Escuela de Ingeniería en Computación
Ingeniería en Computación
IC5701 - Compiladores e Intérpretes

Compilador en C

Programado En Java Usando La Herramienta JCup

Daniel Araya Sambucci
danielarayasambucci@estudiantec.cr
2020207809

Andrés Masís Rojas
andres.masis.rojas@estudiantec.cr
2020127158

Esteban Leiva Montenegro
estebanlm@estudiantec.cr
2020426227

Cartago, Costa Rica
Septiembre 2022

Índice general

Capítulo 1

Introducción

Los compiladores son programas de una computadora que traducen un lenguaje a otro. Un compilador toma como su entrada un programa escrito en un lenguaje fuente y produce un programa equivalente escrito en su lenguaje objetivo (Louden, 2004). Para que un programa llegue a ser compilado o traducido deberá pasar por diferentes procesos; algunos de estos procesos pueden ser, el analizador léxico, analizador sintáctico y semántico. Para este proyecto se desarrollará la segunda parte de un compilador basado en el lenguaje de programación C, el analizador sintáctico ("Parser"). Se desarrollará a partir de la primera entrega funcional del analizador léxico ("Scanner") generado en una entrega anterior.

El proceso de análisis léxico se refiere a validar que todos los tokens se permuten de manera correcta. En este caso, se refiere a estructura y orden de como se escriben los ciclos, declaraciones, etc. El análisis del significado de dichas estructuras es realizaado posteriormente en la etapa semántica.

Para la realización de este proyecto, se programó en Java y se usó la herramienta Cup en conjunto con la herramienta JFlex. La librería Cup hace posible que el programador declare la gramática y trabaje en conjunto con el JFlex para realizaar análisis sintáctico.

1.1. Estrategia de Solución

Para el desarrollo de la solución se hizo un uso intensivo de la herramienta Cup. Esta herramienta se encarga de generar los first, follow y tabla de parseo correspondientes a partir de una gramática dada. Por ello, la solución se centra en definir una gramática correcta con la estructura que el Cup necesita para hacer el análisis sintáctico.

Inicialmente se definen los terminales. Estos se refieren a las unidades básicas que ya no se pueden dividir. Para este proyecto se tienen, definiendolos de una forma general en grupos, identificadores, palabras reservadas, operadores y literales. A su vez, el error es manejado como un terminal pero este es definido

y tratado de una forma particular por Cup, para poder manejar el retorno de dicho valor. Esta sección es importante pues con base en ella se crean las producciones. Eventualmente las producciones se componen de tokens y aterrizan en estos. Así mismo, estas se definen inicialmente en el sigma y los identificadores los alimenta el JFlex.

A partir de los terminales se definen las producciones. Las producciones indican las permutaciones permitidas de tokens y como estos se debe relacionar entre sí. Inicialmente se declaran cuales son todos los no terminales que aparecerán y después se describe el comportamiento de cada producción.

El archivo más importante de la solución es "parser.cuo". Este es el que aloja todas las definiciones de los terminales y no terminales, producciones y toda la gramática en general. Por ende, este archivo es el pilar estructural mediante el cual se genera el Analizador Sintáctico. A continuación se va a dar un listado general de las distintas producciones que se definieron. Para ver cada expresión regular en detalle puede revisar el código fuente, ya que por la naturaleza de este documento sería tedioso para el lector entrar en mucho detalle acá.

El programa inicia en la producción PROGRAMA, donde abre el camino a que las distintas partes se puedan ejecutar varias veces. Posteriormente se definieron las declaraciones. Esto con el fin de crear una nueva variable o función. Siempre se inicia con un tipo de dato y un identificador. Antes del tipo de dato puede venir el token const para indicar que es una constante. Después de eso vienen múltiples producciones para representar los distintos caminos.

Si al inicio vino const se va de una vez por la producción de variables. Con las variables siempre tiene que haber al menos un identificador. Despues de ese identificador inicial puede venir una asignación de valor o una lista de más identificadores con o sin asignación de valor. La lista de identificadores se separa por comas. Siempre se cierra con punto y coma.

Si se trata de una función se va por este camino si encuentra un paréntesis izquierdo después del identificador. Dentro de los paréntesis se llama a la producción de parámetros. Esta producción acepta que no venga nada o varios parametros. En caso de haber más de un parámetro se separa por comas. Si vienen parámetros, esta producción espera un tipo de dato y después un identificador. Siempre se cierra con paréntesis derecho.

Después trabaja con las asignaciones a variables ya creadas. Para ello se crea una producción específicamente con los operadores de asignación. Estos son igual, suma igual, resta igual, multiplicacion igual, division igual y modulo igual. Esta producción se llama entre un identificador y un literal, identificador o expresión.

Posteriormente, se trabaja con las expresiones condicionales propias de los if o ciclos. Acá se especifican las expresiones con or y con and. También se dan las condiciones de igual o distinto. En otra producción se definen los casos para mayor que, menor que, mayor estricto y menor estricto.

La expresión más simple que puede haber está en ExpPrimaria. Eventualmente todas las producciones de expresiones aterrizan en ExpPrimaria. Esta incluye identificadores, literales, contantes y paréntesis.

Después se trabajan los if-else. Acá se entra con el token first if. Posteriormente espera un paréntesis. De ahí llama la producción de expresión. Después se cierra con paretesis derecho. Después de eso viene una producción statement que se encarga de abrir y cerrar los corchetes y todo lo que está en medio de ellos.

Como puede que después no venga un else, o venga uno, se tiene una producción que maneja eso. Esta es la producción seleccionStatementIf y también seleccionStatementIfElse.

Posteriormente tenemos que manejar las estructuras iterativas, o sea los ciclos como el for o while. En sintáxis, estas son muy parecidas al if, viene el token parentesis los cuales contienen una expresión booleana y despues corchetes con código. La diferencias es que no hay producción para un else y se cambia el token inicial por el respectivo, por ejemplo while.

Capítulo 2

Casos de Prueba

El compilador cumple en su totalidad con las funcionalidades descritas en el enunciado del proyecto. El Parser (Analizador Sintáctico) identifica varios tipos de producciones:

- Declaraciones
- Asignaciones
- Expresiones
- Estructuras de control

Cada uno de esos grandes grupos será desarrollado a continuación, mostrando los resultados que generó la funcionalidad definida para cada apartado, mediante casos de uso, así como otros apartados que se consideró pertinentes demostrar.

Empezando por el primer aspecto a tomar en cuenta, se encuentran las asignaciones. El compilador es capaz de detectar que se comenzó una asignación cuando comienza con la palabra reservada const o un tipo de dato. De ahí avanza esperando un identificador. Si viene un parentesis es una función. Si no es una variable que debe cerrar con punto y coma o una lista. Si comienza así pero no termina como se indicá el compilador detecta el error.

Con la asignación, para que entre debe comenzar con uno de los identificadores previamente captados por el Scanner, después un operador de asignación y finalmente un valor y punto y coma. Si entra a esta gramática pero se termina violando entonces el parser da error.

Con los if, se prueba que cada vez que se encuentra esta palabra reservada después haya un parétesis izquierdo. Este debe ser seguido por una expresión booleana y se cierra con paréntesis derecho. Después hay corcheters y un cuerpo. Despues puede haber un else o no.

Con los ciclos se tiene una estructura similar al if. Por ejemplo, el while es, la palabra reservada y un parentésis con expresión booleana. No obstante, el for tiene una sintáxis muy particular. Esta es for seguido por parentesis, corechetes y un cuerpo entre estos. Pero lo que el for espera dentro de los paréntesis no es

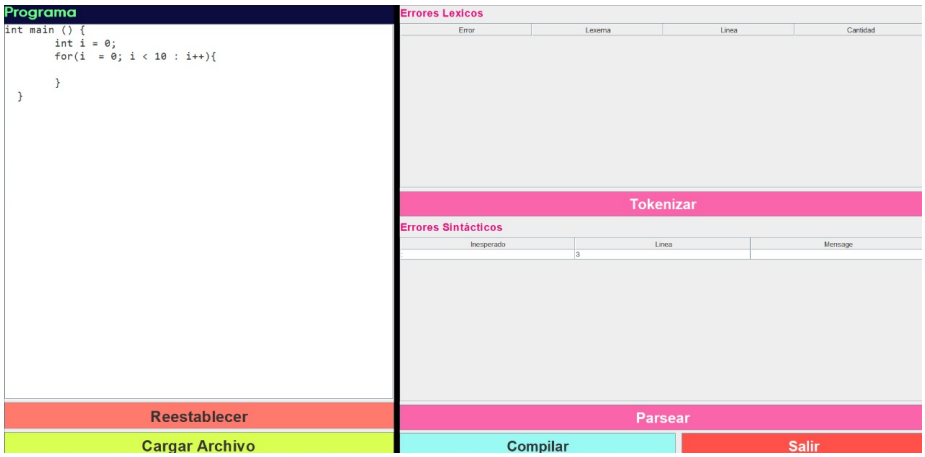

Figura 2.1: Error de if porque esepraba una expresión

solo una expresión booleana. En este caso es una declaración de variable, punto y coma, expresión boolean, punto y coma y aumento o decremento de la variable para cerrar el paréntesis.

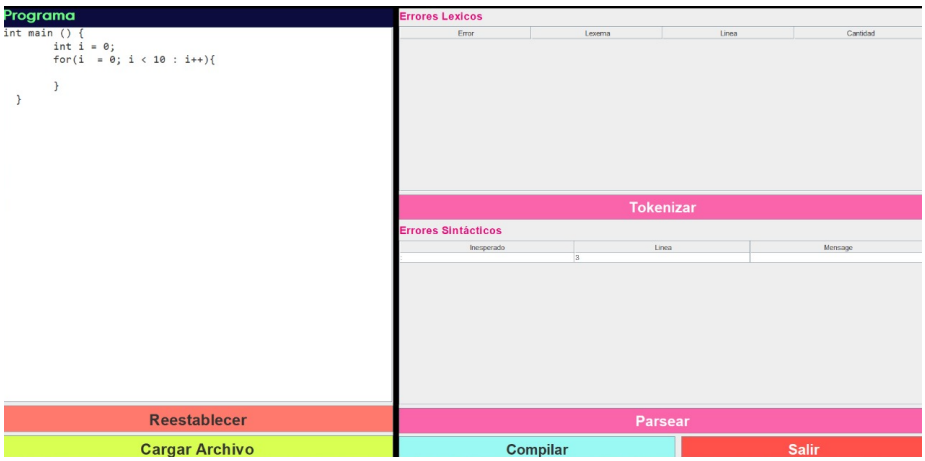

Figura 2.2: Error porque tenía : en lugar de ;

Recuerde que cualquier intrucción en C debe terminar con punto y coma. Acá se prueba que nuestro compilador detecta esos errores.

También se prueba que nuestro compliador detecta correctamente la estrucutura del switch-case. Ya que se espera que los casos tengan alguna acción y que se cierre con default.

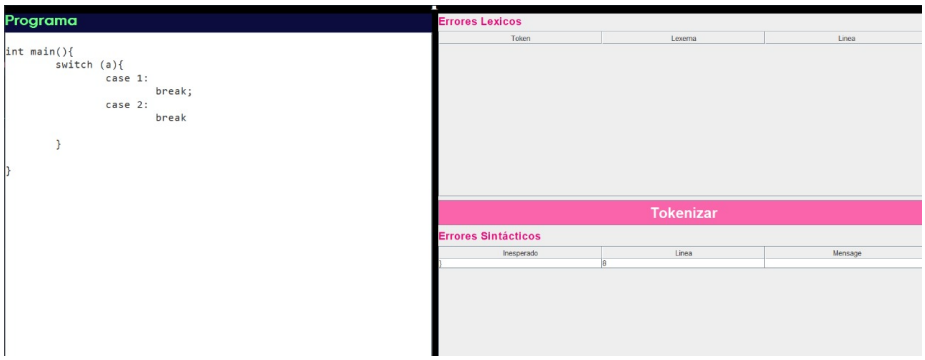

Figura 2.3: Error porque no hay ; despues del break

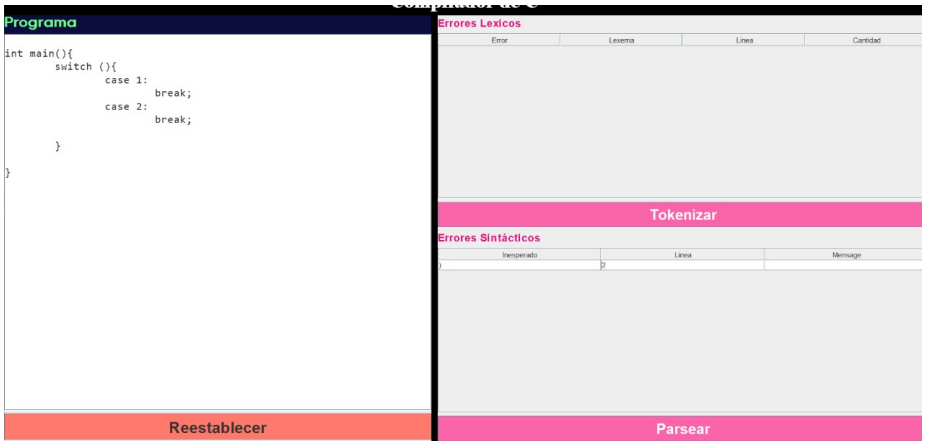

Figura 2.4: Error porque no hay expresiones en los casos ni default

2.1. Código fuente

A continuación se mostrará el código fuente del archivo.cup con la última versión funcional

```
1  package CompiladorC;
2
3  import java_cup.runtime.*;
4  import java.io.*;
5  import java.util.*;
6  import java_cup.runtime.XMLElement.*;
7  import javax.xml.stream.*;
8  import javax.xml.transform.*;
9  import javax.xml.transform.stream.*;
10
11
12 class Parser;
13
```

```
14  parser code {:
15
16      public static int errors = 0;
17
18      public void syntax_error(Symbol cur_token){
19   System.err.println("Syntax error at "+cur_token);
20      }
21      public static void newScope(){
22   typenames.push(new HashSet<String>());
23      }
24      public static void deleteScope(){
25   typenames.pop();
26      }
27      public static boolean lookupType(String name){
28   for (HashSet<String> scope: typenames)
29       if (scope.contains(name)) return true;
30   return false;
31      }
32      public static void addType(String name){
33   typenames.peek().add(name);
34      }
35      public static LinkedList<HashSet<String>> typenames = new
            LinkedList<HashSet<String>>();
36      public Parser(LexerCupAnalyzer lex, ComplexSymbolFactory sf) {
37   super(lex,sf);
38      }
39
40      public static void main(String args[]) {
41   try {
42       ComplexSymbolFactory csf = new ComplexSymbolFactory();
43       // create a buffering scanner wrapper
44      String expresion = "int prueba(){if (a == b){int hola = 0;}else
            {} return 0;};";
45         LexerCupAnalyzer lexer = new LexerCupAnalyzer(new
               StringReader(expresion));
46       // start parsing
47       Parser p = new Parser(lexer,csf);
48       System.out.println("Parser runs: ");
49       newScope();
50       XMLElement e = (XMLElement)p.parse().value;/*
51       // create XML output file
52       XMLOutputFactory outFactory = XMLOutputFactory.newInstance();
53       XMLStreamWriter sw = outFactory.createXMLStreamWriter(new
             FileOutputStream(args[1]));
54       // dump XML output to the file
55       XMLElement.dump(lexer,sw,e); //,"expr","stmt");
56       // transform the parse tree into an AST and a rendered HTML
             version
57       Transformer transformer = TransformerFactory.newInstance()
58    .newTransformer(new StreamSource(new File("tree.xsl")));
59       Source text = new StreamSource(new File(args[1]));
60       transformer.transform(text, new StreamResult(new File("output.
             html")));
61   */
62       System.out.println("Parsing finished!");
63    } catch (Exception e) {
64       e.printStackTrace();
```

```
65   }
66       }
67
68       public void report_error(String message, Object info) {
69           if (info instanceof String){
70        errors++;
71             System.err.println("  "+ errors + "==> " + info + " "+
                   message +
72            "\n        Parsing resumed from 2nd token before" ); //+ s.
                  current_lexeme()+"\n");
73           }
74           else {
75            StringBuffer m = new StringBuffer("Error ");
76            if (info instanceof java_cup.runtime.Symbol)
77                m.append( "("+info.toString()+")" );
78            m.append(" : "+message);
79            System.err.println(m);
80           }
81         }
82
83         public void report_fatal_error(String message, Object info) {
84           report_error(message, info);
85           throw new RuntimeException("Fatal Syntax Error");
86         }
87   :};
88   /*Terminales*/
89
90   terminal OPERADOR, PALABRA_RESERVADA, LITERAL, IDENTIFICADOR,
91        Auto, Break, Case, Char, Const, Continue, Default, Do, Double,
              Else, Enum, Extern, Float, For, Goto,
92        If, Int, Long, Register, Return, Short, Signed, Sizeof, Static
              , Struct, Switch,
93        Typedef, Union, Unsigned, Void, Volatile, While, Mas, Menos,
              Multiplicacion, Division, Decremento,
94        Incremento, Igual, DobleIgual, MayorIgual, Mayor, MenorIgual,
              Menor, Diferente, OrDoble, AndDoble, Not, Coma,
95        PuntoComa, ParentesisIzq, ParentesisDer, CorcheteIzq,
              CorcheteDer, LlaveIzq, LlaveDer, DosPuntos,
96        Punto, SumaAsignacion, RestaAsignacion,
              MultiplicacionAsignacion, DivisionAsignacion,
              SignoPregunta,
97        And, Circunflejo, Modulo, Or, DesplazamientoDerecha,
              DesplazamientoIzquierda, Tilde, ModuloAsignacion,
98        AndAsignacion, CircunflejoAsignacion, OrAsignacion,
              DesplazamientoIzquierdaAsignacion,
99        DesplazamientoDerechaAsignacion, Flecha, Read, Write;
100
101  /*No Terminales*/
102
103  nonterminal TipoDato, PROGRAMA,
104      DECLARACIONES, Declaracion, Declaracion_Specs,
              Init_Declarador_Lista, Init_Declarador, Declarador,
105       Asignar_Expresion, ExpLogica_OR, ExpLogica_AND, ExpOr, ExpAnd,
              ExpIgualdad, ExpRelacional, ExpAditiva,
106       ExpMultiplicativa, Inicializador, Inicializador_lista,
              ExpUnaria, ExpPostfija, ExpPrimaria, Exp,
```

```
107       Operador_Asignacion , DeclaracionFuncion , Declaracion_Lista ,
              DeclaracionCompuesta , Lista_Statement ,
108       Statement , Statement_Label , ExpStatement , SeleccionStatement ,
              IteracionStatement , JumpStatement ,
109       SeleccionStatementElse , ExpConst , ExpCondicional ,
              Parametros_Lista , seleccionStatementIFElse ,
110       seleccionStatementIF , DeclaracionParametros ,
              Declarador_Abstracto , Lista_Identificador ,
111       WriteStatement , ReadStatement
112
113
114
115       ;
116   /*Presedencia*/
117
118
119   //precedence left Resta;
120   //precedence left Multiplicacion;
121   //precedence left Division;
122
123
124   /*Gram ticas*/
125
126   start with PROGRAMA;
127
128   //DECLARACIONES
129
130   Declaracion ::= Declaracion_Specs PuntoComa
131       | Declaracion_Specs Init_Declarador_Lista PuntoComa;
132
133   TipoDato ::= Char:tipo | Int:tipo | Long:tipo | Short:tipo | Void:
          tipo;
134
135   Declaracion_Specs ::=
136       Const:tipo Declaracion_Specs:ds
137       | Static:tipo Declaracion_Specs:ds
138       | TipoDato:ts;
139
140   Init_Declarador_Lista ::=Init_Declarador:id
141    | Init_Declarador_Lista:idl Coma Init_Declarador:id
142    ;
143
144   Init_Declarador ::=Declarador:d
145    | Declarador:d Igual Inicializador:i;
146
147   Declarador ::=IDENTIFICADOR:identifier
148    | ParentesisIzq Declarador:dd ParentesisDer
149    | Declarador:dd CorcheteIzq ExpConst:ce CorcheteDer
150    | Declarador:dd CorcheteIzq CorcheteDer
151    | Declarador:dd ParentesisIzq Parametros_Lista:pl ParentesisDer
152    | Declarador:dd ParentesisIzq Lista_Identificador:li ParentesisDer
153    | Declarador:dd ParentesisIzq ParentesisDer;
154
155   Parametros_Lista ::= DeclaracionParametros:dp
156                        | Parametros_Lista:pl Coma
                             DeclaracionParametros:dp;
157
```

```
158 DeclaracionParametros ::=Declaracion_Specs:ds Declarador:d
159  | Declaracion_Specs:ds Declarador_Abstracto:ad
160  | Declaracion_Specs:ds
161  ;
162
163 Declarador_Abstracto ::= ParentesisIzq:id Declarador_Abstracto:ad
        ParentesisDer
164  | CorcheteIzq:id CorcheteDer
165  | CorcheteIzq:id ExpConst:ce CorcheteDer
166  | Declarador_Abstracto:dad CorcheteIzq:id CorcheteDer
167  | Declarador_Abstracto:dad CorcheteIzq:id ExpConst:ce CorcheteDer
168  | ParentesisIzq:id ParentesisDer
169  | ParentesisIzq:id Parametros_Lista:ptl ParentesisDer
170  | Declarador_Abstracto:dad ParentesisIzq:id ParentesisDer
171  | Declarador_Abstracto:dad ParentesisIzq:id Parametros_Lista:ptl
         ParentesisDer;
172
173 Lista_Identificador ::=IDENTIFICADOR:id
174  | Lista_Identificador:li Coma IDENTIFICADOR:id;
175
176 Inicializador ::=Asignar_Expresion:ae
177  | LlaveIzq Inicializador_lista:il LlaveDer
178  | LlaveIzq Inicializador_lista:il Coma LlaveDer
179  ;
180
181 Inicializador_lista ::=Inicializador:i
182  | Inicializador_lista:il Coma Inicializador:i;
183
184 Asignar_Expresion ::= ExpCondicional:ec
185       | ExpUnaria:ue Operador_Asignacion:aop Asignar_Expresion:ae
186       ;
187
188 Operador_Asignacion ::=Igual
189  | MultiplicacionAsignacion
190  | DivisionAsignacion
191  | ModuloAsignacion
192  | SumaAsignacion
193  | RestaAsignacion;
194  //| MenorIgual
195  //| MayorIgual;
196
197 ExpCondicional ::= ExpLogica_OR:elo
198     | ExpLogica_OR:elo SignoPregunta Exp:e DosPuntos ExpCondicional
            :ec;
199
200 ExpLogica_OR ::= ExpLogica_AND:ela
201     | ExpLogica_OR:elo OrDoble:op ExpLogica_AND;
202
203 ExpLogica_AND ::= ExpOr:eo
204     | ExpLogica_AND:ela AndDoble:op ExpOr:eo;
205
206 ExpOr ::= ExpAnd:ea
207     | ExpOr:eo Or:op ExpAnd:ea;
208
209 ExpAnd ::= ExpIgualdad:ei
210     | ExpAnd:ea And ExpIgualdad:ei;
211
```

```
212 ExpIgualdad ::= ExpRelacional:er
213     | ExpIgualdad:ei DobleIgual ExpRelacional:er
214     | ExpIgualdad:ei Diferente ExpRelacional:er;
215
216 ExpRelacional ::= ExpAditiva:ea
217         | ExpRelacional:re Menor:op ExpAditiva:ea
218         | ExpRelacional:re Mayor:op ExpAditiva:ea
219         | ExpRelacional:re MenorIgual:op ExpAditiva:ea
220         | ExpRelacional:re MayorIgual:op ExpAditiva:ea;
221
222 ExpAditiva ::= ExpMultiplicativa:em
223     | ExpAditiva:ea Mas:op ExpMultiplicativa:em
224     | ExpAditiva:ea Menos:op ExpMultiplicativa:em;
225
226 ExpMultiplicativa ::= ExpUnaria:ec
227      | ExpMultiplicativa:me Multiplicacion:op ExpUnaria:eu
228      | ExpMultiplicativa:me Division:op ExpUnaria:eu
229      | ExpMultiplicativa:me Modulo:op ExpUnaria:eu
230      ;
231
232 ExpUnaria ::= ExpPostfija:ep
233     | Incremento:op ExpUnaria:eu
234     | Decremento:op ExpUnaria:eu
235     ;
236
237 ExpPostfija ::= ExpPrimaria:ep
238    | ExpPostfija:ep CorcheteIzq Exp:index CorcheteDer
239    | ExpPostfija:ep ParentesisIzq ParentesisDer
240    | ExpPostfija:ep ParentesisIzq Exp:e ParentesisDer
241    | ExpPostfija:ep Punto IDENTIFICADOR:id
242    | ExpPostfija:ep Incremento:op
243    | ExpPostfija:ep Decremento:op
244    ;
245
246 Exp ::= Asignar_Expresion:ae
247     | Exp:e Coma Asignar_Expresion:ae;
248
249 ExpConst ::= ExpCondicional:ec;
250
251 ExpPrimaria ::= IDENTIFICADOR:ident
252     | Const:constant
253     | LITERAL:literal
254     | ParentesisIzq Exp ParentesisDer;
255
256 PROGRAMA ::= DECLARACIONES
257   | PROGRAMA DECLARACIONES;
258
259 DECLARACIONES ::= Declaracion
260     |    DeclaracionFuncion;
261
262 Declaracion_Lista ::= Declaracion:d
263     | Declaracion_Lista:dl Declaracion;
264
265 DeclaracionFuncion ::= Declaracion_Specs:ds Declarador:d
        Declaracion_Lista:dl {: Parser.newScope(); :}
        DeclaracionCompuesta:dc {: Parser.deleteScope(); :}
```

```
266       | Declaracion_Specs:ds Declarador:d {: Parser.newScope(); :}
              DeclaracionCompuesta:dc {: Parser.deleteScope(); :}
267       | Declarador:d Declaracion_Lista:dl {: Parser.newScope(); :}
              DeclaracionCompuesta:dc {: Parser.deleteScope(); :}
268       | Declarador:d {: Parser.newScope(); :} DeclaracionCompuesta:dc
              {: Parser.deleteScope(); :}
269       | DeclaracionCompuesta:d
270       ;
271
272
273   Lista_Statement ::=Statement:s
274    | Lista_Statement:ls Statement:s;
275
276
277   DeclaracionCompuesta ::=LlaveIzq LlaveDer
278       | LlaveIzq Lista_Statement:ls LlaveDer
279    | LlaveIzq Declaracion_Lista:dl LlaveDer
280    | LlaveIzq Declaracion_Lista:dl Lista_Statement:ls LlaveDer;
281
282   ExpStatement ::= PuntoComa
283       | Exp:e PuntoComa;
284
285   Statement_Label ::= IDENTIFICADOR:id DosPuntos Statement:s
286       | Case ExpConst:ec DosPuntos Statement:s
287       | Default DosPuntos Statement:s;
288
289   // ----------Problema del if else
290   SeleccionStatement ::= seleccionStatementIF
291       | Switch ParentesisIzq Exp:e ParentesisDer Statement:s;
292   seleccionStatementIF ::=If ParentesisIzq Exp:e ParentesisDer
          Statement:s1 seleccionStatementIFElse; //SeleccionStatementElse
293   seleccionStatementIFElse ::= Else Statement:s2;
294
295
296
297   // ::= Else Statement:s2 | DECLARACIONES ;// TENER CUIDADO
298
299   IteracionStatement ::= While ParentesisIzq Exp:e ParentesisDer
          Statement:s
300       | Do Statement:s While ParentesisIzq Exp:e ParentesisDer
              PuntoComa
301       | For ParentesisIzq ExpStatement:es1 ExpStatement:es2
              ParentesisDer Statement:s
302       | For ParentesisIzq ExpStatement:es1 ExpStatement:es2 Exp:e
              ParentesisDer Statement:s;
303
304
305   Statement ::=
306       Statement_Label:sl
307       | {: Parser.newScope(); :}  DeclaracionCompuesta:cs {: Parser.
              deleteScope(); :}
308    | ExpStatement:es
309    | SeleccionStatement:ss
310    | IteracionStatement:is
311    | JumpStatement:js
312    | ReadStatement:rs
313    | WriteStatement:ws
```

```
314   ;
315
316  JumpStatement ::= Continue PuntoComa
317       | Break PuntoComa
318       | Return PuntoComa
319       | Return Exp:e PuntoComa;
320
321  ReadStatement ::= Read ParentesisIzq ExpStatement:es ParentesisDer;
322
323  WriteStatement ::= Write ParentesisIzq Exp:e ParentesisDer;
```

Capítulo 3

Manual de Usuario

Este manual está planeado para alguien que no sabe cómo fue que se programó la solución y ya tiene una versión completa y estable del producto. Al abrir el programa, inicialmente verá una interfaz gráfica sencilla. En esta verá un espacio en blanco grande a la izquierda de la ventana. En este espacio puede escribir lo que desee (preferiblemente que dicho texto tenga tokens y gramática de C pues es un compilador de C). Debajo del espacio para el texto, encontrará los botones Reestablecer y Cargar Archivo. Al lado derecho de la pantalla verá dos espacios. Estos espacios son para el análisis sintáctico y léxico respectivamente. En cada uno de estos espacios vendrán los errores encontrados para cada caso. También los espacios de Errores Léxicos y Errores Sintácticos tienen abajo los botones de tokenizar y parsear respectivamente. En la parte inferior derecha de la pantalla encontrará dos botones, Compilar y Salir.

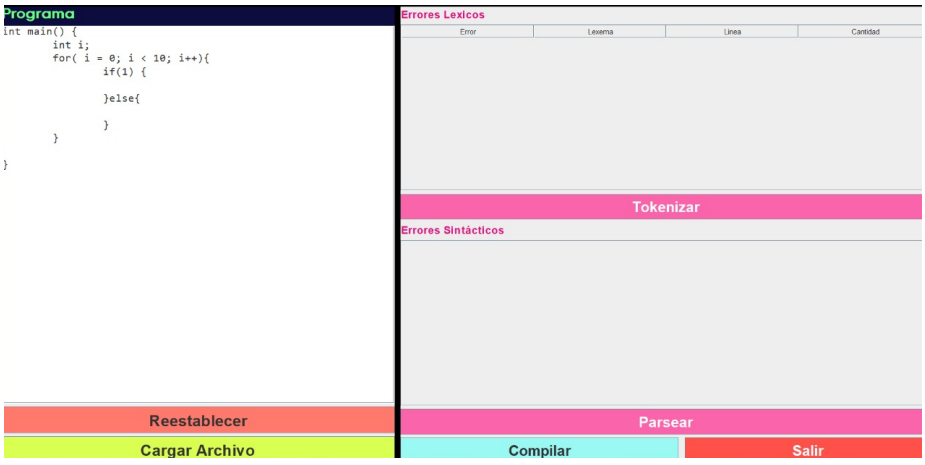

Figura 3.1: Ejemplo de uso del programa

Para poner un texto para que este sea analizado hay dos opciones. La pri-

mera opción es escribir manualmente sobre el espacio en blanco a la izquierda seleccionando el área de texto.

La segunda opción es cargar un archivo. Para ello debe dar click en el botón Çargar Archivo". Esto le abrirá el explorador de archivos de su computadora y puede seleccionar algún documento de texto o extensión .c que tenga guardado.

En caso de cargar un archivo, una vez que lo seleccione, todo el contenido de este se pondrá en el espacio para texto de la izquierda. Note que, en ambos casos, escritura manual o con archivo, siempre va a quedar texto en el espacio destinado para este propósito. Si es mucho texto y no cabe en el área de escritura, puede navegar con el scrollbar. Una vez que ya haya texto, se puede proceder al proceso de compilación. Para ello debe presionar el botón Compilar.

Una vez que presione este botón aparecerán en el espacio de Errores Léxicos todos los errores de este tipo que se hayan encontrado. Acá aparecerá cuál es el error, la línea donde aparece y la cantidad total de apariciones. Al igual que con los tokens, si un mismo error aparece en varias líneas se pone la lista de números de líneas y si el error aparece más de una vez en la misma línea se pone el número de línea y entre paréntesis la cantidad de apariciones en esta.

Al compilar, también se presenta un resultado muy similar para el parseo. En el espacio inferior, se muestran todos los errores sintácticos encontrados. Se da información que ayude al usuario que estuvo mal y donde. Se indica el error, la cantidad de apariciones de dicho error y las líneas donde se encuentran. Además de que generó el error, que fue lo que no se aceptó.

Si nada más me interesan los errores léxicos, puedo usar el botón tokenizar para que actualice solo el campo correspondiente.

De igual manera, si solo me interesan los errores sintácticos pulso el botón Parsear.

Para limpiar la interfaz gráfica, se usa el botón reestablecer. Este botón elimina el texto del área de escritura y también elimina los errores léxicos y sintácticos ya encontrados. Deja la interfaz gráfica como en su estado original.

En resumen, al dar çlick.[en] el botón Compilar se ejecuta la revisión léxica del texto que se tenga en ese momento. Esto va a desplegar todos los tokens y errores encontrados con su información respectiva. En caso de que no haya texto y se aprete Compilar, no va a pasar nada.

También se tiene el botón Borrar. Este botón resetea la sección de Aceptados y Errores.

Capítulo 4

Bitácora

4.1. Actividades

A continuación se muestra una tabla con las actividades realizadas durante el desarrollo de este proyecto, las fechas corresponden a los días que se hizo reunión; no obstante, se excluye el tiempo utilizados para la búsqueda de información y referencias.

Actividades		
Fecha	Actividad	Detalles relevantes
22/10/2022	Corrección de Errores Menores del Analizador Léxico	Del proyecto anterior habían unos errores con los identificadores que ya se corrigieron
24/10/2022	Reunión Inicial	Entender el propósito y distribuir tareas
25/10/2022	Inicio de trabajo en gramáticas	Creación de gramáticas de C
25/10/2002	Investigación de cómo usar JCup	Aprender como usar la herramienta de parseo
28/10/2022	Adaptar GUI a este proyecto	El proyecto anterior tenía menos cosas entonces había que ajustar la GUI
04/10/2022	Implementación de gramáticas en JCup	Las gramáticas hechas en borrador se implentaron ya en el programa
05/11/2022	Corrección de errores	Se analizaron y corrigeron todos los errores de shift-reduce y reduce-reduce
07/11/2022	Pruebas	Las pruebas demostraron algunos errores. Afortunadamente eran menores entonces se corrigeron rápido.
07/11/2022	Corrección de errores	Se hicieron las pruebas para asegurarse que el análisis sintáctico fuera correcto y robusto.
07/11/2022	Documentación	Parte escrita del proyecto que complementa al programa

Capítulo 5

Conclusión

5.1. Análisis de resultados

- Utilización de JCup para análisis sintáctico: 100 %
  Se utilizó la herramienta JCup en el lenguaje de programación Java por completo.
- Definición de tokens: 100 %

  Se definieron los terminales del lenguaje de programación C.
- Definición de no terminales: 100 %

  Identifica correctamente los producciones de la gramática.
- Eliminación de errores de gramática: 100 %

  Se quitaron los errores shift-reduce.
- Definición de gramática libre de contexto C: 100 %

  Las gramáticas que conforman el archivo "parser.cup"funcionan de forma correcta.
- Leer texto: 100 %

  El programa permite cargar texto desde un archivo .txt o .c, asímismo hacer pruebas con tokens ingresados por el usuario directamente en la interfaz gráfica.
- Identificar errores: 80 %

La mayoría de veces se capturan los errores recuperando la ejecución del programa, aunque a veces no lo logra. No obstante siempre funciona bien cuando hay un error en la gramática de C, capturando dicho error en específico. A pesar de esto, en estos casos especiales se detiene la ejecución.

- Documentación: 100 %
  El documento tiene todas las especificaciones solicitadas en el enunciado del proyecto.

5.2. Lecciones Aprendidas

Este proyecto aportó beneficios a la formación como ingenieros de todos los miembros del equipo. Hubo aprendizaje técnico donde se profundizó lo estudiado en el aula. Se complementó la teoría dada por la profesora con fuentes en Internet y además se puso en práctica todo este conocimiento, lo que ayudó a reforzar aún más. También se trabajaron habilidades blandas tanto intrapersonales como interpersonales. Se mejoró el trabajo en equipo y la organización de cada miembro.

Con respecto a las metodologías de trabajo, se probó aplicar la delegación. Esto significa que a cada miembro se le asignaba un avance y este debió compartirlo con el resto del grupo apenas lo terminara. Es importante que esto se aplico solo para tareas más sencillas como elementos de la documentación. Para tareas más complejas sí se dio un trabajo entre los tres miembros donde todos aportaban ideas y se ayudaban mutuamente. Por ejemplo, en la creación de gramáticas o la corrección de errores shift-reduce y reduce-reduce en el Cup.

Para este proyecto se dio un uso intensivo de medios de comunicación digitales como WhatsApp o Discord. Debido a los horarios y actividades personales de cada miembro, fue muy difícil que todos estuvieran disponibles en el mismo momento y en el mismo lugar. La ventaja es que se fortaleció el uso de estas herramientas que cada vez son más comunes en el ámbito profesional, como el sistema de control de versiones GitHub y el uso de las brances. Adicionalmente, siempre se procuró una participación de todos los miembros y la colaboración a pesar de ser medios virtuales. Afortunadamente no se tuvieron problemas a lo interno del grupo. Solo se fortalecieron habilidades como la cooperación, trabajo en equipo y organización de tareas. Estas son características muy importantes para satisfacer el mercado laboral.

Por otro lado, este proyecto fue especialmente beneficioso para reforzar los conceptos vistos en clase. todos los miembros pudimos aplicar los conceptos vistos en clase. El hecho de poder aplicar estos conceptos de manera práctica permitió ver cómo se integran entre sí y qué aplicación tienen en un caso real. Adicionalmente, se pudo practicar intensivamente el tema de gramáticas libres de contexto. Esto mejoró el entendimiento de conceptos como terminales y producciones y en especial se entendió mejor cómo funcionan los errores shift-reduce o reduce-reduce. Adicionalmente, esto reforzó el entendimiento de como esto no se queda solo en la teoría, sino que es funcional en compiladores. También el

proyecto sirvió para repasar cómo las expresiones regulares describen lenguajes regulares.

En el caso del aprendizaje con el uso de la herramienta Cup, lo cual fue una herramienta muy útil todos los miembros de nuestro equipo. Aprendimos que la herramienta nos facilita el trabajo de generar un programa compilador basado en los tokens que ya tenemos y solo nos centramos en definir la gramática para obtener el análisis sintáctico. Nos pareció bastante ingenioso lo bien que se acopla esta herramienta con Cup. Esto no solo facilita el trabajo, sino que ayuda a unir conceptualmente los procesos de análisis léxico con el análisis sintáctico. En términos generales la comunicación y trabajo en equipo siempre fue presente ya que nos apoyamos de buenas herramientas que impulsaban esta práctica como el "live share"del entorno de desarrollo de Intellij lo cual nos facilitó el trabajo y decidimos implementar esta herramienta para los próximos proyectos programados. La herramienta "live-share" permitió que todos entendiéramos que estaban haciendo los demás y tener mejor estructura como equipo.

5.3. Link del repositorio en Github

`https://github.com/geralm/Compilador-del-lenguaje-C.git`

Referencias

CUP. (n.d.). http://www2.cs.tum.edu/projects/cup/examples.php

Shift/reduce conflict in java cup - dangling else issue. (2015, December 13). Stack Overflow. https://stackoverflow.com/questions/34254103/shift-reduce-conflict-in-java-cup-dangling-else-issue

Integración de JFlex y CUP (analizadores léxico y sintáctico) - PDF Free Download. (n.d.). https://docplayer.es/25036146-Integracion-de-jflex-y-cup-analizadores-lexico-y-sintactico.html

Sintax.cup - 321c64c4. (n.d.). https://controlc.com/321c64c4

Víctor Orozco. (2022, June 20). Creación de un analizador sintáctico con JFlex, CUP, Maven y Java 17. YouTube. https://www.youtube.com/watch?v=4XKelO44u5U