control	10
Crear un programa que utilice una estructura ‘if‘	5
Crear un programa que utilice una estructura ‘while‘	5
Agregar soporte para funciones definidas por el usuario	10
Crear un programa que defina y llame a una función	5
Implementar un sistema de tipos básico	10
Total	**100**