Construcción de Compiladores - Laboratorio Guiado No. 2

Gabriel Brolo, Bidkar Pojoy

Julio de 2024

Adrian Fulladolsa Palma 21592

Elías Alberto Alvarado Razón 21808

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.

4. Actividades con ANTLR

1. Cree un programa que asigne un valor a una variable.

Código generado en archivo program/Actividades/actividad1.txt

```
C/C++
a = 10
```

```
appuser@599a496a1045:/program$ python3 Driver.py Actividades/actividad1.txt
appuser@599a496a1045:/program$
```

2. Cree un programa que realice una operación aritmética simple.

Código generado en archivo program/Actividades/actividad2.txt

```
C/C++
2 + 1
```

```
appuser@e72b48e8af1a:/program$ python3 Driver.py Actividades/actividad2.txt
appuser@e72b48e8af1a:/program$
```

3. Experimente con expresiones más complejas.

Código generado en archivo program/Actividades/actividad1.txt

```
C/C++
5 + 6 * 2 / 4
```

```
appuser@599a496a1045:/program$ python3 Driver.py Actividades/actividad3.txt
appuser@599a496a1045:/program$
```

4. Modifique el lenguaje para incluir la asignación de variables con expresiones aritméticas.

Código generado en archivo program/Actividades/actividad4.txt

```
C/C++
a = 1 + 2
```

```
appuser@e72b48e8af1a:/program$ python3 Driver.py Actividades/actividad4.txt
appuser@e72b48e8af1a:/program$
```

5. Agregue manejo de errores al compilador para detectar tokens inválidos en el programa fuente.

Código generado en archivo program/Actividades/actividad5.txt

```
C/C++
a = #
```

```
appuser@599a496a1045:/program$ python3 Driver.py Actividades/actividad5.txt
Error: En línea 1, columna 4: token recognition error at: '#'. Token no reconocido.
Error: En línea 1, columna 5: mismatched input '\r\n' expecting {'(', ID, INT}. Token no reconocido.
appuser@599a496a1045:/program$
```

6. Cree un programa que utilice paréntesis para cambiar la precedencia de operadores.

Código generado en archivo program/Actividades/actividad4.txt

```
C/C++
(1 + 2) * 1 / (4 - 2)
```

```
appuser@e72b48e8af1a:/program$ python3 Driver.py Actividades/actividad6.txt
appuser@e72b48e8af1a:/program$
```

7. Extienda el lenguaje para soportar comentarios de una sola línea.

Modificaciones realizadas en archivo program/MiniLang.g4

```
C/C++
stat:   expr NEWLINE                # printExpr
    |   ID '=' expr NEWLINE         # assign
    |   NEWLINE                     # blank
    |   COMMENT NEWLINE?             # comment
    ;

COMMENT : '//' ~[\r\n]* -> skip ; // match comment
```

Código generado en archivo program/Actividades/actividad7.txt

```
C/C++
a = 5  //comentario prueba
// comentario segundo
```

```
appuser@e72b48e8af1a:/program$ python3 Driver.py Actividades/actividad7.txt
appuser@e72b48e8af1a:/program$
```

8. Agregue operadores de comparación (==, !=, ¡, ¿, ¡=, ¿=) al lenguaje.
Modificaciones realizadas en archivo program/MiniLang.g4

```
C/C++
expr:   expr ('==' | '!=' | '<' | '>' | '<=' | '>=') expr   # Compare
    |   expr ('*'|'/') expr                                 # MulDiv
    |   expr ('+'|'-') expr                                 # AddSub
    |   INT                                                 # int
    |   ID                                                  # id
    |   '(' expr ')'                                        # parens
    ;

EQ:     '==';
NEQ:    '!=';
LT:     '<';
GT:     '>';
LE:     '<=';
GE:     '>=';
```

9. Cree un programa que utilice operadores de comparación.
Código generado en archivo program/Actividades/actividad9.txt

```
C/C++
a = 1 < 2
b = 1 != 2
2 == 2
3 >= 2
2 > 1
124 <= 123
```

```
appuser@e72b48e8af1a:/program$ python3 Driver.py Actividades/actividad9.txt
appuser@e72b48e8af1a:/program$
```

10. Extienda el lenguaje para soportar estructuras de control como ‘if‘ y ‘while‘.

Modificaciones realizadas en archivo program/MiniLang.g4

```
C/C++
stat:   expr NEWLINE                                        # printExpr
    |   ID '=' expr NEWLINE                                 # assign
    |   'if' expr 'then' block ('else' block)? 'endif'      # ifElse
    |   'while' expr 'do' block 'endwhile'                  # whileLoop
    |   NEWLINE                                             # blank
    |   COMMENT NEWLINE?                                    # comment
    ;

block:  (stat | NEWLINE)*;
```

11. Cree un programa que utilice una estructura ‘if‘.

Código generado en archivo program/Actividades/actividad11.txt

```
C/C++
i = 0

if i > 10 then
i = i + 10
endif
```

```
appuser@e72b48e8af1a:/program$ python3 Driver.py Actividades/actividad11.txt
appuser@e72b48e8af1a:/program$
```

12. Cree un programa que utilice una estructura ‘while‘.

Código generado en archivo program/Actividades/actividad12.txt

```
C/C++
i = 0
while i < 10 then
i = i + 1
endwhile
```

```
appuser@e72b48e8af1a:/program$ python3 Driver.py Actividades/actividad12.txt
appuser@e72b48e8af1a:/program$
```

13. Agregue soporte para funciones definidas por el usuario.

Modificaciones realizadas en archivo program/MiniLang.g4

```
C/C++
stat:   expr NEWLINE                                    # printExpr
    |   ID '=' expr NEWLINE                           # assign
    |   'if' expr 'then' block ('else' block)? 'endif'   # ifElse
    |   'while' expr 'do' block 'endwhile'                 # whileLoop
    |   NEWLINE                                     # blank
    |   COMMENT NEWLINE?              # comment
    |   funcDecl NEWLINE                                # functionDefine
    ;

funcDecl: 'def' ID '(' paramList? ')' block 'enddef' ;

paramList: ID (',' ID)* ;

funcCall: ID '(' argList? ')' ;

argList: expr (',' expr)* ;
```

14. Cree un programa que defina y llame a una función.

Código generado en archivo program/Actividades/actividad14.txt

```
C/C++

 def add(a, b)
     result = a + b
 enddef


 x = add(2, 3)
 y = add(x, 4)
```

```
appuser@e72b48e8af1a:/program$ python3 Driver.py Actividades/actividad14.txt
appuser@e72b48e8af1a:/program$
```

15. Implemente un sistema de tipos básico que, además de incluir enteros, también incluya cadenas (yo se 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).

    En esta agregamos el token STRING al miniLang de esta manera:

```
C/C++
expr:   expr ('==' | '!=' | '<' | '>' | '<=' | '>=') expr   # Compare
    |   expr ('*'|'/') expr                                  # MulDiv
    |   expr ('+'|'-') expr                                  # AddSub
    |   INT                                                  # int
    |   STRING                                               # string
    |   ID                                                   # id
    |   '(' expr ')'                                         # parens
    ;

STRING : '"' (~["\r\n])* '"' ;
```

De esta manera se busca además de expresiones con tipo INT que se define como uno o más dígitos de 0 a 9, se buscan conjuntos de caracteres encerrados entre comillas dobles, gracias a esto el siguiente código es válido:

```
C/C++
"a"
i = "avv"
```

```
appuser@e72b48e8af1a:/program$ python3 Driver.py Actividades/actividad15.txt
appuser@e72b48e8af1a:/program$
```

Retornando un resultado de aprobación, en cambio si eliminamos una de las comillas dobles se obtiene error

```
C/C++
"a"
i = "avv
```

```
appuser@e72b48e8af1a:/program$ python3 Driver.py Actividades/actividad15.txt
line 2:4 token recognition error at: '"avv\n'
line 3:0 mismatched input '\n' expecting {'(', STRING, ID, INT}
```

```
C/C++
"a"
i = avv"
```

```
appuser@e72b48e8af1a:/program$ python3 Driver.py Actividades/actividad15.txt
line 2:7 token recognition error at: '"\n'
```

Repositorio de GitHub:

https://github.com/adrianfulla/Compilers-2024/tree/Lab2

6. 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.

7. Rúbrica

La siguiente rúbrica ser´a utilizada para asignar puntos a las tareas anteriores:

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 comparaci´on al lenguaje	10
Crear un programa que utilice operadores de comparaci´on	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 funci´on	5
Implementar un sistema de tipos b´asico	10
Total	100