TEC | Tecnológico de Costa Rica

Escuela de Computación

Compiladores e Intérpretes

Proyecto 1

Profesor:

Allan Rodríguez Dávila

Estudiante:

Fabian Villalobos Rodríguez
Carné: 2018254323

Aivy Yisela Masis Rivera
Carné: 2016253759

Cartago, Costa Rica

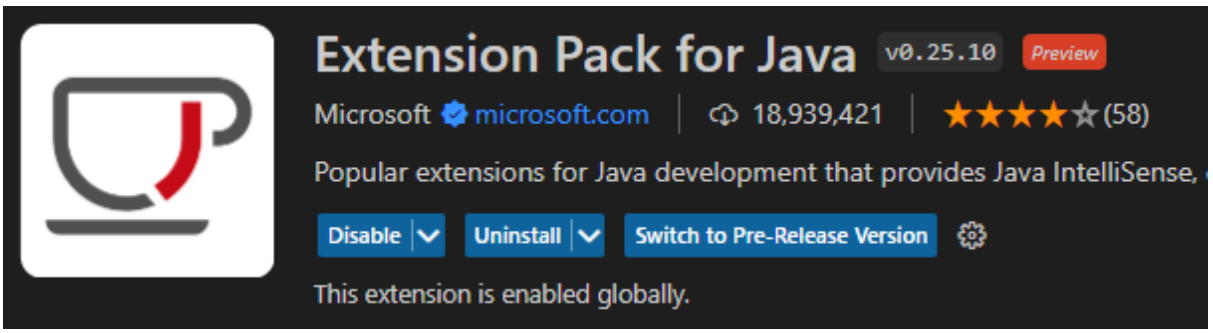
02 de Abril del 2023

I semestre, 2023

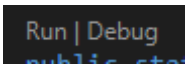
Manual de usuario:

Para este caso se utilizará VSCode para la ejecución del proyecto.

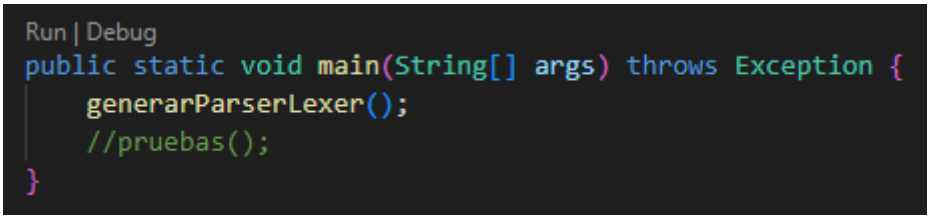
Se deben instalar algunas extensiones de VSC primero:



Luego desde el archivo src/App.java dentro del repositorio en la función main deberá aparecer dos palabras encima en gris:



Con el botón de Run se puede ejecutar el proyecto desde ahí, como paso número 1 para poder ejecutar el analizador se debe crear la gramática, para esto se debe correr el main de la siguiente forma:



Note que la función generarParserLexer() esta no comentada y pruebas() es comentada, esto es para generar la gramática primero, una vez esto compilado se puede proceder a probar la gramática.

Para esto se necesita un archivo de texto con un programa simple, este archivo debe de ubicarse en src/pruebas , para este caso utilizaremos el archivo prueba_parser2.txt entonces se vería de la siguiente forma:

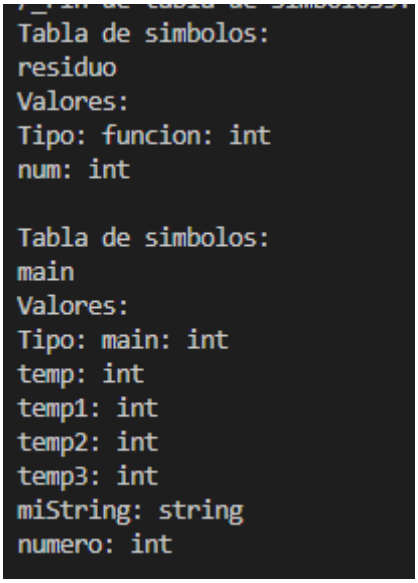
```
int residuo (int num){
        int num = 1$
        return num$
    }

int main() {
    int temp = 12 ** residuo(num)$
    if (temp == 12 # !xd)
        {
            int temp1 = read()$
            int temp2 = temp - temp$
            break$
        }
    else
        {
            int temp3 = var+23$
        }
    string miString = "num"$
    int numero = 45 / 34 ~ 4$
    print("numero")$
}
```

Una vez el archivo de texto esté creado se debe modificar el la función de main en el archivo App.java de la siguiente forma:

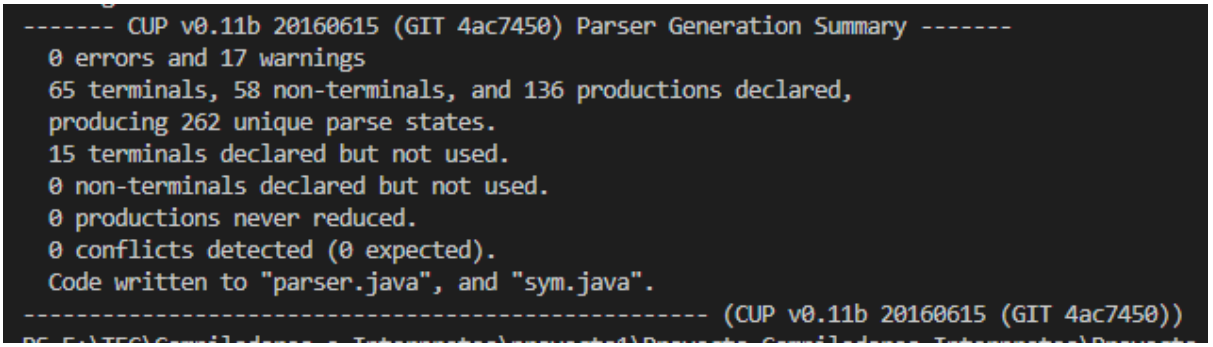
```
Run | Debug
public static void main(String[] args) throws Exception {
    //generarParserLexer();
    pruebas();
}
```

Note que ahora las líneas comentadas son al revez. Ahora se presiona el botón de Run otra vez y en la terminal se ve de la siguiente forma:



Pruebas de funcionalidad: incluir screenshots.

Verificación de la correcta compilación del parser.cup y la gramática:



Documento de texto a comprobar:

```
int residuo (int num){
        int num = 1$
        return num$
    }

int main() {
    int temp = 12 ** residuo(num)$
    if (temp == 12 # !xd)
        {
            int temp1 = read()$
            int temp2 = temp - temp$
            break$
        }
    else
        {
            int temp3 = var+23$
        }
    string miString = "num"$
    int numero = 45 / 34 ~ 4$
    print("numero")$
}
```

Resultado:

Información al respecto de la primer función:

```
/_Nueva tabla de simbolos_/:  residuo
PARSER: identificardor: num
PARSER: sentencia: 1

/_Fin de tabla de simbolos2: _/null(int num) {null$return num$}
```

Segunda función, función de main:

```
/_Nueva tabla de simbolos_/:  main
PARSER: identificardor: temp
PARSER: sentencia: 12 ** residuo(num)

PARSER: identificardor: temp1
PARSER: sentencia: read()

PARSER: identificardor: temp2
PARSER: sentencia: temp - temp

null$ null$
PARSER: identificardor: temp3
PARSER: sentencia: var + 23

null$ if (temp == 12 # xd){null$ null$break$} else{null$}
PARSER: identificardor: miString
PARSER: sentencia: num

null$ if (temp == 12 # xd){null$ null$break$} else{null$} null$
PARSER: identificardor: numero
PARSER: sentencia: 45 / 34 ~ 4

null$ if (temp == 12 # xd){null$ null$break$} else{null$} null$ null$
null$ if (temp == 12 # xd){null$ null$break$} else{null$} null$ null$ print(numero)$
/_Fin de tabla de simbolos3: _/ : main
```

Tablas de símbolos por función:

```
Tabla de simbolos:
residuo
Valores:
Tipo: funcion: int
num: int

Tabla de simbolos:
main
Valores:
Tipo: main: int
temp: int
temp1: int
temp2: int
temp3: int
miString: string
numero: int
```

Descripción del problema.

Se debe crear una gramática basada en las especificaciones dadas, utilizando las herramientas de Java Cup y JFlex para la generación de dicha gramática, manejo

de errores, creación del analizador lexico y sintactico así como la tabla de símbolos que genera a la hora de analizar un programa.

Diseño del programa:

Para el diseño de la gramática se tienen las siguientes definiciones:

Producciones
#Signos y operandos

aumento --> \++
decremento--> \--
multiplicacion --> *
suma --> \+
resta --> \-
equivalente --> \=
negacion --> \!
finalExpre --> \$
division --> \/
modulo --> \~
potencia --> **
aperturaB --> \{
cerraduraB --> \}
aperturaC --> \[
cerraduraC --> \]
aperturaP --> \(
cerraduraP --> \)
comentarioL --> \@
comentApert --> \/_
comentCerrad --> _/
conjuncion --> \^
disyuncion --> \#
True --> 0 | “true”
False --> 1 | “false”

#Bases (definiciones básicas para la formulación del resto)

letra --> [a-zA-Z_]
id --> letra(letra|\d)*
caracter --> \S{1}
string --> caracter+
digito --> [0-9]
digitoN --> [1-9]
bool --> True | False
menor --> \<
mayor --> \>
menorIgual --> \=>
mayorIgual --> \=<
diferente --> \!=
igual --> \==

#Agrupaciones

tipo --> 'int' | 'bool' | 'string' | 'char' | 'float'
tipoNum --> 'int' | 'float'
tipoFuncion --> tipoNum | 'bool'

operacionArit --> suma | resta | division | multiplicacion | modulo | potencia
operacionRac --> menor | mayor | menorIgual | mayorIgual | diferente | igual
operacionLog --> diferente | igual | conjuncion | disjuncion

#Expresiones complejas

numeroE --> resta? 0 | digito Ndigito*
numeroF --> resta? 0.digito* | digito Ndigito*\.digito*

grupoArreglo --> numeroE | caracter
lista --> grupoArreglo (\,grupoArreglo)*
arregloSim --> tipoArreglo id aperturaC numeroE cerraduraC
arregloAsig--> arregloSim equivalente aperturaC (grupoarreglo | lista) cerraduraC
arregloInd --> id aperturaC numeroE cerraduraC equivalente grupoArreglo

operando --> numeroE | numeroF | id |funcionAsig
operacionNumSim --> operando operacionArit operando
operacionNum --> operacionNumSim (operacionArit operando)*
variableNum --> tipoNum id equivalente ((numeroE | numeroF) | operacionNum | id)
operacionSumUna --> (aumento operando) | (aumento variableNum)
operacionResUna --> (decremento operando) | (decremento variableNum)
operacionRacSim --> operando operacionRac operando
operacionRacNum --> operacionRacSim (operacionRac operando)*

operandoBool --> expNegada | bool | operacionRacNum | id | funcionAsig
condicionSim --> operandoBool operacionLog operandoBool
condicion --> condicionSim (operacionLog condicionSim)*
variableB --> "bool" id equivalente condicion
expNegada --> negacion(condicion | variableB | operandoBool)

variable --> tipo id | arreglo
variableAsig --> (variable equivalente (funcionAsig | caracter | string | id)) | variableNum | variableB | arregloAsig

if --> "if" aperturaP condicion* cerraduraP aperturaB bloque break? cerraduraB
elif --> "elif" aperturaP condicion* cerraduraP aperturaB bloque break? cerraduraB
else --> "else" aperturaB bloque break? cerraduraB
bloqueIf --> if elif? else?

do --> "do" aperturaB bloque break? cerraduraB
while --> "while" aperturaP condicion cerraduraP
bloqueDoWhile --> do while
break --> "break"

for --> "for" (variable | variableAsig) "in" id (operacionSumaUna | operacionRestaUna) aperturaB bloque break? crerraduraB
bloqueFor --> for aperturaB bloque cerraduraB

parametro --> id (\,id)*
funcion --> tipoFuncion id aperturaP (id | parametro)? cerraduraP aperturaB bloque return cerraduraB

funcionAsig --> id aperturaP (id | parametro)? cerraduraP
declaracion --> (variable | variableAsig | operacionNum | operacionRac | arreglo | arregloAsig | arregloInd | input | print | funcionAsig)finalExpre
sentencia --> declaracion | bloqueIf | funcion | bloqueDoWhile | bloqueFor
bloque --> sentencia*
return --> "return" (id | numeroE | numeroF)

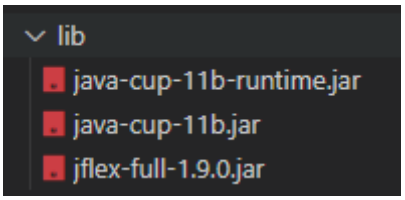
funMain --> tipo "main" aperturaP (id | parametro)? cerraduraP aperturaB bloque cerraduraB
comentarioLineal --> comentarioL string

comentarioBloque --> comentApert string comentCerrad
print --> "print" aperturaP string cerraduraP
inputVar --> string | tipoNum
input --> "input" aperturaP inputVar cerraduraP

Esta gramática se pasó a jflex y cup para generar el analizador, y luego se tomo como ejemplo las explicaciones del profesor en clases para la creación del analizador y el manejo de tablas de símbolos.

Librerías usadas:

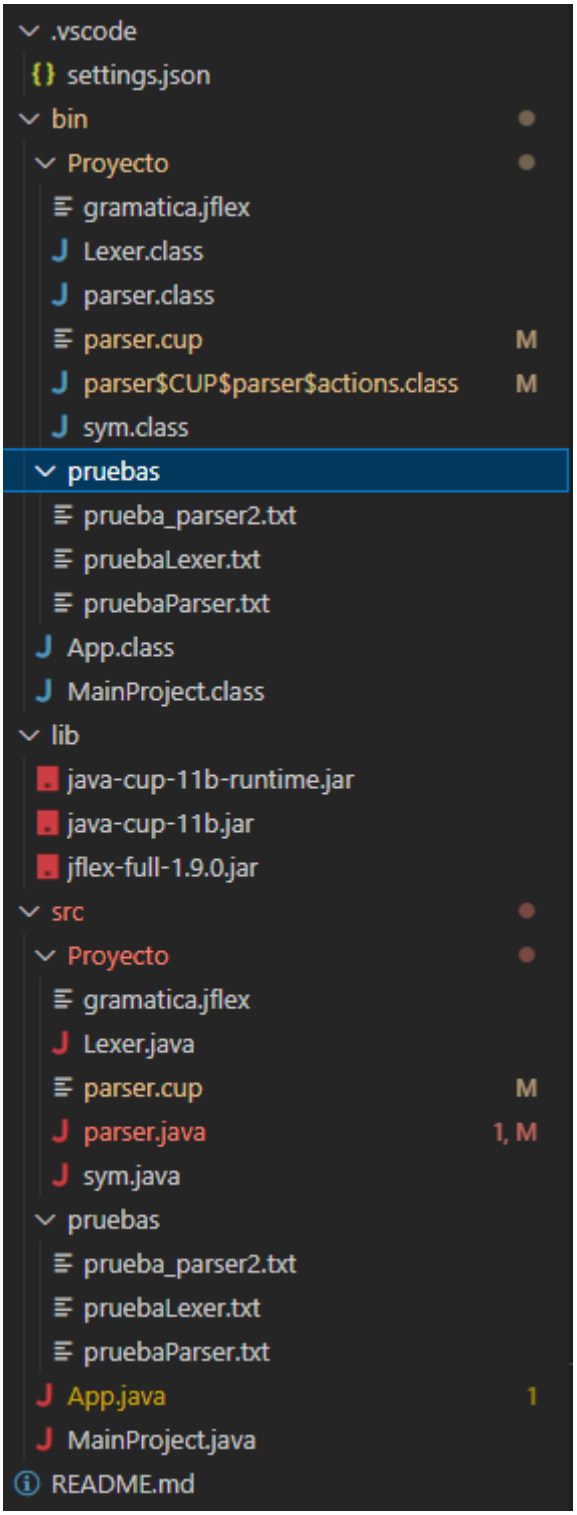
Se utilizaron las siguientes librerías de java para la creación del analizador:



Para el manejo de la tabla de símbolos:

```
/*Imports para la tabla de simbolos*/
import java.util.HashMap;
import java.util.Hash;
import java.util.ArrayList;
```

Estructura del proyecto:



Análisis de resultados:

-La creación de la gramática y la utilización de las herramientas fue exitoso

-La creación de la tabla de símbolos fue creada con algunos errores, esto porque estábamos faltos de tiempo pero se puede ver la funcionalidad que tiene.

-Hay algunos detalles con respecto a la creación de variables, booleanos y otros, donde por falta de tiempo no se pudieron corregir pero están funcionando aunque puede ser un poco rígido.

Bitácora

https://github.com/MaickF/Proyecto_Compiladores_Interpretes