



**Escuela de Ingeniería en Computación**

**IC-5701 Compiladores e intérpretes**

**Grupo 01**

**Profesor Ignacio Trejos Zelaya**

**Proyecto 01 - Fase 01**

**Análisis léxico y Sintáctico**

**Estudiantes**

**David Castro Holguin - 2018105813**

**Keila Álvarez Bermúdez - 2018227606**

**Anner Calvo Granados - 2017107433**

**Fecha**

**05/10/2021**

## Tabla de contenidos

<b>Manejo del texto fuente</b>	<b>3</b>
<b>Modificaciones al compilador</b>	<b>3</b>
<b>GUI</b>	<b>3</b>
<b>Core</b>	<b>3</b>
<b>Triangle</b>	<b>10</b>
<b>Syntactic Analyzer</b>	<b>10</b>
<b>Abstract Syntax Trees</b>	<b>19</b>
Modelos	26
Tree Drawer	27
<b>HTMLWriter</b>	<b>30</b>
<b>Lógica utilizada</b>	<b>30</b>
XMLWriter	32
<b>Lógica utilizada</b>	<b>32</b>
<b>Nuevos errores sintácticos</b>	<b>34</b>
<b>Plan de pruebas</b>	<b>34</b>
<b>Discusión y análisis de los resultados obtenidos</b>	<b>45</b>
<b>Una reflexión sobre la experiencia de modificar fragmentos de un compilador</b>	<b>46</b>
<b>Instrucciones de Compilación</b>	<b>46</b>
<b>Instrucciones de ejecución</b>	<b>46</b>
<b>Descripción de las tareas realizadas</b>	<b>47</b>
<b>Anexos</b>	<b>48</b>

## Manejo del texto fuente

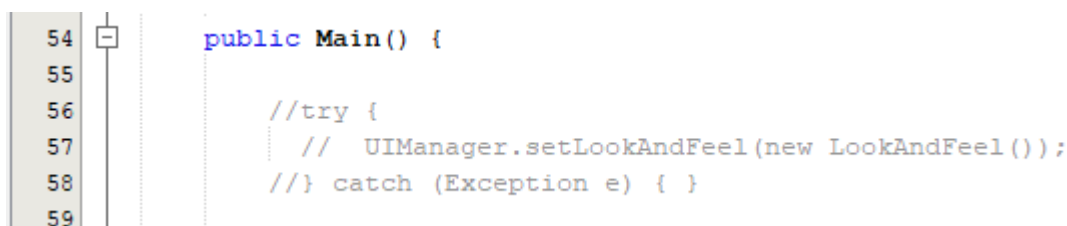
En el esquema del manejo del texto fuente como la lectura de archivos de entrada o de estructura de datos del IDE no fueron modificados.

## Modificaciones al compilador

### GUI

#### Main.java

A la hora de compilar el proyecto generaba un error al tratar de usar la librería Look and Feel, luego de investigar esta es una librería antigua y que no era necesaria para el funcionamiento del compilador, así que se comentaron las líneas que la invocaban.



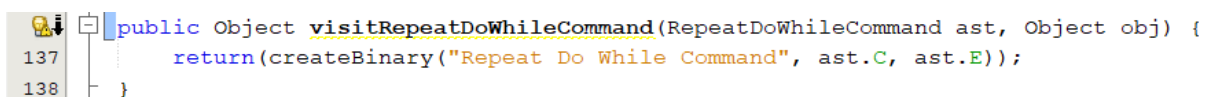
```
54 public Main() {
55
56     //try {
57         // UIManager.setLookAndFeel(new LookAndFeel());
58     //} catch (Exception e) { }
59 }
```

### Core

#### TreeVisitor.java

Para cada AST nuevo se creó un método con nombre visit y el nombre del AST, por ejemplo, para el AST *RepeatDoWhileCommand* se creó un método llamado *visitRepeatDoWhileCommand* y así con todos los AST nuevos. Estos métodos reciben dos parámetros de entrada, el primero es un ast del tipo respectivo de cada método, siguiendo con el ejemplo anterior el método *visitRepeatDoWhileCommand* recibiría como primer parámetro un objeto de tipo *RepeatDoWhileCommand* y como segundo parámetro un objeto de tipo *Object*, y esto se repite para todos los demás métodos.

Estos métodos crean y retornan un AST de tipo binario, ternario o cuaternario según corresponda para cada AST. Tal como el AST *RepeatDoWhileCommand* que está compuesto únicamente por un comando y una expresión entonces retornaría un AST de tipo binario, tal como se muestra en la siguiente imagen.



```
137 public Object visitRepeatDoWhileCommand(RepeatDoWhileCommand ast, Object obj) {
138     return(createBinary("Repeat Do While Command", ast.C, ast.E));
139 }
```

Y lo mismo se repetiría en los métodos de los AST de tipo comando como RepeatWhileDoCommand, RepeatUntilDoCommand, RepeatDoUntilCommand, RepeatInCommand, RepeatForRangeCommand, RepeatForRangeWhileCommand, RepeatForRangeUntilCommand y la declaración ForRangeIdentifierExpression, tal como se muestra a continuación:

### RepeatWhileDoCommand

```
129 public Object visitRepeatWhileDo(RepeatWhileDoCommand ast, Object obj) {  
130     return(createBinary("Repeat While Do Command", ast.E, ast.C));  
    }
```

### RepeatUntilDoCommand

```
133 public Object visitRepeatUntilDo(RepeatUntilDoCommand ast, Object obj) {  
134     return(createBinary("Repeat Until Do Command", ast.E, ast.C));  
    }
```

### RepeatDoUntilCommand

```
141 public Object visitRepeatDoUntilCommand(RepeatDoUntilCommand ast, Object obj) {  
142     return(createBinary("Repeat Do Until Command", ast.C, ast.E));  
    }
```

### RepeatInCommand

```
145 public Object visitRepeatInCommand(RepeatInCommand ast, Object o) {  
146     return(createBinary("Repeat In Command", ast.D, ast.C));  
    }
```

### RepeatForRangeCommand

```
149 public Object visitRepeatForRangeCommand(RepeatForRangeCommand ast, Object o) {  
150     return(createTernary("Repeat For Range Command", ast.D, ast.E, ast.C));  
    }
```

### RepeatForRangeWhileCommand

```
153 public Object visitRepeatForRangeWhileCommand(RepeatForRangeWhileCommand ast, Object o) {  
154     return(createQuaternary("Repeat For Range While Command", ast.D, ast.E1, ast.E2, ast.C));  
    }
```

### RepeatForRangeUntilCommand

```
157 public Object visitRepeatForRangeUntilCommand(RepeatForRangeUntilCommand ast, Object o) {  
158     return(createQuaternary("Repeat For Range Until Command", ast.D, ast.E1, ast.E2, ast.C));  
    }
```

## ForRangeIdentifierExpression

```
246 public Object visitForRangeIdentifierExpression(ForRangeIdentifierExpression ast, Object obj) {
247     return(createBinary("For Range Identifier Expression Declaration", ast.I, ast.E));
}
```

De igual forma, esta estructura se repite en las reglas, para las cuales se crearon árboles como RecursiveDeclaration, LocalDeclaration, ProcDeclaration, FuncDeclaration, parseProcFunc, ProcFuncSDeclaration y VarExpressionDeclaration, los cuales se muestran a continuación:

## RecursiveDeclaration

```
276 public Object visitRecursiveDeclaration(RecursiveDeclaration ast, Object o) {
277     return(createUnary("RecursiveDeclaration", ast.D1)); //Se agrego el metodo
}
```

## LocalDeclaration

```
260 public Object visitLocalDeclaration(LocalDeclaration ast, Object o) {
261     return(createBinary("LocalDeclaration", ast.D1, ast.D2)); //Se agrego el metodo
}
```

## ProcDeclaration

```
236 public Object visitProcDeclaration(ProcDeclaration ast, Object obj) {
237     return(createTernary("Procedure Declaration", ast.I, ast.FPS, ast.C));
}
```

## FuncDeclaration

```
232 public Object visitFuncDeclaration(FuncDeclaration ast, Object obj) {
233     return(createQuaternary("Function Declaration", ast.I, ast.FPS, ast.T, ast.E));
}
```

## ProcFuncSDeclaration

```
272 public Object visitProcFuncSDeclaration(ProcFuncSDeclaration ast, Object o) {
273     return(createBinary("ProcFuncSDeclaration", ast.D1, ast.D2)); //Se agrego el metodo
}
```

## VarExpressionDeclaration

```
264 public Object visitVarExpressionDeclaration(VarExpressionDeclaration ast, Object o) {
265     return(createBinary("VarExpressionDeclaration", ast.E, ast.I)); //Se agrego el metodo
}
```

Y lo mismo se repetiría en los métodos de los AST pero en este caso para la funcionalidad referente al SELECT, tal como se muestra a continuación:

## SelectCommand

```
276 public Object visitSelectCommand(selectCommand ast, Object o) {
277     return(createBinary("selectCommand", ast.D, ast.E)); //Se agrego el metodo
}
```

## RangeDeclaration

```
280 public Object visitRangeDeclaration(RangeDeclaration ast, Object o) {
281     return(createBinary("RangeDeclaration", ast.D1, ast.D2)); //Se agrego el metodo
}
```

## CaseLiteralDeclaration

```
284 public Object visitCaseLiteralDeclaration(CaseLiteralDeclaration ast, Object o) {
285     return(createUnary("CaseLiteralDeclaration", ast.D1)); //Se agrego el metodo
}
```

## ElseCaseCommand

```
288 public Object visitElseCaseCommand(ElseCaseCommand ast, Object o) {
289     return(createUnary("ElseCaseCommand", ast.C)); //Se agrego el metodo
}
```

## LiteralDeclaration

```
292 public Object visitLiteralDeclaration(LiteralDeclaration ast, Object o) {
293     return(createUnary("LiteralDeclaration", ast.LL)); //Se agrego el metodo
}
```

## CharDeclaration

```
296 public Object visitCharDeclaration(CharDeclaration ast, Object o) {
297     return(createUnary("CharDeclaration", ast.CH)); //Se agrego el metodo
}
```

## TableVisitor.java

Para cada nuevo AST se creó un método de nombre visit más el nombre del AST, es decir en el caso de AST *RepeatDoWhileCommand* el método sería *visitRepeatDoWhileCommand* este método recibe dos parámetros, el primero es del tipo del AST y el segundo es de tipo Object. El método por medio del ast que entra como parámetro se llama el método *visit* creado en cada clase de los AST. Los métodos creados se muestran a continuación en las siguientes imágenes:

## RepeatWhileDoCommand

```
154 public Object visitRepeatWhileDo(RepeatWhileDoCommand ast, Object o) {
155     ast.E.visit(this, null);
156     ast.C.visit(this, null);
157     return(null);
158 }
```

## RepeatUntilDoCommand

```
161 public Object visitRepeatUntilDo(RepeatUntilDoCommand ast, Object o) {  
162     ast.E.visit(this, null);  
163     ast.C.visit(this, null);  
164     return(null);  
165 }
```

## RepeatDoWhileCommand

```
168 public Object visitRepeatDoWhileCommand(RepeatDoWhileCommand ast, Object o) {  
169     ast.C.visit(this, null);  
170     ast.E.visit(this, null);  
171     return(null);  
172 }
```

## RepeatDoUntilCommand

```
175 public Object visitRepeatDoUntilCommand(RepeatDoUntilCommand ast, Object o) {  
176     ast.C.visit(this, null);  
177     ast.E.visit(this, null);  
178     return(null);  
179 }
```

## RepeatInCommand

```
182 public Object visitRepeatInCommand(RepeatInCommand ast, Object o) {  
183     ast.D.visit(this, null);  
184     ast.C.visit(this, null);  
185     return(null);  
186 }
```

## RepeatForRangeCommand

```
189 public Object visitRepeatForRangeCommand(RepeatForRangeCommand ast, Object o) {  
190     ast.D.visit(this, null);  
191     ast.E.visit(this, null);  
192     ast.C.visit(this, null);  
193     return(null);  
194 }
```

## RepeatForRangeWhileCommand

```
197 public Object visitRepeatForRangeWhileCommand(RepeatForRangeWhileCommand ast, Object o) {
198     ast.D.visit(this, null);
199     ast.E1.visit(this, null);
200     ast.E2.visit(this, null);
201     ast.C.visit(this, null);
202     return(null);
203 }
```

## RepeatForRangeUntilCommand

```
206 public Object visitRepeatForRangeUntilCommand(RepeatForRangeUntilCommand ast, Object o) {
207     ast.D.visit(this, null);
208     ast.E1.visit(this, null);
209     ast.E2.visit(this, null);
210     ast.C.visit(this, null);
211     return(null);
212 }
```

## ForRangeIdentifierExpression

```
388 public Object visitForRangeIdentifierExpression(ForRangeIdentifierExpression ast, Object o) {
389     ast.I.visit(this, null);
390     ast.E.visit(this, null);
391     return(null);
392 }
```

## RecursiveDeclaration

```
429 public Object visitRecursiveDeclaration(RecursiveDeclaration ast, Object o) { //Se agrego el metodo vi
430     ast.D1.visit(this, null);
431     return(null);
432 }
433 }
```

## LocalDeclaration

```
405 public Object visitLocalDeclaration(LocalDeclaration ast, Object o) { //Se agrego el metodo visitLocalD
406     ast.D1.visit(this, null);
407     ast.D2.visit(this, null);
408     return(null);
409 }
410 }
```

## ProcFuncSDeclaration

```
421 public Object visitProcFuncSDeclaration(ProcFuncSDeclaration ast, Object o) { //Se agrego el metodo visitI
422     ast.D1.visit(this, null);
423     ast.D2.visit(this, null);
424     return(null);
425 }
426 }
```



## VarExpressionDeclaration

```
413 public Object visitVarExpressionDeclaration(VarExpressionDeclaration ast, Object o) { //Se agrego el met
414     ast.I.visit(this, null);
415     ast.E.visit(this, null);
416     return(null);
417 }
418 }
```

## SelectCommand

```
443 public Object visitSelectCommand(selectCommand ast, Object o) { //Se agrego el metodo visitSelectCommand()
444     ast.D.visit(this, null);
445     ast.E.visit(this, null);
446     return(null);
447 }
448 }
```

## RangeDeclaration

```
451 public Object visitRangeDeclaration(RangeDeclaration ast, Object o) { //Se agrego el metodo visitRangeDeclara
452     ast.D1.visit(this, null);
453     ast.D2.visit(this, null);
454     return(null);
455 }
456 }
```

## CaseLiteralDeclaration

```
459 public Object visitCaseLiteralDeclaration(CaseLiteralDeclaration ast, Object o) { //Se agrego el metodo visitC
460     ast.D1.visit(this, null);
461     return(null);
462 }
463 }
```

## ElseCaseCommand

```
436 public Object visitElseCaseCommand(ElseCaseCommand ast, Object o) { //Se agrego el metodo visitElseCaseComman
437     ast.C.visit(this, null);
438     return(null);
439 }
440 }
```

## LiteralDeclaration

```
459 public Object visitCaseLiteralDeclaration(CaseLiteralDeclaration ast, Object o) { //Se agrego el metodo visit
460     ast.D1.visit(this, null);
461     return(null);
462 }
463 }
```

## CharDeclaration

```
473 public Object visitCharDeclaration(CharDeclaration ast, Object o) { //Se agrego el metodo visitCharDeclaratio
474     ast.CH.visit(this, null);
475     return(null);
476 }
477 }
```

# Triangle

## IDECompiler.java

Se hace la llamada a los métodos encargados de generar los archivos HTML y XML, según el resultado que brinde el análisis del código compilado.

```
60
61 // nuevo: Llamada al generador HTML
62 String filesDestination = sourceName.substring(0, sourceName.length() - 4);
63 HTML writerHTML = new HTML(sourceName);
64 writerHTML.generateHTML(filesDestination);
```

HTML

```
82 if (success)
83 {
84     System.out.println("Compilation was successful.");
85     // nuevo: Llamada al XML
86     XMLWriterTree writerXML = new XMLWriterTree(sourceName);
87     writerXML.writer(rootAST);
88 }
```

XML

## Syntactic Analyzer

### Parser.java

El parser es donde se implementan todas las modificaciones y se integran los nuevos comandos y reglas. En cuanto a los comandos, se realizaron modificaciones a los comandos IF y LET.

```
323 case Token.IF:
324 {
325     acceptIt();
326     Expression eAST = parseExpression();
327     accept(Token.THEN);
328     Command c1AST = parseCommand();
329     Expression eAST2 = null;
330     Command c2AST = null;
331     Command c3AST = null;
332     while (currentToken.kind == Token.PIPE) // Modificacion del pipe(elif)
333     {
334         acceptIt();
335         eAST2 = parseExpression();
336         accept(Token.THEN);
337         c2AST = parseCommand();
338         c3AST = c2AST;
339     }
340
341     accept(Token.ELSE);
342     c3AST = parseCommand();
343     accept(Token.END);
344     finish(commandPos);
345     commandAST = new IfCommand(eAST, c1AST, c3AST, commandPos);
346 }
347 break;
```

Comando IF

```

311     case Token.LET: // Comando modificado
312     {
313         acceptIt();
314         Declaration dAST = parseDeclaration();
315         accept(Token.IN);
316         Command cAST = parseCommand(); // cambio
317         accept(Token.END);
318         finish(commandPos);
319         commandAST = new LetCommand(dAST, cAST, commandPos);
320     }
321     break;

```

*Comando LET*

En cuanto a los comandos nuevos se añadieron los siguientes: *SELECT*, *SKIP*, *REPEAT*, *RECURSIVE*, *LOCAL*

```

349     case Token.SELECT: // Comando nuevo
350     {
351         acceptIt();
352         Expression eAST = parseExpression();
353         accept(Token.FROM);
354         Declaration casesAST = parseCases();
355         accept(Token.END);
356         finish(commandPos);
357         commandAST = new selectCommand(casesAST, eAST, commandPos);
358     }
359     break;
360

```

*Comando SELECT*

```

372     case Token.SKIP: // Comando nuevo
373     {
374         acceptIt();
375         finish(commandPos);
376         commandAST = new EmptyCommand(commandPos);
377         break;
378     }

```

*Comando SKIP*

```

843     case Token.RECURSIVE:
844     {
845         acceptIt();
846         Declaration dAST = parseProcFuncS();
847         accept (Token.END);
848         finish(declarationPos);
849         declarationAST = new RecursiveDeclaration(dAST, declarationPos);
850     }
851     break;

```

*Comando RECURSIVE*

```

853     case Token.LOCAL:
854     {
855         acceptIt();
856         Declaration dlAST = parseDeclaration();
857         accept(Token.IN);
858         Declaration d2AST = parseDeclaration();
859         accept(Token.END);
860         finish(declarationPos);
861         declarationAST = new LocalDeclaration(dlAST, d2AST, declarationPos);
862     }
863     break;

```

Comando LOCAL

Para el caso del comando *REPEAT*, se hizo una implementación más compleja en la que se deben tomar en cuenta todas las formas en las que se puede presentar el comando. Para esto, se desarrollaron 4 métodos parser según el tipo de repeat los cuales son: *parseRepeatCases*, *parseRepeatDo*, *parseRepeatRangeIn*, *parseRepeatRange*.

```

397 Command parseRepeatCases() throws SyntaxError {
398     Command commandAST = null; // in case there's a syntactic error
399
400     SourcePosition commandPos = new SourcePosition();
401     start(commandPos);
402
403     switch (currentToken.kind) {
404     case Token.WHILE:
405     {
406         acceptIt();
407         Expression eAST = parseExpression();
408         accept(Token.DO);
409         Command cAST = parseCommand();
410         accept(Token.END);
411         finish(commandPos);
412         commandAST = new RepeatWhileDoCommand(eAST, cAST, commandPos); //
413     }
414     break;
415     case Token.UNTIL:

```

*parseRepeatCases*

```

453 Command parseRepeatDo(Command cAST) throws SyntaxError{
454     Command commandAST = null; // in case there's a syntactic error
455
456     SourcePosition commandPos = new SourcePosition();
457     start(commandPos);
458
459     switch (currentToken.kind) {
460     case Token.WHILE:
461     {
462         acceptIt();
463         Expression eAST = parseExpression();
464         accept(Token.END);
465         finish(commandPos);
466         commandAST = new RepeatDoWhileCommand(cAST, eAST, commandPos); //
467     }
468     break;

```

*parseRepeatDo*

```

490 Command parseRepeatRangeIn(Identifier iAST) throws SyntaxError{
491     Command commandAST = null; // in case there's a syntactic error
492
493     SourcePosition commandPos = new SourcePosition();
494     start(commandPos);
495
496     switch (currentToken.kind) {
497         case Token.BECOMES: //:=
498         {
499             acceptIt();
500             accept(Token.RANGE);
501             Expression elAST = parseExpression();
502             accept(Token.DOUBLEDOT);
503             Expression e2AST = parseExpression();
504             commandAST = parseRepeatRange(iAST, elAST, e2AST);
505
506         }
507         break;

```

*parseRepeatIn*

```

533 Command parseRepeatRange(Identifier iAST, Expression elAST, Expression e2AST) throws SyntaxError{
534     Command commandAST = null; // in case there's a syntactic error
535
536     SourcePosition commandPos = new SourcePosition();
537     start(commandPos);
538
539     switch (currentToken.kind) {
540         case Token.DO:
541         {
542             acceptIt();
543             Command cAST = parseCommand();
544             accept(Token.END);
545             finish(commandPos);
546             Declaration RangeVarDecl = new ForRangeIdentifierExpression(iAST, elAST, commandPos);
547             commandAST = new RepeatForRangeDoCommand(RangeVarDecl, e2AST, cAST, commandPos);
548         }
549         break;

```

*parseRepeatRange*

En el parser también se implementan nuevas reglas, las cuales trabajan en combinación con componentes que estaban anteriormente definidos, sin embargo, para asegurar un correcto funcionamiento a algunos se les tuvo que hacer unos cambios por ejemplo:

## Declaration

Esta implementación anteriormente hacía uso del método singleDeclaration, esta funcionalidad se cambió para que hiciera uso del método Compound-Declaration, estructuralmente este método sigue la misma línea que el anterior, pero al llamar otro método cambia la funcionalidad de la ejecución y nos presenta nuevas oportunidades de uso.

```

825 Declaration parseDeclaration() throws SyntaxError {
826     Declaration declarationAST = null; // in case there's a syntactic error
827
828     SourcePosition declarationPos = new SourcePosition();
829     start(declarationPos);
830     declarationAST = parseCompoundDeclaration();
831     while (currentToken.kind == Token.SEMICOLON) {
832         acceptIt();
833         Declaration d2AST = parseCompoundDeclaration();
834         finish(declarationPos);
835         declarationAST = new SequentialDeclaration(declarationAST, d2AST, declarationPos);
836     }
837     return declarationAST;
838 }
839

```

## Compound-Declaration

Para cumplir con lo anterior descrito se crea la implementación de un nuevo método llamado `parseCompoundDeclaration`, el cuál a su vez presenta 2 funcionalidades nuevas con respecto al código suministrado por el profesor, estas son la funcionalidad “recursive” y “local”

Recursive hace el llamado a nuevas reglas implementadas en Parser, y cómo su nombre lo dice, tiene la funcionalidad entre los Procs y Funcs que se implementan en los archivos `.tri.`, el recursive hace el llamado solamente a uno de los métodos de la clase el cual se encarga de la recursividad entre los procesos y funciones.

```

846 Declaration parseCompoundDeclaration() throws SyntaxError{
847     Declaration declarationAST = null; // in case there's a syntactic error the user see this
848
849     SourcePosition declarationPos = new SourcePosition();
850     start(declarationPos);
851     switch (currentToken.kind){
852         case Token.RECURSIVE:
853             {
854                 acceptIt();
855                 Declaration dAST = parseProcFuncS();
856                 accept (Token.END);
857                 finish(declarationPos);
858                 declarationAST = new RecursiveDeclaration(dAST, declarationPos);
859             }
860         break;

```

Local a diferencia de recursive, este hace llamado a métodos dentro de la clase que fueron creados anteriormente, y este cumple con la funcionalidad de analizar la interacción entre Declaraciones.

```

862         case Token.LOCAL:
863             {
864                 acceptIt();
865                 Declaration d1AST = parseDeclaration();
866                 accept(Token.IN);
867                 Declaration d2AST = parseDeclaration();
868                 accept(Token.END);
869                 finish(declarationPos);
870                 declarationAST = new LocalDeclaration(d1AST, d2AST, declarationPos);
871             }
872         break;

```

## Proc-Func

Esta es una de las nuevas reglas de las que la funcionalidad recursiva hace uso, esta tiene dos procesos internos, un proc y un func, ambas inicialmente tienen un estructura similar, la cuál cambia en la que retornan los elementos, siendo el proc el que una parte de su estructura cómo Command y func que tiene un analizador de tipo seguido por una expresión

```
892 Declaration parseProcFunc() throws SyntaxError{
893     Declaration dAST = null;
894
895     SourcePosition declarationPos = new SourcePosition();
896     start(declarationPos);
897
898     switch (currentToken.kind) {
899
900     case Token.PROC:
901     {
902         acceptIt();
903         Identifier iAST = parseIdentifier();
904         accept(Token.LPAREN);
905         FormalParameterSequence fpsAST = parseFormalParameterSequence();
906         accept(Token.RPAREN);
907         accept(Token.IS);
908         Command cAST = parseCommand();
909         accept(Token.END);
910         finish(declarationPos);
911         dAST = new ProcDeclaration(iAST, fpsAST, cAST, declarationPos);
912     }
913     break;
914
915     case Token.FUNC:
916     {
917         acceptIt();
918         Identifier iAST = parseIdentifier();
919         accept(Token.LPAREN);
920         FormalParameterSequence fpsAST = parseFormalParameterSequence();
921         accept(Token.RPAREN);
922         accept(Token.COLON);
923         TypeDenoter tAST = parseTypeDenoter();
924         accept(Token.IS);
925         Expression eAST = parseExpression();
926         finish(declarationPos);
927         dAST = new FuncDeclaration(iAST, fpsAST, tAST, eAST, declarationPos);
928     }
929     break;
930
931     default:
932     syntacticError("\"%s\" cannot start a declaration",
933         currentToken.spelling);
934     break;
935 }
```

## Proc-Funcs

Este método es una de las nuevas reglas implementadas, este método es el que presenta la recursividad de los elementos Proc-Func mencionados anteriormente, el cual se lleva a cabo de la siguiente manera, el método siempre hace una llamada a la funcionalidad Proc-Func, pero no se limita a únicamente una llamada, sino que, el método puede recibir de forma indefinida elementos de este mismo tipo siempre que cumpla con la sintaxis correcta.

```

941 Declaration parseProcFuncS() throws SyntaxError{
942     Declaration dAST = null;
943
944     Declaration procAST1 = null;
945     Declaration procAST2 = null;
946
947     SourcePosition declarationPos = new SourcePosition();
948     start(declarationPos);
949
950
951     procAST1 = parseProcFunc();
952     finish(declarationPos);
953
954     if(currentToken.kind == Token.PIPE){
955         while (currentToken.kind == Token.PIPE){
956             accept(Token.PIPE);
957             start(declarationPos);
958             procAST2 = parseProcFunc();
959             finish(declarationPos);
960             dAST = new ProcFuncSDeclaration(procAST1, procAST2, declarationPos);
961         }
962     }else{
963         syntacticError("\'%\' not expected parsing proc-func expression, expected |", currentToken.spelling);
964     }
965
966     return dAST;
967 }

```

## Cases

Es una de las nuevas reglas implementadas, estas reglas para lograr ser efectivas requieren una serie de interacciones entre reglas, el llamado a este método se lleva a cabo por medio del método Single-Command el cual tiene un apartado SELECT, este es el encargado de iniciar toda la funcionalidad en los casos. Si se centra solo en la implementación del ParseCase, este tiene la funcionalidad de identificar los componentes necesarios para iniciar el análisis, el cuál puede recibir uno o más casos dependiendo de los requerimientos, así como el uso o no de un else como respuesta.

```

1072 Declaration parseCases() throws SyntaxError{
1073     Declaration declarationAST = null; // in case there's a syntactic error the user see this
1074
1075     Declaration casesAST1 = null;
1076     Declaration casesAST2 = null;
1077     Command casesAST3 = null;
1078
1079     SourcePosition declarationPos = new SourcePosition();
1080     start(declarationPos);
1081
1082     accept(Token.WHEN);
1083     casesAST1 = parseCase();
1084     finish(declarationPos);
1085
1086     while (currentToken.kind == Token.WHEN){
1087         acceptIt();
1088         casesAST2 = parseCase();
1089         finish(declarationPos);
1090         casesAST2 = new SelectWhen(casesAST2, declarationPos);
1091     }
1092
1093     if(currentToken.kind == Token.ELSE){
1094         acceptIt();
1095         casesAST3 = parseElseCase();
1096         finish(declarationPos);
1097     }
1098
1099     return casesAST1;
1100 }

```



## Case

Este método tiene la funcionalidad más importante de la ejecución, decide cual será la implementación más atómica que será realizada, por ende, analiza dos posibles casos, el primero donde se haga uso de un RANGE para la aplicación de la funcionalidad o que reciba un literal, ya sea numeral o un carácter letra.

```
1102 Declaration parseCase() throws SyntaxError{
1103     Declaration dAST = null; // in case there's a syntactic error the user see this
1104     Declaration dAST1 = null;
1105     Declaration dAST2 = null;
1106     Command cAST = null;
1107
1108     SourcePosition declarationPos = new SourcePosition();
1109     start(declarationPos);
1110
1111     if (currentToken.kind == Token.RANGE){
1112         acceptIt();
1113         dAST1 = parseCaseLiteral();
1114         accept(Token.DOUBLEDOT);
1115         dAST2 = parseCaseLiteral();
1116         finish(declarationPos);
1117         dAST1 = new RangeDeclaration(dAST1, dAST2, declarationPos);
1118     }else if (currentToken.kind == Token.INTLITERAL || currentToken.kind == Token.CHARLITERAL){
1119         dAST1 = parseCaseLiteral();
1120         finish(declarationPos);
1121         dAST1 = new CaseLiteralDeclaration(dAST1, declarationPos);
1122     }else{
1123         syntacticError("\'%\" expected literal, char or range declaration\", currentToken.spelling);
1124     }
1125     accept(Token.THEN);
1126     cAST = parseCommand();
1127     finish(declarationPos);
1128
1129     return dAST1;
1130 }
1131
```

## ElseCase

Esta regla es llamada cuando en el método case se recibe un else, recordemos que existe la posibilidad de que tenga o no una llamada al else, este devuelve un comando con la información recibida del select en su llamada.

```
1133 Command parseElseCase() throws SyntaxError{
1134     Command commandAST = null; // in case there's a syntactic error the user see this
1135
1136     SourcePosition commandPos = new SourcePosition();
1137     start(commandPos);
1138
1139     commandAST = parseCommand();
1140     commandAST = new ElseCaseCommand(commandAST, commandPos);
1141
1142     return commandAST;
1143 }
```

## Case-Literal

Este es el método más atómico de la implementación de los casos, el cual tiene cómo funcionalidad analizar si la información que recibe es un literal numérico o un carácter letra.

```

1145 Declaration parseCaseLiteral() throws SyntaxError{
1146     Declaration declarationAST = null; // in case there's a syntactic error the user see this
1147
1148     IntegerLiteral intAST = null;
1149     CharacterLiteral charAST = null;
1150
1151     SourcePosition declarationPos = new SourcePosition();
1152     start(declarationPos);
1153
1154     switch (currentToken.kind){
1155     case Token.INTLITERAL:
1156     {
1157         //acceptIt();
1158         intAST = parseIntegerLiteral();
1159         finish(declarationPos);
1160         declarationAST = new LiteralDeclaration(intAST, declarationPos);
1161     }
1162     break;
1163     case Token.CHARLITERAL:
1164     {
1165         //acceptIt();
1166         charAST = parseCharacterLiteral();
1167         finish(declarationPos);
1168         declarationAST = new CharDeclaration(charAST, declarationPos);
1169     }
1170     break;
1171     default:
1172         syntacticError("\"\" expected literal or char", currentToken.spelling);
1173     break;
1174 }
1175

```

## Token.java

Se agregaron nuevas palabras reservadas indicadas en las especificaciones del proyecto, tal como, FOR, FROM, LOCAL, RANGE, RECURSIVE, REPEAT, SELECT, SKIP, TO, UNTIL, WHEN, DOUBLEDOT y PIPE. Sin embargo, estos fueron nombres asignados a las numeraciones tanto de variables como de símbolos.

La escritura de todas estas palabras reservadas y signos de puntuación propiamente en el lenguaje es igual, solo cambia en que sus nombres de variables son escritos en mayúsculas, pero las palabras reservadas y comandos en el lenguaje como tal, deben escribirse en minúsculas, a excepción de DOUBLEDOT que son doble puntos escritos de la siguiente manera '..', y PIPE que es '|'.

```

90     THEN                = 24,
91     TO                  = 25, // nuevo
92     TYPE                = 26,
93     UNTIL               = 27, // nuevo
94     VAR                 = 28,
95     WHEN                = 29, // nuevo
96     WHILE               = 30,
97
98     // punctuation...
99     DOT                 = 31,
100    DOUBLEDOT            = 32, // nuevo
101    COLON                = 33,
102    SEMICOLON            = 34,
103    COMMA                = 35,
104    BECOMES              = 36,
105    IS                   = 37,
106    PIPE                 = 38, // nuevo

```

## Abstract Syntax Trees

Para cada AST se creó una clase con su respectivo nombre en las cuales se creó un método constructor el cual recibe como parámetros expresiones, declaraciones o comandos según los requiera cada clase, además de la implementación del método visit el cual recibe dos parámetros, el primero es un objeto de Tipo y el segundo es de tipo Object.

- RepeatInCommand

```
14 public class RepeatInCommand extends Command {
15
16     public RepeatInCommand (Declaration dAST, Command cAST,
17                             SourcePosition thePosition) {
18         super (thePosition);
19         D = dAST;
20         C = cAST;
21     }
22
23     public Object visit(Visitor v, Object o) {
24         return v.visitRepeatInCommand(this, o);
25     }
26
27     public Declaration D;
28     public Command C;
29 }
30
```

- ForRangeIdentifierExpression

```
14 public class ForRangeIdentifierExpression extends Declaration {
15
16     public ForRangeIdentifierExpression (Identifier iAST, Expression eAST,
17                                         SourcePosition thePosition) {
18         super (thePosition);
19         I = iAST;
20         E = eAST;
21     }
22
23     public Object visit(Visitor v, Object o) {
24         return v.visitForRangeIdentifierExpression(this, o);
25     }
26
27     public Identifier I;
28     public Expression E;
29 }
30
```

- RepeatWhileDoCommand.java

```

14 public class RepeatWhileDoCommand extends Command {
15
16     public RepeatWhileDoCommand (Expression eAST, Command cAST, SourcePosition
17         thePosition) {
18         super (thePosition);
19         E = eAST;
20         C = cAST;
21     }
22
23     public Object visit(Visitor v, Object o) {
24         return v.visitRepeatWhileDo(this, o);
25     }
26
27     public Expression E;
28     public Command C;
29 }

```

- RepeatUntilDoCommand

```

14 public class RepeatUntilDoCommand extends Command {
15
16     public RepeatUntilDoCommand (Expression eAST, Command cAST,
17         SourcePosition thePosition) {
18         super (thePosition);
19         E = eAST;
20         C = cAST;
21     }
22
23     public Object visit(Visitor v, Object o) {
24         return v.visitRepeatUntilDo(this, o);
25     }
26
27     public Expression E;
28     public Command C;
29 }

```

- RepeatDoWhileCommand

```

14 public class RepeatDoWhileCommand extends Command{
15
16     public RepeatDoWhileCommand (Command cAST, Expression eAST,
17         SourcePosition thePosition) {
18         super (thePosition);
19         C = cAST;
20         E = eAST;
21     }
22
23     public Object visit(Visitor v, Object o) {
24         return v.visitRepeatDoWhileCommand(this, o);
25     }
26
27     public Command C;
28     public Expression E;
29 }

```

- RepeatDoUntilCommand

```

14 public class RepeatDoUntilCommand extends Command {
15
16     public RepeatDoUntilCommand (Command cAST, Expression eAST,
17         SourcePosition thePosition) {
18         super (thePosition);
19         C = cAST;
20         E = eAST;
21     }
22
23     public Object visit(Visitor v, Object o) {
24         return v.visitRepeatDoUntilCommand(this, o);
25     }
26
27     public Command C;
28     public Expression E;
29
30 }

```

- RepeatForRangeDoCommand

```

14 public class RepeatForRangeDoCommand extends Command{
15
16     public RepeatForRangeDoCommand (Declaration dAST, Expression eAST,
17         Command cAST, SourcePosition thePosition) {
18         super (thePosition);
19         D = dAST;
20         E = eAST;
21         C = cAST;
22     }
23
24
25     public Object visit(Visitor v, Object o) {
26         return v.visitRepeatForRangeDoCommand(this, o);
27     }
28
29     public Declaration D;
30     public Expression E;
31     public Command C;
32
33 }

```

- RepeatForRangeWhileCommand

```

14 public class RepeatForRangeWhileCommand extends Command {
15
16     public RepeatForRangeWhileCommand (Declaration dAST, Expression e1AST,
17         Expression e2AST, Command cAST, SourcePosition thePosition) {
18         super (thePosition);
19         D = dAST;
20         E1 = e1AST;
21         E2 = e2AST;
22         C = cAST;
23     }
24
25     public Object visit(Visitor v, Object o) {
26         return v.visitRepeatForRangeWhileCommand(this, o);
27     }
28
29     public Declaration D;
30     public Expression E1, E2;
31     public Command C;
32
33 }

```

- RepeatForRangeUntilCommand

```

14 public class RepeatForRangeUntilCommand extends Command {
15
16     public RepeatForRangeUntilCommand (Declaration dAST, Expression e1AST,
17         Expression e2AST, Command cAST, SourcePosition thePosition) {
18         super (thePosition);
19         D = dAST;
20         E1 = e1AST;
21         E2 = e2AST;
22         C = cAST;
23     }
24
25     public Object visit(Visitor v, Object o) {
26         return v.visitRepeatForRangeUntilCommand(this, o);
27     }
28
29     public Declaration D;
30     public Expression E1, E2;
31     public Command C;
32
33 }

```

- RecursiveDeclaration

```

19 public class RecursiveDeclaration extends Declaration {
20
21     public RecursiveDeclaration (Declaration d1AST,
22         SourcePosition thePosition) {
23         super (thePosition);
24         D1 = d1AST;
25     }
26
27     public Object visit(Visitor v, Object o) {
28         return v.visitRecursiveDeclaration(this, o);
29     }
30
31     public Declaration D1;
32 }

```

- LocalDeclaration

```

19 public class LocalDeclaration extends Declaration {
20
21     public LocalDeclaration (Declaration d1AST, Declaration d2AST,
22         SourcePosition thePosition) {
23         super (thePosition);
24         D1 = d1AST;
25         D2 = d2AST;
26     }
27
28     public Object visit(Visitor v, Object o) {
29         return v.visitLocalDeclaration(this, o);
30     }
31
32     public Declaration D1, D2;
33 }

```

- ProcDeclaration

```

19 | public class ProcDeclaration extends Declaration {
20 |
21 |     public ProcDeclaration (Identifier iAST, FormalParameterSequence fpsAST,
22 |                             Command cAST, SourcePosition thePosition) {
23 |         super (thePosition);
24 |         I = iAST;
25 |         FPS = fpsAST;
26 |         C = cAST;
27 |     }
28 |
29 |     public Object visit (Visitor v, Object o) {
30 |         return v.visitProcDeclaration(this, o);
31 |     }
32 |
33 |     public Identifier I;
34 |     public FormalParameterSequence FPS;
35 |     public Command C;
36 | }
37 |

```

- FuncDeclaration

```

19 | public class FuncDeclaration extends Declaration {
20 |
21 |     public FuncDeclaration (Identifier iAST, FormalParameterSequence fpsAST,
22 |                             TypeDenoter tAST, Expression eAST,
23 |                             SourcePosition thePosition) {
24 |         super (thePosition);
25 |         I = iAST;
26 |         FPS = fpsAST;
27 |         T = tAST;
28 |         E = eAST;
29 |     }
30 |
31 |     public Object visit (Visitor v, Object o) {
32 |         return v.visitFuncDeclaration(this, o);
33 |     }
34 |
35 |     public Identifier I;
36 |     public FormalParameterSequence FPS;
37 |     public TypeDenoter T;
38 |     public Expression E;
39 | }

```

- ProcFuncSDeclaration

```

19 | public class ProcFuncSDeclaration extends Declaration {
20 |
21 |     public ProcFuncSDeclaration (Declaration d1AST, Declaration d2AST,
22 |                                   SourcePosition thePosition) {
23 |         super (thePosition);
24 |         D1 = d1AST;
25 |         D2 = d2AST;
26 |     }
27 |
28 |     public Object visit(Visitor v, Object o) {
29 |         return v.visitProcFuncSDeclaration(this, o);
30 |     }
31 |
32 |     public Declaration D1, D2;
33 | }

```

- VarExpressionDeclaration

```
19 public class VarExpressionDeclaration extends Declaration {
20
21     public VarExpressionDeclaration (Identifier iAST, Expression eAST,
22                                     SourcePosition thePosition) {
23         super (thePosition);
24         I = iAST;
25         E = eAST;
26     }
27
28     public Object visit(Visitor v, Object o) {
29         return v.visitVarExpressionDeclaration(this, o);
30     }
31
32     public Identifier I;
33     public Expression E;
34 }
```

- SelectCommand

```
19 public class selectCommand extends Command {
20
21     public selectCommand (Declaration dAST, Expression eAST, SourcePosition thePosition) {
22         super (thePosition);
23         D = dAST;
24         E = eAST;
25     }
26
27     public Object visit(Visitor v, Object o) {
28         return v.visitSelectCommand(this, o);
29     }
30
31     public Declaration D;
32     public Expression E;
33 }
34
```

- RangeDeclaration

```
19 public class RangeDeclaration extends Declaration {
20
21     public RangeDeclaration (Declaration d1AST, Declaration d2AST,
22                             SourcePosition thePosition) {
23         super (thePosition);
24         D1 = d1AST;
25         D2 = d2AST;
26     }
27
28     public Object visit(Visitor v, Object o) {
29         return v.visitRangeDeclaration(this, o);
30     }
31
32     public Declaration D1, D2;
33 }
```

- CaseLiteralDeclaration

```
19 public class CaseLiteralDeclaration extends Declaration {
20
21     public CaseLiteralDeclaration (Declaration d1AST, SourcePosition thePosition) {
22         super (thePosition);
23         D1 = d1AST;
24     }
25
26     public Object visit(Visitor v, Object o) {
27         return v.visitCaseLiteralDeclaration(this, o);
28     }
29
30     public Declaration D1;
31 }
```



- ElseCaseCommand

```

14 public class ElseCaseCommand extends Command {
15
16     public ElseCaseCommand (Command cAST, SourcePosition thePosition) {
17         super (thePosition);
18         C = cAST;
19     }
20
21     public Object visit(Visitor v, Object o) {
22         return v.visitElseCaseCommand(this, o);
23     }
24
25     public Command C;
26
27 }

```

- LiteralDeclaration

```

20 public class LiteralDeclaration extends Declaration {
21
22     public LiteralDeclaration (IntegerLiteral ilAST,
23                               SourcePosition thePosition) {
24         super (thePosition);
25         IL = ilAST;
26     }
27
28     public Object visit(Visitor v, Object o) {
29         return v.visitLiteralDeclaration(this, o);
30     }
31
32     public IntegerLiteral IL;
33 }

```

- CharDeclaration

```

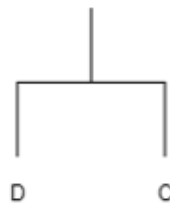
19 public class CharDeclaration extends Declaration {
20
21     public CharDeclaration (CharacterLiteral chAST,
22                             SourcePosition thePosition) {
23         super (thePosition);
24         CH = chAST;
25     }
26
27     public Object visit(Visitor v, Object o) {
28         return v.visitCharDeclaration(this, o);
29     }
30
31     public CharacterLiteral CH;
32 }

```

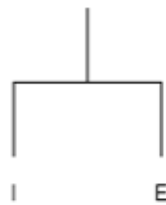
## Modelos

D: Declaration    C: Command  
E: Expression    I: Identifier

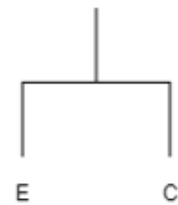
RepeatInCommand



ForRangeIdentifierExpression

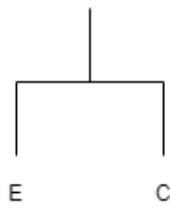


RepeatWhileDoCommand

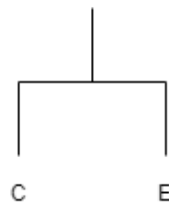


C: Command  
E: Expression

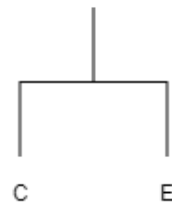
RepeatUntilDoCommand



RepeatDoWhileCommand

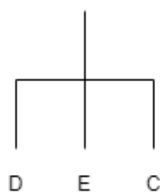


RepeatDoUntilCommand



D: Declaration    C: Command  
E: Expression    I: Identifier  
E1: Expression    E2: Expression

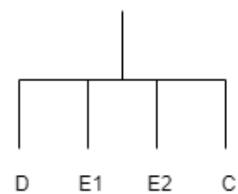
RepeatForRangeDoCommand



RepeatForRangeWhileCommand

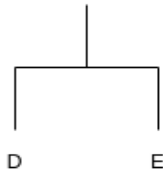


RepeatForRangeWhileCommand



D: Declaration    C: Command  
D1: Declaration   IL: IntegerLiteral  
D2: Declaration   E: Expression

SelectCommand



RangeDeclaration



• CaseLiteralDeclaration



ElseCaseCommand



LiteralDeclaration



• CharDeclaration



## Tree Drawer

### LayoutVisitor.java

Para cada AST nuevo se creó un método con nombre visit y el nombre del AST, por ejemplo, para el AST *RepeatDoWhileCommand* se creó un método llamado *visitRepeatDoWhileCommand* y así con todos los AST nuevos. Estos métodos reciben dos parámetro de entrada, el primero es un ast del tipo respectivo de cada método, siguiendo con el ejemplo anterior el método *visitRepeatDoWhileCommand* recibiría como primer parámetro un objeto de tipo *RepeatDoWhileCommand* y como segundo parámetro un objeto de tipo Object, y esto se repite para todos los demás métodos.

Estos métodos diseñan los AST ya sean de tipo binario, ternario o cuaternario según corresponda para cada AST. Tal como el AST *RepeatDoWhileCommand* que está compuesto únicamente por un comando y una expresión entonces retornaría el diseño de un AST de tipo binario, tal como se muestra en la siguiente imagen.

```

142 public Object visitRepeatDoWhileCommand(RepeatDoWhileCommand ast, Object obj)
143     return layoutBinary("RepeatDoWhileCom", ast.C, ast.E);
  
```

Y lo mismo se repetiría en los métodos de los AST de tipo comando como *RepeatWhileDoCommand*, *RepeatUntilDoCommand*, *RepeatDoUntilCommand*, *RepeatInCommand*, *RepeatForRangeCommand*, *RepeatForRangeWhileCommand*, *RepeatForRangeUntilCommand* y la declaración *ForRangeIdentifierExpression*, tal como se muestra a continuación:

## RepeatWhileDoCommand

```
134 public Object visitRepeatWhileDo(RepeatWhileDoCommand ast, Object obj) {  
135     return layoutBinary("RepeatWhileCom", ast.E, ast.C);  
}
```

## RepeatUntilDoCommand

```
138 public Object visitRepeatUntilDo(RepeatUntilDoCommand ast, Object obj) {  
139     return layoutBinary("RepeatUntilCom", ast.E, ast.C);  
}
```

## RepeatDoUntilCommand

```
146 public Object visitRepeatDoUntilCommand(RepeatDoUntilCommand ast, Object o) {  
147     return layoutBinary("RepeatDoUntilCom", ast.C, ast.E);  
}
```

## RepeatInCommand

```
150 public Object visitRepeatInCommand(RepeatInCommand ast, Object o) {  
151     return layoutBinary("RepeatInCom", ast.D, ast.C);  
}
```

## RepeatForRangeCommand

```
154 public Object visitRepeatForRangeCommand(RepeatForRangeCommand ast, Object o) {  
155     return layoutTernary("RepeatForRangeCom", ast.D, ast.E, ast.C);  
}
```

## RepeatForRangeWhileCommand

```
158 public Object visitRepeatForRangeWhileCommand(RepeatForRangeWhileCommand ast, Object o) {  
159     return layoutQuaternary("RepeatForRangeWhileCom", ast.D, ast.E1, ast.E2, ast.C);  
}
```

## RepeatForRangeUntilCommand

```
162 public Object visitRepeatForRangeUntilCommand(RepeatForRangeUntilCommand ast, Object o) {  
163     return layoutQuaternary("RepeatForRangeUntilCom", ast.D, ast.E1, ast.E2, ast.C);  
}
```

## ForRangeIdentifierExpression

```
248 public Object visitForRangeIdentifierExpression(ForRangeIdentifierExpression ast, Object o) {  
249     return layoutBinary("ForRangeIdentifierExpressionDecl", ast.I, ast.E);  
}
```

De igual forma, esta estructura se repite en las reglas, para las cuales se crearon árboles como RecursiveDeclaration, LocalDeclaration, ProcFuncSDeclaration y VarExpressionDeclaration, los cuales se muestran a continuación:

### RecursiveDeclaration

```
274 public Object visitRecursiveDeclaration(R RecursiveDeclaration ast, Object o) {  
275     return layoutUnary("RecursiveDeclaration", ast.D1); //Se agrego el metodo  
276 }
```

### LocalDeclaration

```
262 public Object visitLocalDeclaration(LocalDeclaration ast, Object o) {  
263     return layoutBinary("LocalDeclaration", ast.D1, ast.D2); //Se agrego el metodo  
264 }
```

### ProcFuncSDeclaration

```
270 public Object visitProcFuncSDeclaration(ProcFuncSDeclaration ast, Object o) {  
271     return layoutBinary("ProcFuncSDeclaration", ast.D1, ast.D2); //Se agrego el metodo  
272 }
```

### VarExpressionDeclaration

```
266 public Object visitVarExpressionDeclaration(VarExpressionDeclaration ast, Object o) {  
267     return layoutBinary("VarExpressionDeclaration", ast.E, ast.I); //Se agrego el metodo  
268 }
```

### SelectCommand

```
282 public Object visitSelectCommand(selectCommand ast, Object o) {  
283     return layoutBinary("selectCommand", ast.D, ast.E); //Se agrego el metodo  
284 }
```

### RangeDeclaration

```
286 public Object visitRangeDeclaration(RangeDeclaration ast, Object o) {  
287     return layoutBinary("RangeDeclaration", ast.D1, ast.D2); //Se agrego el metodo  
288 }
```

### CaseLiteralDeclaration

```
290 public Object visitCaseLiteralDeclaration(CaseLiteralDeclaration ast, Object o) {  
291     return layoutUnary("CaseLiteralDeclaration", ast.D1); //Se agrego el metodo  
292 }
```

### ElseCaseCommand

```
278 public Object visitElseCaseCommand(ElseCaseCommand ast, Object o) {  
279     return layoutUnary("ElseCaseCommand", ast.C); //Se agrego el metodo  
280 }
```

## LiteralDeclaration

```
294 public Object visitLiteralDeclaration(LiteralDeclaration ast, Object o) {  
295     return layoutUnary("LiteralDeclaration", ast.IL); //Se agrego el metodo  
    }
```

## CharDeclaration

```
298 public Object visitCharDeclaration(CharDeclaration ast, Object o) {  
299     return layoutUnary("CharDeclaration", ast.CH); //Se agrego el metodo  
    }
```

## HTMLWriter

### Lógica utilizada

Para generar el HTML se planteó cómo estrategia hacer uso de las estructuras ya creadas, de esta forma nos aseguramos que la creación del archivo .html siga el mismo comportamiento que el resto de la aplicación.

Inicialmente la llamada a esta funcionalidad se hace desde el IDE Compiler en el cual se crea un writerHTML que trae las funcionalidades de HTML, que es la clase encargada de la evaluación y generación del contenido del archivo. Desde el mismo IDE Compiler se hace la llamada al método generateHTML método encargado de iniciar el proceso de creación, el cual recibe cómo parámetro el destino del archivo a crear.

El método generateHTML tiene cómo principal funcionalidad crear el encabezado del HTML, y es importante porque en este encabezado es donde se marcan los estilos que van a tener las diferentes partes de la estructura del archivo a compilar, por lo que se declara que:

- Los comentarios se mostrarán en color verde
- Los literales tendrán un color azul
- Las palabras reservadas tienen un formato 'bold'
- El resto de los componentes tienen un texto normal, siendo monoespaciadas y de tamaño 1em

Posteriormente hace la llamada al método scanToken que es la parte principal de la creación del archivo HTML, cómo se mencionó anteriormente se hace uso de estructuras ya creadas para seguir la misma línea de trabajo, la estructura scanToken es una estructura representada en Scanner, sin embargo, para la generación del HTML se hacen algunos cambios, cómo por ejemplo, la revisión de los componentes para enviarlos a métodos que escriban en el archivo .html, ya no haciendo la identificación de los tokens cómo era su funcionalidad principal.

Estos datos son enviados a diferentes métodos encargados de principalmente escribir en el archivo html, una vez analizados toda la estructura el archivo .tri que se está compilando, se crea un archivo en el mismo directorio donde se encuentra el archivo.tri, el cual se puede abrir en el navegador y mostrará en la web la misma estructuras del archivo .tri con las diferencias en los colores mencionados en el encabezado a fin de diferenciar la funcionalidad de cada uno de los elementos.

## HTML.java

Encabezado del HTML, en este se especifican el tipo de letra y color de los elementos

```
public void generateHTML(String fileName) {
    this.content+="<!DOCTYPE html>\n"
        + "<html>\n"
        + "<head>\n"
        + "<style>\n"
        + "\t\t\t\t\t{font-size: 1em; font-family: \"Courier New\", monospaced;}\n"
        + "\t\t\t\t\t.literal{color : #004080;}\n"
        + "\t\t\t\t\t.comment{color: #009933;}\n"
        + "\t\t\t\t\t.reservedword {font-weight:bold;}\n"
        + "</style>\n"
        + "</head>\n"
        + "<body>\n";

    this.scan();
    this.content+="</body>\n" +
        "</html>";

    try {
        if (!this.error){
            fileWriter = new FileWriter(fileName + ".html");
            fileWriter.write(this.content);
            fileWriter.close();
        }
    } catch (IOException e) {
        System.err.println("Error while writing HTML file for print the AST");
        e.printStackTrace();
    }
}
```

Métodos para la creación de los elementos del HTML

```

public void writeComment(String comment) {
    this.content+="(<span class='comment'>" +comment+"</span>");
}

public void writeReservedWord(String spelling) {
    this.content+="(<strong>" +spelling+"</strong>");
}

public void writeLiteralWord(String spelling) {
    this.content+="(<span class='literal'>" +spelling+"</span>\n");
}

public void writeNormalWord(String spelling) {
    this.content+="(<span>" +spelling+"</span>\n");
}

```

## XMLWriter

### Lógica utilizada

Crear un archivo con extensión XML y crear un método el cual recibiera un String como parámetro y que esto lo escribiera en el archivo creado anteriormente. Lo que entraría como parámetro serían las etiquetas de iniciación y cierre del XML.

### XMLWriterVisitor.java

Para la representación de los árboles de sintaxis abstracta como texto en XML se creó un método llamado *writeLineXML*, el cual es el encargado de escribir las etiquetas correspondientes en el archivo llamado *fileWriter*. Dicho método fue creado en la clase *XMLWriterVisitor.java* ubicada en el paquete *XMLWriter*.

Al método *writeLineXML(String line)* que realiza la escritura en el archivo le ingresa como parámetro el nombre de la etiqueta que se escribirá en el XML la cual es de tipo *String* y por medio del método *write()* que ya tiene instanciado la librería *java.io.FileWriter* la cual debe ser importada, se le introduce como parámetro la etiqueta line que ingresó anteriormente como parámetro en el método, a manera de ejemplo, lo anteriormente explicado se denota de la siguiente manera:

**[nombre del archivo].write([etiqueta que será escrita en el archivo]) = *fileWriter.write(line)*.**



El cuerpo del método está compuesto por la estructura `try {} Catch {}`, es decir, si por alguna razón ocurre un error durante la escritura en el archivo este imprimirá el mensaje *"Error writing to XML file to print Abstract Syntax Tree"*, de lo contrario realizará la escritura en el archivo, acción explicada anteriormente.

La clase *XMLWriterVisitor.java* contiene los métodos ubicados también en la clase *Visitor.java* existente anteriormente en el proyecto. A esta estructura se le agregó en cada método de los *Comand*, *Expressions*, *Declarations*, *Array Aggregates*, *Record Aggregates*, *Formal Parameters*, *Actual Parameters*, *Type Denoters*, *Literals*, *Identifiers and Operators*, *Values or Variable Names* y *Programs*, el llamado al método *writeLineXML(String line)* anteriormente descrito. En cada llamado a los métodos presentes en dicha clase, se le ingresa como parámetro el nombre de la etiqueta ejemplo *"writeLineXML("<AssignCommand>")* y se llaman a sus respectivos comandos o expresiones dependiendo de la estructura de cada método, y al terminar dichos llamados se llama nuevamente al método, pero ahora le ingresa como parámetro la etiqueta de cierre del XML, ejemplo *"writeLineXML("</AssignCommand>")*. En caso de ser una estructura empty únicamente se llamaría una vez al método y con la etiqueta de cierre de esta forma *("<EmptyCommand/>")*.

## **XMLWriterTree.java**

Para la creación del AST en el archivo XML se creó el método llamado *writer* el cual recibe un parámetro de tipo *String* que contiene la ruta en la que se encuentra el archivo que fue compilado. A esta ruta se le añade un *".xml"* para crear un archivo XML y que se cree en la misma ruta del programa compilado. Se crea un nuevo objeto de tipo *XMLWriterVisitor* y se le envía como parámetro el archivo XML creado de tipo *fileWriter*.

Después en la clase *XMLWriterVisitor* se crea el AST como se mencionó en el apartado anterior donde se explica la clase *XMLWriterVisitor*, y por último se cierra el archivo XML. A continuación se muestra el método anteriormente explicado.

```

24 public void writer(Program ast) {
25     //Preparar el archivo para escribir
26     try {
27         FileWriter fileWriter = new FileWriter(filesDestination+".xml");
28
29
30         //Encabezado XML
31         fileWriter.write("<?xml version=\"1.0\" standalone=\"yes\"?>\n");
32         XMLWriterVisitor layout = new XMLWriterVisitor(fileWriter);
33
34         ast.visit(layout, null);
35
36         fileWriter.close();
37
38     } catch (IOException e) {
39         System.err.println("Error while creating file for print the AST");
40         e.printStackTrace();
41     }
42 }
43

```

## Nuevos errores sintácticos

Al realizar modificaciones dentro del lenguaje de triángulo, aparecieron nuevos errores sintácticos dentro del sistema. Por ejemplo al haber agregado la palabra “end” al final de cada comando se retorna el error en caso de no incluirla. Ocurre del mismo modo con cada una de las modificaciones al lenguaje.

```

***** Triangle Compiler (IDE-Triangle 1.0) *****
Syntactic Analysis ...
ERROR: "end" expected here 3..3
Compilation was unsuccessful.

```

## Plan de pruebas

### Pruebas Begin

Prueba	Objetivo del caso de prueba	Resultados esperados	Resultados observados
BeginErr1	! Error: Begin no existe	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
BeginOk1	! OK: begin ya no es palabra reservada ! y el comando puede ser compuesto	Compilar de manera exitosa Mostrar el AST en el IDE Crear el XML y el HTML	Compiló de manera exitosa Mostró el AST en el IDE Creó el XML y el HTML

### Pruebas Empty

Prueba	Objetivo del caso de prueba	Resultados esperados	Resultados observados
<b>EmptyErr1</b>	! el comando vacío ya no puede ser épsilon	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML

### Pruebas If

Prueba	Objetivo del caso de prueba	Resultados esperados	Resultados observados
<b>IfErr1</b>	! Error: Faltó el then (puso paréntesis, como en C)	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
<b>IfErr2</b>	! Error: Falta el else	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
<b>IfErr3</b>	! Error: Puso do en vez de then	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
<b>IfErr4</b>	! Error: Falta el end del if	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
<b>IfErr5</b>	! Error: Falta el then de la segunda alternativa	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
<b>IfErr6</b>	! Error: elif no es lo mismo que	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
<b>IfErr7</b>	! Error: elsif no es lo mismo que	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
<b>IfOk1</b>	! OK	Compilar de manera exitosa Mostrar el AST en el IDE Crear el XML y el HTML	Compiló de manera exitosa Mostró el AST en el IDE Creó el XML y el HTML
<b>IfOk2</b>	! OK	Compilar de manera exitosa Mostrar el AST en el IDE	Compiló de manera exitosa Mostró el AST en el IDE

		Crear el XML y el HTML	Creó el XML y el HTML
--	--	------------------------	-----------------------

### Pruebas Let

Prueba	Objetivo del caso de prueba	Resultados esperados	Resultados observados
<b>LetErr1</b>	! Error: Falta el in	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
<b>LetErr2</b>	! Error: declaración vacía	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
<b>LetErr3</b>	! Error: Faltan los comandos	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
<b>LetErr4</b>	! end ! Error: Falta el end	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
<b>LetErr5</b>	! el ; separa declaraciones, no las termina ! hubo ; pero no hay declaración ! in no puede iniciar una declaración	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
<b>LetOk1</b>	! OK	Compilar de manera exitosa Mostrar el AST en el IDE Crear el XML y el HTML	Compiló de manera exitosa Mostró el AST en el IDE Creó el XML y el HTML

## Pruebas local

Prueba	Objetivo del caso de prueba	Resultados esperados	Resultados observados
<b>LocalErr1</b>	! Error: Falta el in del local	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
<b>LocalErr2</b>	! Error: Falta el end del local	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
<b>LocalErr3</b>	! Error: declaración privada en el local quedó vacía	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
<b>LocalErr4</b>	! Error: Segunda declaración (la parte pública) en local está vacía	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
<b>LocalOk1</b>	! Ok	Compilar de manera exitosa Mostrar el AST en el IDE Crear el XML y el HTML	Compiló de manera exitosa Mostró el AST en el IDE Creó el XML y el HTML

## Pruebas recursive

Prueba	Objetivo del caso de prueba	Resultados esperados	Resultados observados
<b>RecursiveErr0</b>	! Error: Solo hay una alternativa	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
<b>RecursiveErr1</b>	! Error: Solo hay una alternativa	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
<b>RecursiveErr2</b>	! Error: una de las alternativas no es un proc o func	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
<b>RecursiveErr3</b>	! Error: No hay alternativas	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
<b>RecursiveErr4</b>	! Error: Falta el " " que separe los proc-funcs	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML

<b>RecursiveErr5</b>	! Error: Falta el end del recursive	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
<b>RecursiveErr6</b>	! Error: Solo hay una alternativa, pero antes de   falta un end	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
<b>RecursiveOk1</b>	! OK	Compilar de manera exitosa Mostrar el AST en el IDE Crear el XML y el HTML	Compiló de manera exitosa Mostró el AST en el IDE Creó el XML y el HTML

### Prueba RepeatDoUntil

Prueba	Objetivo del caso de prueba	Resultados esperados	Resultados observados
RepeatDoUntilErr1	! Error: Falta el repeat	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
RepeatDoUntilErr2	! Error: Falta el do	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
RepeatDoUntilErr3	!falta end	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
RepeatDoUntilOk1	! OK	Compilar de manera exitosa Mostrar el AST en el IDE Crear el XML y el HTML	Compiló de manera exitosa Mostró el AST en el IDE Creó el XML y el HTML
RepeatDoUntilOk2	! OK	Compilar de manera exitosa Mostrar el AST en el IDE Crear el XML y el HTML	Compiló de manera exitosa Mostró el AST en el IDE Creó el XML y el HTML

### Pruebas RepeatDoWhile

Prueba	Objetivo del caso de prueba	Resultados esperados	Resultados observados
RepeatDoWhileErr1	! Error: Hay 2 do	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
RepeatDoWhileErr2	! Error: Falta do	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
RepeatDoWhileOk1	! OK	Compilar de manera exitosa Mostrar el AST en el IDE Crear el XML y el HTML	Compiló de manera exitosa Mostró el AST en el IDE Creó el XML y el HTML
RepeatDoWhileOk2	! OK	Compilar de manera exitosa Mostrar el AST en el IDE Crear el XML y el HTML	Compiló de manera exitosa Mostró el AST en el IDE Creó el XML y el HTML

### Pruebas RepeatFor

Prueba	Objetivo del caso de prueba	Resultados esperados	Resultados observados
RepeatForErr1	! falta for	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
RepeatForErr2	! falta :=	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
RepeatForErr3	! falta .. en el range	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
RepeatForErr4	! falta do	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
RepeatForErr5	! falta end	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML

### Pruebas RepeatForIn

Prueba	Objetivo del caso de prueba	Resultados esperados	Resultados observados
RepeatForInErr0	! Falta la expresión	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
RepeatForInErr1	! Falta el do	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
RepeatForInErr2	! Falta el end	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
RepeatForInOk0	! OK	Compilar de manera exitosa Mostrar el AST en el IDE Crear el XML y el HTML	Compiló de manera exitosa Mostró el AST en el IDE Creó el XML y el HTML
RepeatForInOk1	! OK sintácticamente ! Tiene dos errores contextuales: tipo (expresión no es arreglo) e identificación (x no es visible en la expresión)	Compilar de manera exitosa Mostrar el AST en el IDE Crear el XML y el HTML	Compiló de manera exitosa Mostró el AST en el IDE Creó el XML y el HTML

### Pruebas RepeatFor

Prueba	Objetivo del caso de prueba	Resultados esperados	Resultados observados
RepeatForOk1	! OK	Compilar de manera exitosa Mostrar el AST en el IDE Crear el XML y el HTML	Compiló de manera exitosa Mostró el AST en el IDE Creó el XML y el HTML
RepeatForOk2	! OK compuesto	Compilar de manera exitosa Mostrar el AST en el IDE Crear el XML y el HTML	Compiló de manera exitosa Mostró el AST en el IDE Creó el XML y el HTML



### Pruebas RepeatForUntil

Prueba	Objetivo del caso de prueba	Resultados esperados	Resultados observados
RepeatForUntilErr1	!falta until o while después del 6	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
RepeatForUntilErr2	!falta el do	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
RepeatForUntilErr3	!falta el end	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
RepeatForUntilOk1	!ok	Compilar de manera exitosa Mostrar el AST en el IDE Crear el XML y el HTML	Compiló de manera exitosa Mostró el AST en el IDE Creó el XML y el HTML
RepeatForUntilOk2	!ok	Compilar de manera exitosa Mostrar el AST en el IDE Crear el XML y el HTML	Compiló de manera exitosa Mostró el AST en el IDE Creó el XML y el HTML

### Pruebas RepeatForWhile

Prueba	Objetivo del caso de prueba	Resultados esperados	Resultados observados
RepeatForWhileErr1	!falta while o until después del 6	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
RepeatForWhileErr2	!falta el do	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
RepeatForWhileErr3	!falta end	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
RepeatForWhileOk1	!ok	Compilar de manera exitosa Mostrar el AST en el IDE Crear el XML y el HTML	Compiló de manera exitosa Mostró el AST en el IDE Creó el XML y el HTML

RepeatForWhileOk2	!OK	Compilar de manera exitosa Mostrar el AST en el IDE Crear el XML y el HTML	Compiló de manera exitosa Mostró el AST en el IDE Creó el XML y el HTML
-------------------	-----	----------------------------------------------------------------------------------	-------------------------------------------------------------------------------

### Pruebas RepeatUntilDo

Prueba	Objetivo del caso de prueba	Resultados esperados	Resultados observados
RepeatUntilDoErr1	! Error: Falta el repeat antes de until	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
RepeatUntilDoErr2	! Error: Falta el do	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
RepeatUntilDoOk1	! OK	Compilar de manera exitosa Mostrar el AST en el IDE Crear el XML y el HTML	Compiló de manera exitosa Mostró el AST en el IDE Creó el XML y el HTML
RepeatUntilDoOk2	! OK	Compilar de manera exitosa Mostrar el AST en el IDE Crear el XML y el HTML	Compiló de manera exitosa Mostró el AST en el IDE Creó el XML y el HTML

### Pruebas RepeatWhileDo

Prueba	Objetivo del caso de prueba	Resultados esperados	Resultados observados
RepeatWhileDoErr1	! Error vieja sintaxis de while	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
RepeatWhileDoErr2	! Error: falta el do	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
RepeatWhileDoOk1	! OK	Compilar de manera exitosa Mostrar el AST en el IDE Crear el XML y el HTML	Compiló de manera exitosa Mostró el AST en el IDE Creó el XML y el HTML
RepeatWhileDoOk2	! OK	Compilar de manera exitosa Mostrar el AST en el IDE Crear el XML y el HTML	Compiló de manera exitosa Mostró el AST en el IDE Creó el XML y el HTML

## Pruebas Select

Prueba	Objetivo del caso de prueba	Resultados esperados	Resultados observados
SelectErr1	! Error: falta el literal o el range	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
SelectErr2	! Error: falta el range literales	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
SelectErr3	! Error: falta .. literal	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
SelectErr4	! Error: falta expresión en select	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
SelectErr5	! case range debe tener ".." , no ".".	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
SelectErr6	! buen rango ! rango incompleto	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
SelectErr7	! case range bien ! falta literal después de ..	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
SelectErr8	! Error: detecta else espurio	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
SelectOK1	! OK	Compilar de manera exitosa Mostrar el AST en el IDE Crear el XML y el HTML	Compiló de manera exitosa Mostró el AST en el IDE Creó el HTML
SelectOK2	! else OK	Compilar de manera exitosa Mostrar el AST en el IDE Crear el XML y el HTML	Compiló de manera exitosa Creó el HTML
SelectOK3	! OK	Compilar de manera exitosa Mostrar el AST en el IDE Crear el XML y el HTML	Compiló de manera exitosa Creó el HTML
SelectOK4	! OK	Compilar de manera exitosa	Compiló de manera exitosa

		Mostrar el AST en el IDE Crear el XML y el HTML	Creó el HTML
SelectOK5	! comienza select anidado !OK, está en select anidado ! terminó select anidado	Compilar de manera exitosa Mostrar el AST en el IDE Crear el XML y el HTML	Compiló de manera exitosa Creó el HTML
SelectOK6	! OK	Compilar de manera exitosa Mostrar el AST en el IDE Crear el XML y el HTML	Compiló de manera exitosa Creó el HTML
SelectOK7	! OK	Compilar de manera exitosa Mostrar el AST en el IDE Crear el XML y el HTML	Compiló de manera exitosa Creó el HTML

### Pruebas Skip

Prueba	Objetivo del caso de prueba	Resultados esperados	Resultados observados
SkipErr1	! Error: Después del ";" faltaría un skip	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
SkipErr2	! Error: Comando vacío ya no es épsilon	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
SkipOk1	! skip es un comando, consecuentemente un programa válido	Compilar de manera exitosa Mostrar el AST en el IDE Crear el XML y el HTML	Compiló de manera exitosa Mostró el AST en el IDE Creó el XML y el HTML
SkipOk2	! OK	Compilar de manera exitosa Mostrar el AST en el IDE Crear el XML y el HTML	Compiló de manera exitosa Mostró el AST en el IDE Creó el XML y el HTML

### Pruebas VarInit

Prueba	Objetivo del caso de prueba	Resultados esperados	Resultados observados
VarInitErr1	! Error: No se puede declarar fuera de un let o un private	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML

VarInitErr2	! Error: "i" no debería tener declaración de tipo	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
VarInitErr3	! Error sintáctico: inicialización después de denotador de tipo	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
VarInitErr4	! Error: usa "::-=" y no "::-" ! sintaxis inválida, debe ser como la asignación ("::-=")	No compilar Crear el HTML No crear el XML	No compiló por un error sintáctico Creó el HTML No creó el XML
VarInitOk1	! OK	Compilar de manera exitosa Mostrar el AST en el IDE Crear el XML y el HTML	Compiló de manera exitosa Mostró el AST en el IDE Creó el XML y el HTML

## Discusión y análisis de los resultados obtenidos

Para el plan de prueba se decide hacer uso de todas las herramientas brindadas por el profesor, por ende, se plantean todos los casos de prueba y se presentan en una tabla donde se agrupan por cada una de las funcionalidades disponibles. Todas las pruebas se centran en 4 componentes principales, la compilación, creación del AST, creación del XML y HTML.

Cuando se inician las pruebas y documentación de las mismas, se toma la decisión de comenzar con los errores, dado que son los más sencillos de identificar, dado que al compilar el archivo .tri el error se muestra inmediatamente y este mismo error es comparado con la información presentada por el profesor con respecto al motivo de los errores a fin de corroborar que el proceso tanto correcto como incorrecto fuese hecho de manera correcta. De igual forma se llevó a cabo el proceso con el plan de pruebas positivas, el objetivo en este caso es que todas las pruebas muestran que la compilación fue correcta, además de visualizar el HTML creado en cada caso así como su respectivo XML.

Todas las pruebas presentaron el comportamiento esperado, tanto las pruebas que deberían dar error, como las pruebas que son exitosas en la compilación. Se concluye, una vez realizadas las pruebas, que los resultados obtenidos estaban dentro de las expectativas que se tienen en cada uno de los procesos. En general, el grupo se concuerda con que se logró hacer un buen trabajo, y que se logró comprender todo el proceso para un análisis sintáctico y léxico de un compilador.

## **Una reflexión sobre la experiencia de modificar fragmentos de un compilador**

Modificar un analizador léxico y sintáctico creado por alguien más fue muy gratificante y enriquecedor ya que se comprende cómo crearlos de una forma inteligente y sencilla, además de al ir recorriendo el código se aprenden buenas técnicas de programación que siempre son bien recibidas. Además, nos da la posibilidad de poner en práctica lo aprendido en el curso e ir despejando dudas que surgen conforme el programador se adentra más en el desarrollo del código. También es de ayuda para comprender cómo el funcionamiento sintáctico y léxico afecta directamente el correcto funcionamiento de un programa con este.

### **Instrucciones de Compilación**

Para compilar el programa es necesario abrir el proyecto preferiblemente en NetBeans, esto dado que el desarrollo del proyecto ya fue implementado y la valor ahora es mejorarlo, y según instrucciones la mejor opción es este entorno de desarrollo. Posteriormente se abre un nuevo proyecto y se abre la carpeta llamada `ide-triangle-v1.1.src`, esta carpeta contiene toda la parte relevante del proyecto, todas las carpetas que son necesarias para la implementación del código.

Una vez dentro de esta carpeta, hay que dirigirse a la carpeta llamada `src` y posteriormente a la carpeta llamada `GUI`, donde se encuentra un archivo llamado `main.java` que será el encargado de llevar a cabo la ejecución del proyecto. Posteriormente a encontrar este archivo `main.java` lo que resta es la ejecución del mismo, la cuál, si se está en netbeans, será necesario únicamente acceder en las opciones de la parte superior del entorno a la pestaña llamada `RUN` y ahí seleccionar la primera opción, o en su defecto presionar `F6`, que es el atajo del teclado predeterminado para correr un programa en netbeans.

### **Instrucciones de ejecución**

1. Abrir la interfaz del programa, para abrirlo únicamente hay que dar click al archivo `.jar` adjunto.
2. Una vez abierto el programa, se debe acceder al explorador de archivos
3. Elegir en la carpeta de pruebas uno de los archivos `.tri` disponibles para hacer la prueba correspondiente.
4. Cuando se haya cargado el archivo `.tri`, es momento de hacer la compilación, esta se lleva a cabo dando click a las llaves que aparecen en el header de la interfaz del programa.

## Descripción de las tareas realizadas

David

- Agregar tokens y símbolos
- Implementar el Skip
- Implementar el IF y sus modificaciones
- Implementar el Let y sus modificaciones
- Implementar el Repeat
- Implementar la documentación de las partes que modificó

Keila

- Extensión XML
- Factorizar los Repeat
- Implementar los ASTs
- Implementar los visitors
- Implementar la documentación de las partes que modificó

Anner

- Extension HTML
- Modificar Declaration y añadir Compound Declaration
- Implementar Proc-Funcs
- Implementar el Select
- Implementar Cases y derivados
- Implementar la documentación de las partes que modificó

## Anexos

1. **Gramática del lenguaje Triángulo extendido:** En esta sección podemos encontrar la sintaxis y el léxico de Triangle extendido.

```
Program          ::= Command

Command          ::= single-Command
                  | Command ; single-Command

single-Command   ::= skip
                  | V-name := Expression
                  | Identifier ( Actual-Parameter-Sequence )
                  | begin Command end
                  | let Declaration in Command end
                  | if Expression then Command ( | Expression then
Command) * else Command end
                  | select Expression from Cases end
                  | repeat while Expression do Command end
                  | repeat until Expression do Command end
                  | repeat do Command while Expression end
                  | repeat do Command until Expression end
                  | repeat for Identifier := range Expression ..
Expression do Command end
                  | repeat for Identifier := range Expression ..
Expression while Expression do Command end
                  | repeat for Identifier := range Expression ..
Expression until Expression do Command end
                  | repeat for Identifier in Expression do Command end

Expression       ::= secondary-Expression
```



```

        | let Declaration in Expression
        | if Expression then Expression else Expression
secondary-Expression
    ::= primary-Expression
        | secondary-Expression Operator primary-Expression
primary-Expression
    ::= Integer-Literal
        | Character-Literal
        | V-name
        | Identifier ( Actual-Parameter-Sequence )
        | Operator primary-Expression
        | ( Expression )
        | { Record-Aggregate }
        | [ Array-Aggregate ]

Record-Aggregate ::= Identifier ~ Expression
                  | Identifier ~ Expression , Record-Aggregate

Array-Aggregate  ::= Expression
                  | Expression , Array-Aggregate

V-name           ::= Identifier
                  | V-name . Identifier
                  | V-name [ Expression ]

```

```

Declaration      ::= compound-Declaration

                  | Declaration ; compound-Declaration

compound-Declaration

                ::= single-Declaration

                  | recursive Proc-Funcs end

                  | local Declaration in Declaration end

Proc-Func        ::= proc Identifier ( Formal-Parameter-Sequence )

                  ~ Command end

                  | func Identifier ( Formal-Parameter-Sequence )

                  : Type-denoter ~ Expression

Proc-Funcs

                ::= Proc-Func ( | Proc-Func ) +

single-Declaration

                ::= const Identifier ~ Expression

                  | var Identifier : Type-denoter

                  | proc Identifier ( Formal-Parameter-Sequence ) ~

                    Command end

                  | func Identifier ( Formal-Parameter-Sequence )

                    : Type-denoter ~ Expression

                  | type Identifier ~ Type-denoter

                  | var Identifier := Expression

Formal-Parameter-Sequence

                ::=

                  | proper-Formal-Parameter-Sequence

```

proper-Formal-Parameter-Sequence

```
 ::= Formal-Parameter
    | Formal-Parameter , proper-Formal-Parameter-Sequence
```

Formal-Parameter ::= Identifier : Type-denoter

```
    | var Identifier : Type-denoter
    | proc Identifier ( Formal-Parameter-Sequence )
    | func Identifier ( Formal-Parameter-Sequence )
      : Type-denoter
```

Actual-Parameter-Sequence

```
 ::=
    | proper-Actual-Parameter-Sequence
```

proper-Actual-Parameter-Sequence

```
 ::= Actual-Parameter
    | Actual-Parameter , proper-Actual-Parameter-Sequence
```

Actual-Parameter ::= Expression

```
    | var V-name
    | proc Identifier
    | func Identifier
```

Type-denoter ::= Identifier

```
    | array Integer-Literal of Type-denoter
    | record Record-Type-denoter end
```

Record-Type-Denoter

```
 ::= Identifier : Type-denoter  
   | Identifier : Type-denoter , Record-Type-denoter
```

Cases ::= Case+ [ ElseCase ]

```
Case ::= when (Case-Literal | range Case-Literal ..  
            Case-Literal) then Command
```

```
ElseCase ::= else Command
```

```
Case-Literal ::= Integer-Literal | Character-Literal
```

## 2. Léxico de Triángulo

Program	::= (Token Comment Blank)*
Token	::= Integer-Literal Character-Literal Identifier Operator   array const do else end when   func if in let of proc record   then type var while do until   select skip for from local range   to recursive repeat   . : ; , := ~ ( ) [ ] { } . ..
Integer-Literal	::= Digit Digit*
Character-Literal	::= 'Graphic'
Identifier	::= Letter(Letter Digit)*
Operator	::= Op-character Op-Character*
Comment	::= ! Graphic* end-of-line
Blank	::= space tab end-of-line
Graphic	::= Letter Digit Op-character space tab .  : ; ,   ~ ( ) [ ] { } _ ! ' ` \" # \$
Letter	::= a b c d e f g h i j k l m

**n|o|p|q|r|s|t|u|v|w|x|y|z|**

**A|B|C|D|E|F|G|H|I|J|K|L|M|**

**N|O|P|Q|R|S|T|U|V|W|X|Y|Z**

Digit                   **::= 0|1|2|3|4|5|6|7|8|9**

Op-character           **::= +|-|\*|/|=|<|>|\|&|@|%|^|?**