Instituto Tecnológico de Costa Rica

Escuela de Ingeniería en Computación

Compiladores e intérpretes

Proyecto 1

Profesor: Ignacio Trejos Zelaya

Integrante:

Josue Daniel Canales Mena - 2017134770

Bárbara Gutiérrez Quirós - 2017089951

Jose Fabio Perez Alpizar - 2017075708

Gabriel Quesada Brenes - 2017126064

Fecha de entrega: 05/04/2019

Índice

Analizador sintáctico y léxico

Manejo del texto fuente

No se modificó lo existente.

Modificaciones al analizador léxico

Los cambios que se realizaron en el analizador léxico dentro de la clase Scanner.java fue agregar al método de scanToken los nuevos Tokens solicitados, que son los siguientes:

Pipe	|
Range	..
Initialize	::=
Dollar	$

Cambios a los Tokens y estructuras de datos

Dentro de los cambios que se realizaron en la clase Token.java se encuentran:

- Agregar palabras reservadas.
- Añadir nuevos símbolos
- Eliminar la palabra reservada **begin**

Dentro de las palabras reservadas que se agregaron están:

- choose
- for
- from
- loop
- par
- pass
- private
- recursive
- to
- until
- when
- pipe
- initialize
- dollar
- range

De igual manera dentro del método tokenTable se agregaron los caracteres de las palabras reservadas mencionadas anteriormente.

HTML y XML

El html se estructura de la manera que a partir del AST, se genere en HTML el código con el formato de colores definidos en el enunciado del proyecto. Las palabras reservadas se despliegan en **negrita**, los literales se toman de color azul.

- **Program**: Program es la base principal del html, y está dado por las siguiente estructura.

```
<html>
  <head>
    <style>
      body {
        font-size: 1em !important;
        font-family: Courier !important;
      }
    </style>
  </head>
  <body>
    Command (Estructura)
  </body>
<html>
```

Se definen los atributos del estilo en conjunto de todo el programa en la sección de etiqueta *<style>*

- **ProgramPackage:** Tiene la misma estructura que Program. con el adicional de una instrucción antes de la Estructura de Command, esta ahora siendo PackageDeclaration, con la siguiente forma.

```
<html>
  <head>
    <style>
      body {
        font-size: 1em !important;
        font-family: Courier !important;
      }
```

```
    </style>
  </head>
  <body>
    PackageDeclaration (Estructura)
    Command (Estructura)
  </body>
<html>
```

- **PackageDeclaration:** Este es la estructura de la declaración de usos de paquetes en el programa.

```
<p style= "text-indent: 40px">
  <b>package </b>
  Identifier (Terminal)
  <b>~ </b>
  Declaration (Estructura)
  <b>;</b>
</p>
```

Además de presentar la estructura de identifier y declaration, presenta un aumento en la indentación del texto con el parámetro style.

- **Identifier:** Este es un terminal de tipo character que puede ser alfanumérico, no tiene un despliegue especial, por lo que no se utiliza negrita y atributos del documento. El equema es el siguiente.

  En esta regla del esquema no se necesita sub sub etiqueta. ya que se coloca el texto del identificador sin modificar.

- **ForDeclaration:** Es una implementación de declaration el cual es utilizada para la declaración de una variable de control del for. Su estructura es la siguiente

```
<b>for</b>
Identifier (Terminal)
```

```
<b>from</b>
Expression (Estructura)
```

- **VarDeclaration:** Este es un terminal de tipo character el cual es utilizada, la cual es utilizada para la declaración de una variable, el esquema es el siguiente.4

```
<p>
  <font color=#003399> var
    Identifier (Terminal)
  </font>
  <b> : </b>
  TypeDenoter (Estructura)
<p>
```

- **InitVarDeclaration:** Este es el esquema de tipo de declaración con un valor asociado a la variable.

```
 <p style="text-indent: 40px">
  Identifier (Terminal)
  <b>::=</b>
  Expression (Estructura)
</p>
```

En este formato, en el HTML se verá de la forma

```
foo ::= 4 + 5
```

! El color del número se realiza en la sección de Expression

- **LetExpression:** Estructura de una expresión Let

```
<p>
  <b>let</b>
  Declaration (Estructura)
  <b>in</b>
  Expression (Estructura)
</p>
```

- **IfExpression:** Estructura de un condicional IF, a continuación se mostrará cómo está implementado.

```
<p>
  <b>if</b>
    Expression
  <b>else</b>
```

```
          Expression
             <b>then</b>
          Expression
      </p>
```

- **UnaryExpression:** Estructura donde inicia un operador otra expression.

```
<p>
  <b>
  Operator (Estructura)
  </b>
  Expression (Estructura)
</p>
```

- **BinaryExpression:** Estructura donde inicia un operador otra expression.

```
<p>
  Expression (Estructura)
  <b>
    Operator (Estructura)
  </b>
  Expression  (Estructura)
</p>
```

- **EmptyExpression / EmptyCommand:** Para este terminal, se coloca en el texto html la palabra reservada “pass”, de la manera siguiente.

```
 <b> pass </b>
```

- **Otros:** Para los demás visits de la clase (Definidos en la sección *Extensión de los métodos para visualizar los árboles sintácticos abstractos, página 14*), se realiza el visit de sus componentes hijos sin agregar más código html

Cambios a las reglas sintácticas de Triangle

Dentro de los cambios realizados dentro del Paser.java a las reglas sintácticas de Triangle se encuentran:

- Single-Command:
    - Se eliminó que su primera alternativa fuera un comando vacío, ahora se utiliza pass para designar un comando vacío.
    - Se eliminó las alternativas:
        - “begin” Command “end”
        - “let” Declaration “in” single-Command
        - “if” Expression “then” single-Command “else” single-Command

        - “while” Expression “do” single-Command.
    - Se agregó:
        - “pass”
        - “loop” “while” Expression “do” Command “end”
        - “loop” “until” Expression “do” Command “end”
        - “loop” “do” Command “while” Expression “end”
        - “loop” “do” Command “until” Expression “end”
        - “loop” “for” Identifier “from” Expression “to” Expression “do” Command end
        - “loop” “for” Identifier “from” Expression “to” Expression “while” Expression do Command “end”
        - “loop” “for” Identifier “from” Expression “to” Expression “until” Expression do Command “end”
        - “let” Declaration “in” Command “end”
        - “if” Expression “then” Command “else” Command “end”
        - “choose” Expression “from” Cases “end”
    - Se conservó:
        - V-name “::=” Expression
        - Identifier “(” Actual-Parameter-Sequence “)”
- Reglas de Cases y Range:
    - Cases ::= Case [ElseCase]
    - Case ::= "when" Case-Literals "then" Command
    - ElseCase ::= "else" Command
    - Case-Literals ::= Case-Range ("|" Case-Range)*

    - Case-Range ::= Case-Literal [".." Case-Literal]
    - Case-Literal ::= Integer-Literal | Character-Literal
- Declaration:
    - Se modificó Declaration para que se leyera: Declaration ::= compound-Declaration | Declaration “;” compound-Declaration
- Se agregó una nueva regla para declarar procedimientos y funciones que sean mutuamente recursivos y declaraciones locales en compound-Declaration:
    - single-Declaration
    - "recursive" Proc-Funcs "end"
    - "private" Declaration "in" Declaration "end"
    - "par" single-Declaration ("|" single-Declaration)+ "end"
- De igual manera se agregaron las siguientes reglas a Proc-Func, esto obliga a declarar al menos dos procedimientos y funciones que sean mutuamente recursivos:
    - "proc" Identifier "(" Formal-Parameter-Sequence ")" "~" Command "end"
    - "func" Identifier "(" Formal-Parameter-Sequence ")" ":" Type-denoter "~" Expression
    - Y a Proc-Funcs ::= Proc-Func ("|" Proc-Func)+
- Se tiene que tomar en cuenta la regla single-Declaration, y ahí modificar la opción de “proc” para que se lea lo siguiente:
    - "proc" Identifier "(" Formal-Parameter-Sequence ")" "~" Command "end"

- Y de igual manera a single-Declaration agregarle la declaración de una variable inicializada:
    - "var" Identifier "::=" Expression
- En la agregación de paquetes se agregan las siguientes reglas para poder agrupar declaraciones y asignarles un nombre, además de poder usar las funciones y procedimientos declarados dentro de un paquete.
    - Se modifica Program para que se lea: Program ::= (Package-Declaration ";")* Command
    - Algunas de las reglas relativas a los packages:
        - Agregar en Package-Declaration:
            - "package" Package-Identifier "~" Declaration "end"
        - Agregar a Package-Identifier:
            - Identifier
        - Agregar a Long-Identifier:
            - [Package-Identifier "$"] Identifier
        - Modificar V-Name:
            - [Package-Identifier "$"] Var-Name
        - Cambiar Var-name, esto dentro del código se encuentra como un método en el Parser.java llamado “parseRestOfVName”:
            - Identifier
            - Var-name "." Identifier
            - Var-name "[" Expression "]"
        - La definición de Type-denoter se modifica:
            - Long-Identifier

        - "array" Integer-Literal "of" Type-denoter
        - "record" Record-Type-denoter "end"
    - Dentro del Single-Command y en primary-Expression se modifica la siguiente línea: Identifier "(" Actual-Parameter-Sequence ")" , el Identifier por un Long-Identifier.

Nuevas rutinas de reconocimiento sintáctico o modificaciones

Dentro de los métodos que se agregaron a la clase Parser.java se encuentran los siguientes:

- El método parseProgram se modificó para que se admitieran packages.
- Dentro del método parseSingleCommand se agregó dentró del switch:
    - El case cuando el Token es pass
    - Dentro del case Token.Identifier los casos donde reciba un Token.Dot o Token.LBracket:
    - Se agregó el case del Token.Loop y dentro de ese case un switch con los cases de Token.While, Token.Until, Token.do, Token.For:
    - Dentro del case Token.Let se cambió de parseSingleCommand a parseCommand:
    - Dentro del case Token.If se cambió de parseSingleCommand a parseCommand:
    - Se agregó el case Token.Choose:
    - Se eliminaron los cases de Token.While, Token.Semicolon, Token.End, Token.Else, Token.In, Token.Eot.
- Se agregó el método parseForCommand.

- Se agregó el método PaseDoCommand.
- Se añadió el método parseCases.
- Se añadió el método parseCase.
- Se añadió el método parseCaseLiterals.
- Se añadió el método parseCaseRange.
- Se añadió el método parseCaseLiteral.
- Dentro del método parsePrimaryExpression en el case de Token.Identifier se agregaron los casos donde reciba un Token.Dot o Token.LBracket.
- Se modificó el método parseVname, se agregó un if si es un package identifier o un var-name, que en el caso del código el método se llama parseRestOfVname.
- Del método parseDeclaration se cambió en la primera declaración de single-Declaration a compound-Declaration y en la segunda declaración se cambió a parseDeclaration.
- Se creó el método parseCompoundDeclaration basado en la regla compound-Declaration.
- Dentro del singleDeclaration:
    - Se agregó un switch dentro del case Token.Var, para agregar la regla nueva de Token.Initialize.
    - En el case Token.Proc se cambió de single-command a command y se agregó el token end al final de la declaración.
- El método con nombre parseRestOfVname, como se mencionó anteriormente cumple la regla Var-name.

- Se crearon los métodos parsePackageDeclaration y el parseLongIdentifier, según las reglas indicadas.

Errores sintácticos detectados

- ChooseCommand

Árbol sintáctico

Se agregaron las siguientes clases:

CaseCommand

- Case
- ElseCase
- SequentialCase

CaseLiteralAST

- CaseCharacterLiteral
- CaseIntegerLiteral
- CaseLiterals

CaseRange

- DualRange
- SingleRange
- SequentialCaseRange
- SequentialRange

Command

- ChooseCommand
- DoUntilCommand

- DoWhileCommand
- ForCommand
- ForUntilCommand
- ForWhileCommand
- UntilCommand

Declaration

- ForDeclaration
- InitVarDeclaration
- PackageDeclaration
- PrivateDeclaration
- SequentialPackageDeclaration

LongIdentifier

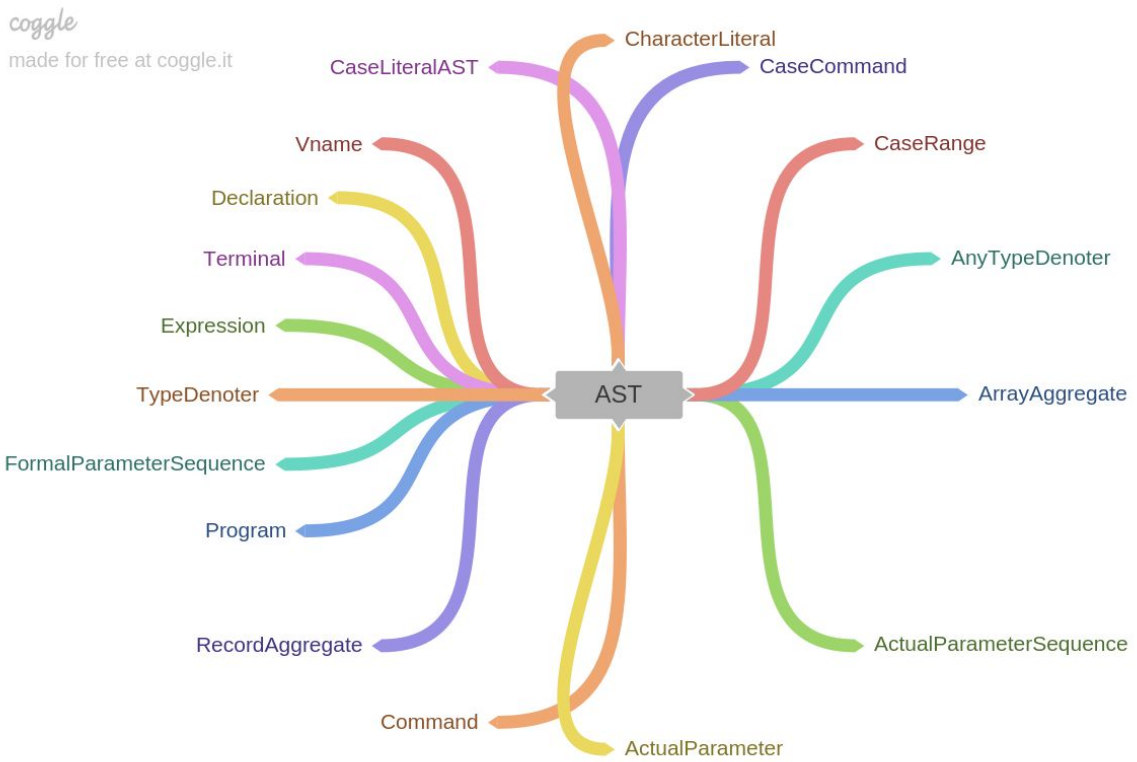
- PackageId
- SimpleIdentifier

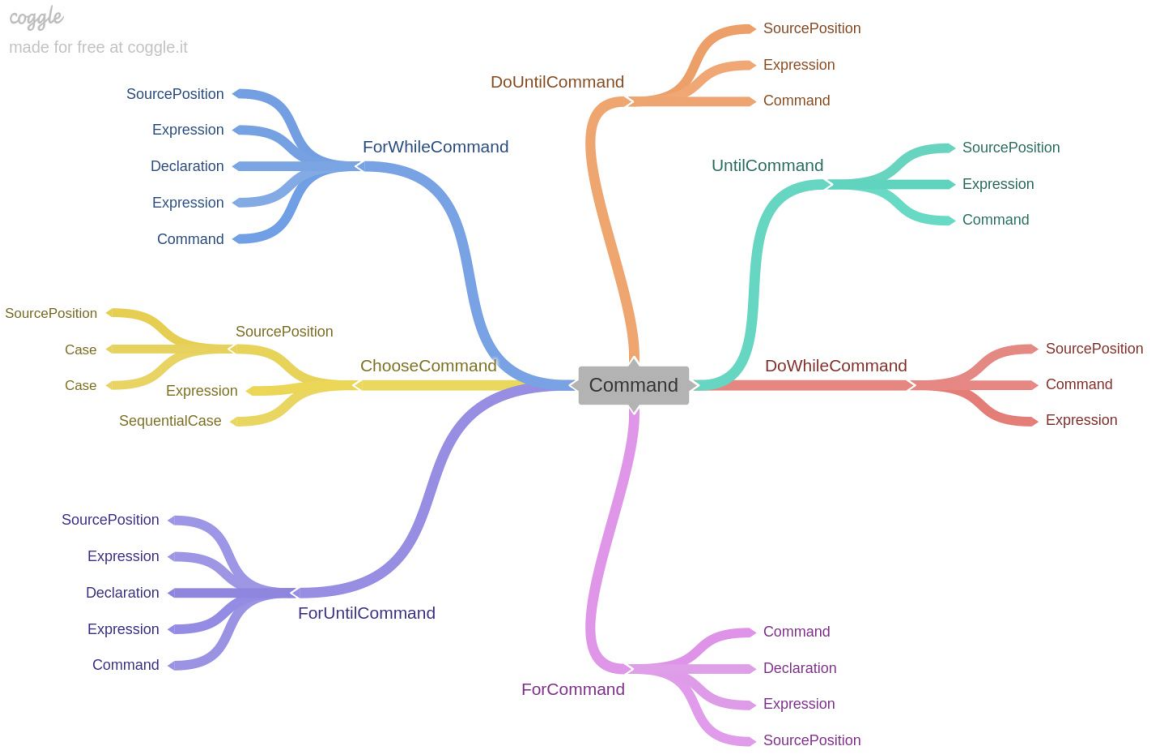
Vname

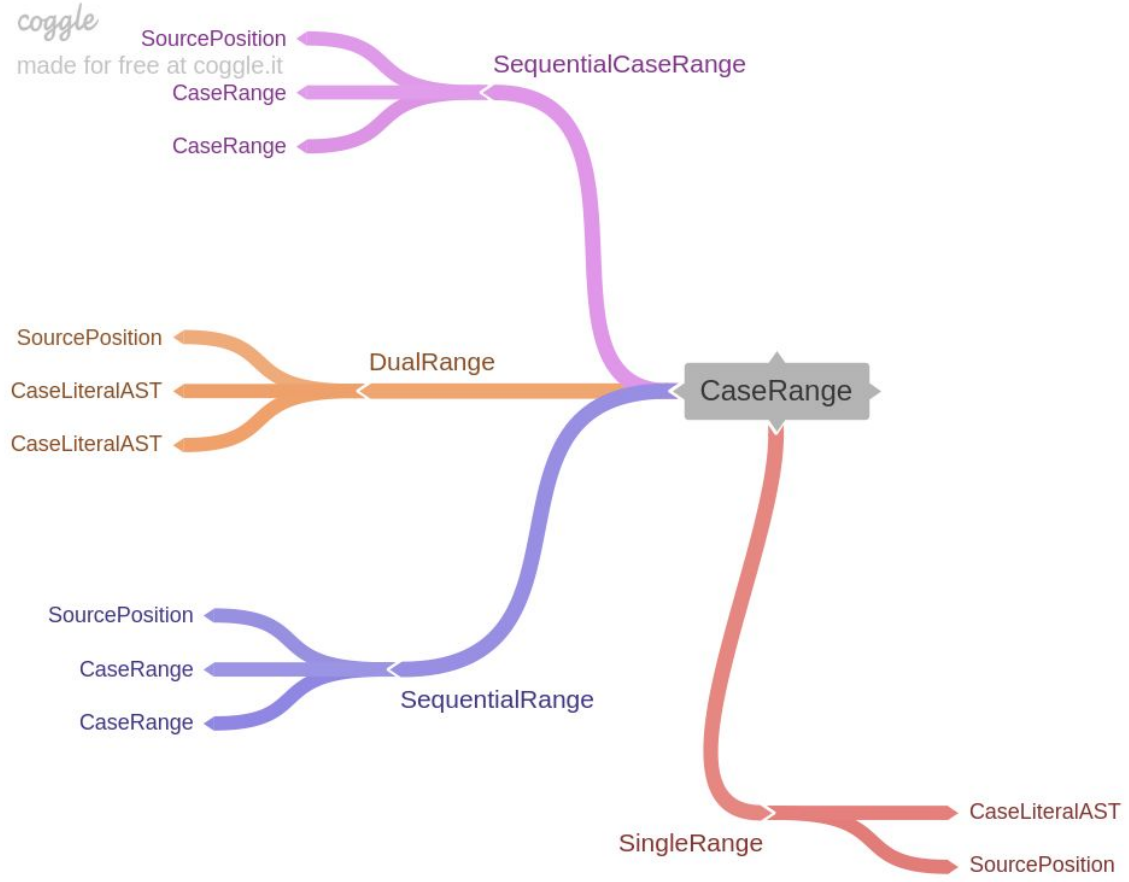
- PackageVName

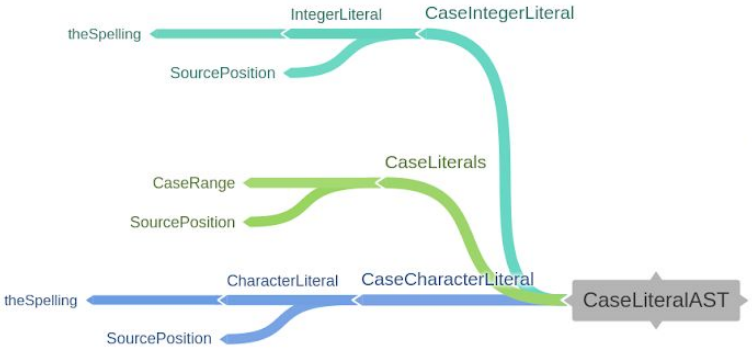
Program

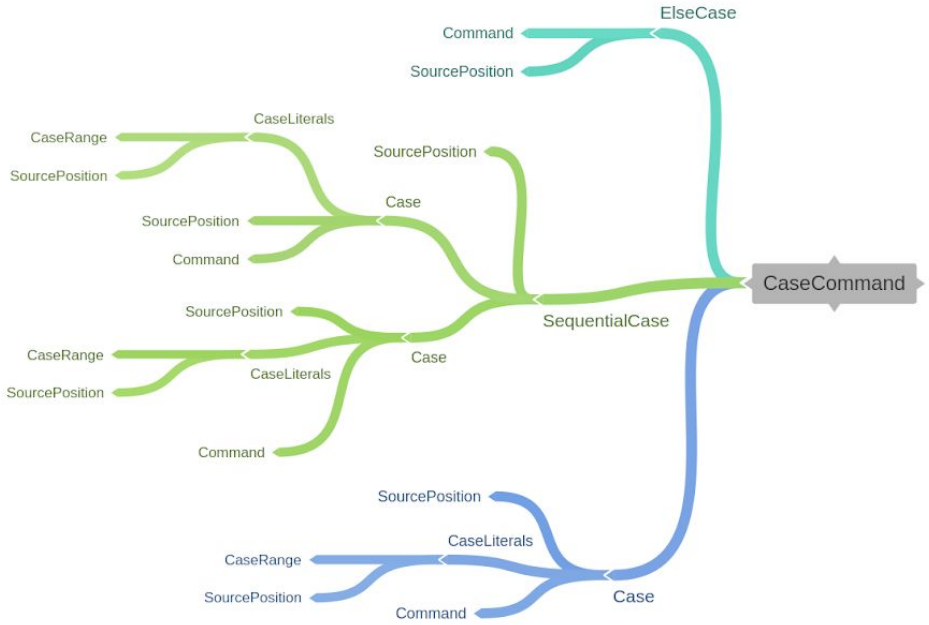
- ProgramPackage

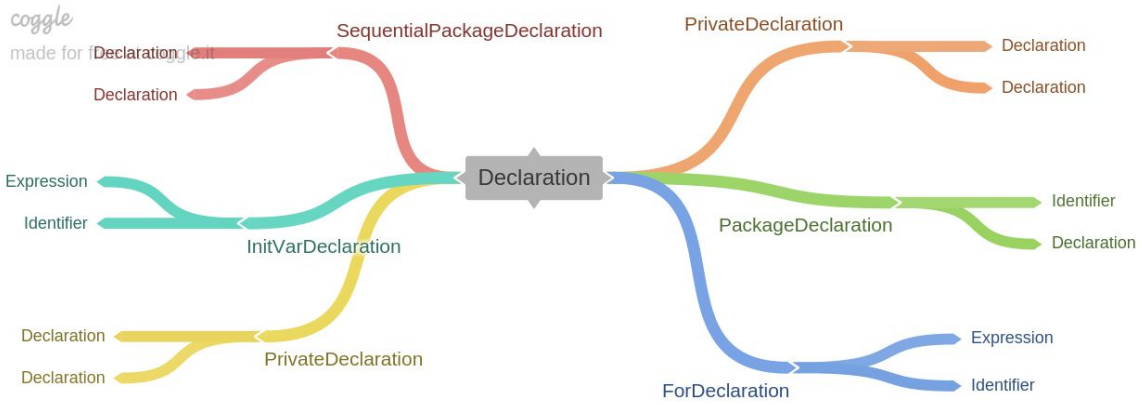

coggle
made for free at coggle.it
AST
CharacterLiteral
CaseLiteralAST
CaseCommand
Vname
CaseRange
Declaration
Terminal
AnyTypeDenoter
Expression
TypeDenoter
ArrayAggregate
FormalParameterSequence
Program
RecordAggregate
ActualParameterSequence
Command
ActualParameter

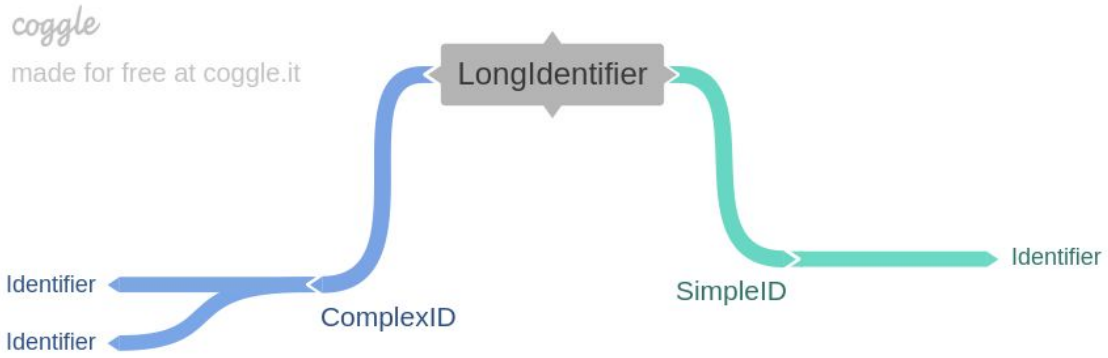

coggle
made for free at coggle.it
Command
DoUntilCommand
SourcePosition
Expression
Command
ForWhileCommand
SourcePosition
Expression
Declaration
Expression
Command
UntilCommand
SourcePosition
Expression
Command
ChooseCommand
SourcePosition
Case
Case
Expression
SequentialCase
DoWhileCommand
SourcePosition
Command
Expression
ForUntilCommand
SourcePosition
Expression
Declaration
Expression
Command
ForCommand
Command
Declaration
Expression
SourcePosition

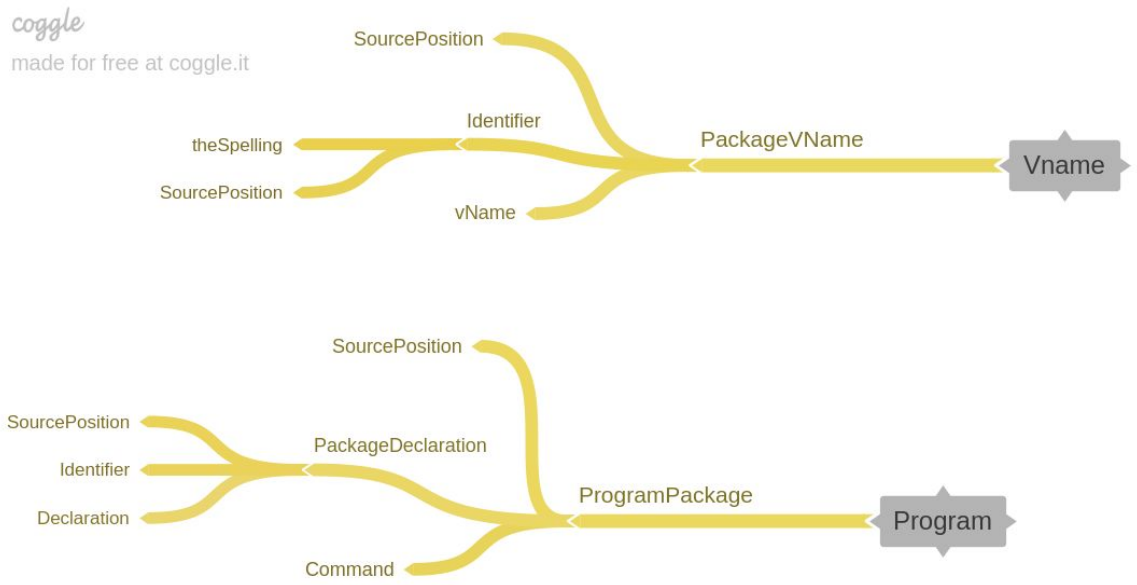





ElseCase
Command
SourcePosition
CaseLiterals
CaseRange
SourcePosition
SourcePosition
Case
SourcePosition
Command
CaseCommand
SequentialCase
SourcePosition
Case
CaseRange
CaseLiterals
SourcePosition
Command
SourcePosition
CaseLiterals
CaseRange
SourcePosition
Case
Command


coggle
made for free at coggle.it
SequentialPackageDeclaration
Declaration
Declaration
PrivateDeclaration
Declaration
Declaration
Expression
Identifier
InitVarDeclaration
Declaration
Declaration
PrivateDeclaration
Declaration
PackageDeclaration
Identifier
Declaration
ForDeclaration
Expression
Identifier


coggle
made for free at coggle.it
LongIdentifier
Identifier
Identifier
ComplexID
SimpleID
Identifier



Extensión de los métodos para visualizar los árboles sintácticos abstractos

Se agregaron los siguientes métodos visit a la interface Visitor.java:

- visitCase
- visitCaseLiterals
- visitCaseIntegerLiteral
- visitCaseCharacterLiteral
- DoUntilCommand
- DoWhileCommand
- ForCommand
- ForDeclaration
- ForUntilCommand
- ForWhileCommand
- SequentialCase
- SequentialRange
- SingleRange

- UntilCommand
- PrivateDeclaration
- DualRange
- ElseCase
- ChooseCommand
- SequentialPackageDeclaration
- PackageDeclaration
- PackageVName
- PackageId
- LongIdentifier

Extensión de los métodos para representar los árboles sintácticos abstractos en XML

En la creación del archivo XML la clase extiende de visitor y consiste en visitar cada estructura encontrada en el AST y abrir brackets para etiquetar las partes hasta llegar a un “leaf” para después empezar a cerrar los mismos brackets. Luego este escribe todo el texto sobre un archivo con extensión .xml al compilar los programas .tri. Los métodos agregados para la creación del archivo XML para la sintaxis nueva de triangle son los siguientes:

- **visitProgramPackage:**

```
writeLineHTML("<ProgramPackage>");
ast.C.visit(this, null);
ast.P.visit(this, null);
writeLineHTML("</ProgramPackage>");
```

- **visitPackageVName:**

writeLineHTML("<PackageVName>");

ast.V.visit(this, null);

ast.I.visit(this, null);

writeLineHTML("</PackageVName>");

- **visitCase:**

  writeLineHTML("<Case>");

  ast.C1.visit(this, null);

  ast.C2.visit(this,null);

  writeLineHTML("</Case>");

- **visitSequentialCase:**

  writeLineHTML("<SequentialCase>");

  ast.C1.visit(this, null);

  ast.C2.visit(this,null);

  writeLineHTML("</SequentialCase>");

- **visitSequentialRange:**

  writeLineHTML("<SequentialRange>");

  ast.R1.visit(this, null);

  ast.R2.visit(this,null);

  writeLineHTML("</SequentialRange>");

- **visitElseCase**

  writeLineHTML("<ElseCase>");

  ast.C.visit(this, null);

  writeLineHTML("</ElseCase>");

- **visitAssignCommand**

```
writeLineHTML("<AssignCommand>");
ast.V.visit(this, null);
ast.E.visit(this, null);
writeLineHTML("</AssignCommand>");
```

- **visitIfCommand**

```
writeLineHTML("<IfCommand>");
ast.E.visit(this, null);
ast.C1.visit(this, null);
ast.C2.visit(this, null);
writeLineHTML("</IfCommand>");
```

- **visitWhileCommand**

```
writeLineHTML("<WhileCommand>");
ast.E.visit(this, null);
ast.C.visit(this, null);
writeLineHTML("</WhileCommand>");
```

- **visitDoUntilCommand**

```
writeLineHTML("<DoUntilCommand>");
ast.C.visit(this, null);
ast.E.visit(this, null);
writeLineHTML("</DoUntilCommand>");
```

- **visitDoWhileCommand**

```
writeLineHTML("<DoWhileCommand>");
ast.C.visit(this, null);
ast.E.visit(this, null);
```

```
writeLineHTML("</DoWhileCommand>");
```

- **visitForCommand**

```
writeLineHTML("<ForCommand>");
ast.C.visit(this, null);
ast.E.visit(this, null);
ast.D.visit(this, null);
writeLineHTML("</ForCommand>");
```

- **visitForUntilCommand**

```
writeLineHTML("<ForUntilCommand>");
ast.C.visit(this, null);
ast.D.visit(this, null);
ast.E1.visit(this, null);
ast.E2.visit(this, null);
writeLineHTML("</ForUntilCommand>");
```

- **visitForWhileCommand**

```
writeLineHTML("<ForWhileCommand>");
ast.C.visit(this, null);
ast.D.visit(this, null);
ast.E1.visit(this, null);
ast.E2.visit(this, null);
writeLineHTML("</ForWhileCommand>");
```

- **visitUntilCommand**

```
writeLineHTML("<UntilCommand>");
ast.C.visit(this, null);
```

```
ast.E.visit(this,null);

writeLineHTML("</UntilCommand>");
```

- **visitChooseCommand**

```
writeLineHTML("<ChooseCommand>");

ast.C.visit(this, null);

ast.E.visit(this,null);

writeLineHTML("</ChooseCommand>");
```

Plan de pruebas

Prueba	Objetivo del caso de prueba	Diseño del caso de prueba	Resultados esperados	Resultados observados

Begin	Probar que begin ya no este como una palabra reservada	let
		  var begin: Integer
		in
		  putint (begin);
		  puteol ()
		end
			Se pueda realizar correctamente ya que begin no es una palabra reservada	Se observó el resultado deseado

Choose 1	Comprueba que se revise bien la sintaxis del choose	choose 1 from when then putint (1) end	Salga un error debido a la falta del literal	No funcionó debido a un problema encontrado en el parser con los Abstract Trees
Choose 2	Comprueba que se revise bien la sintaxis del choose	choose 1 from when 5 6 then putint (1) end	Salga un error por la falta del separador entre los literales	No funcionó debido a un problema encontrado en el parser con los Abstract

				Trees
Choose 3	Comprueba que se revise bien la sintaxis del choose	choose 1 from when 5 | 6 then putint (1) when 7 then putint(7) else putint (0) else end	Salga un error por el último else	No funcionó debido a un problema encontrado en el parser con los Abstract Trees
Choose 4	Comprueba que se revise bien la sintaxis del choose	choose from when 5 | 6 then putint (1) when 7 then putint(7) else putint (0) end	Salga un error debido a que no hay una expresión luego del choose	No funcionó debido a un problema encontrado en el parser con los Abstract Trees
Choose 5	Comprueba que se revise bien la sintaxis del choose	choose 1 from when 0 then putint (1) else putint (1) end --- choose 1 from when 0 | 1 then putint (1) when 2 then putint(2) end	En ambos casos está correcto, por lo que no debería de aparecer ningún error	No funcionó debido a un problema encontrado en el parser con los Abstract Trees
Choose 6	Comprueba que se revise bien la sintaxis del choose	choose 1 from when 0 | 1 then putint (1) when 2 then putint(2) else putint (5) end --- choose 0 from when 1|2|3 then putint (123) when 4 then choose '1' from when '1' then putint (1) else putint (2) end end	En ambos casos está correcto, por lo que no debería de aparecer ningún error	No funcionó debido a un problema encontrado en el parser con los Abstract Trees

Choose 7	Comprueba que se revise bien la sintaxis del choose	choose 1 from
		  when 0 .. 5 then putint (1)
		  when 7 then putint(2)
		end
		---------- 	
		choose 1 from
		  when 0 .. 5|8 then putint (1)
		  when 7 then putint(2)
		end
			En ambos casos está correcto, por lo que no debería de aparecer ningún error	No funcionó debido a un problema encontrado en el parser con los Abstract Trees

If 1	Comprueba la sintaxis del if	if (1=1) puteol() else puteol() end	Muestra un error ya que falta el end y usa ()	Se observó el resultado deseado
If 2	Comprueba la sintaxis del if	if 1=1 then puteol() end	Muestra un error ya que hace falta el else	Se observó el resultado deseado
If 3	Comprueba la sintaxis del if	if 1=1 do puteol() else puteol() end	Muestra un error, ya que se escribió do en lugar de then	Se observó el resultado deseado
if 4	Comprueba la sintaxis del if	if 1=1 then puteol() else puteol()	Muestra un error ya que no se puso el end	Se observó el resultado deseado
If 5	Comprueba la sintaxis del if	if 1=1 then puteol() else puteol() end	No debe mostrar ningún error ya que está correcto	Se observó el resultado deseado

Let 1	Comprueba la sintaxis del let	let var i : Char;
		var j : Integer;
		  const k ::= 8;
		  type dia ~ Integer
		  put(i);
		  put(i)
		end
			Debe mostrar un error ya que hace falta el “in”	Se observó el resultado deseado

Let 2	Comprueba la sintaxis del let	let in
		  put(i);
		  put(i)
		end
			Muestra un error ya que tiene una declaración vacía	Se observó el resultado deseado

Let 3	Comprueba la sintaxis del	let var i : Char;	Muestra un error ya que hace falta	Se observó el resultado

	let	var j : Integer; const k ~ 8; type dia ~ Integer in end	los comandos	deseado

Let 4	Comprueba la sintaxis del let	let var i : Char;
		var j : Integer;
		const k ::= 8;
		type dia ~ Integer
		
		in
		put(i);
		put(i)
			Muestra un error porque falta el end	Se observó el resultado deseado

Let 5	Comprueba la sintaxis del let	let var i : Char; var j : Integer; const k ~ 8; type dia ~ Integer in put(i); put(i) end	No muestra ningún error ya que está correcto	Se observó el resultado deseado
Do Until 1	Comprueba la sintaxis del do until	do pass until 4=4 end	Muestra un error ya que falta el loop	Se observó el resultado deseado
Do Until 2	Comprueba la sintaxis del do until	loop do pass;pass until i < 5	Muestra un error ya que falta el end	Se observó el resultado deseado
Do Until 3	Comprueba la sintaxis del do until	loop pass until 4=4 end	Muestra un error ya que falta el do	Se observó el resultado deseado
Do Until 4	Comprueba la sintaxis del do until	loop do pass until 4=4 end ---------- loop do pass;pass until 4=4 end	No muestran un error ya que están correctos	Se observó el resultado deseado
Do While 1	Comprueba la sintaxis del do while	loop do do pass;pass while 3=3 end	Muestra un error ya que hay 2 do	Se observó el resultado deseado
Do	Comprueba	loop pass while 3=3 end	Muestra un error	Se observó

While 2	la sintaxis del do while		por que no está el do	el resultado deseado
Do While 3	Comprueba la sintaxis del do while	loop do pass while 3=3 end --- loop do pass;pass while 3=3 end	No muestran un error ya que están correctos	Se observó el resultado deseado
For 1	Comprueba la sintaxis del for	loop i from 5 to 6 do pass;pass end	Muestra un error ya que falta el for	Se observó el resultado deseado
For 2	Comprueba la sintaxis del for	loop for i 5 to 6 do pass;pass end	Muestra un error ya que falta el from	Se observó el resultado deseado
For 3	Comprueba la sintaxis del for	loop for i from 5 6 do pass;pass end	Muestra un error ya que falta el to	Se observó el resultado deseado
For 4	Comprueba la sintaxis del for	loop for i from 5 to 6 pass;pass end	Muestra un error ya que falta el do	Se observó el resultado deseado
For 5	Comprueba la sintaxis del for	loop for i from 5 to 6 do pass;pass	Muestra un error ya que falta el end	Se observó el resultado deseado
For 6	Comprueba la sintaxis del for	loop for i from 5 to 6 do pass end --- loop for i from 5 to 6 do pass;pass end	No muestran un error ya que están correctos	Se observó el resultado deseado
For Until 1	Comprueba la sintaxis del for until	loop for i from 5 to 6 i < 5 do pass;pass end	Muestra un error ya que falta until	Se observó el resultado deseado
For Until 2	Comprueba la sintaxis del for until	loop for i from 5 to 6 until i < 5 pass;pass end	Muestra un error ya que falta el do	Se observó el resultado deseado
For Until 3	Comprueba la sintaxis del for until	loop for i from 5 to 6 until i < 5 do pass end --- loop for i from 5 to 6 until i < 5 do pass;pass end	No muestran un error ya que están correctos	Se observó el resultado deseado

For While 1	Comprueba la sintaxis del for while	loop for i from 5 to 6 i < 5 do pass;pass end	Muestra un error ya que falta while	Se observó el resultado deseado
For While 2	Comprueba la sintaxis del for while	loop for i from 5 to 6 while i < 5 pass;pass end	Muestra un error ya que falta el do	Se observó el resultado deseado
For While 3	Comprueba la sintaxis del for while	loop for i from 5 to 6 while i < 5 do pass;pass	Muestra un error ya que falta el end	Se observó el resultado deseado
For While 4	Comprueba la sintaxis del for while	loop for i from 5 to 6 while i < 5 do pass end --- loop for i from 5 to 6 while i < 5 do pass;pass end	No muestran un error ya que están correctos	Se observó el resultado deseado
Until do 1	Comprueba la sintaxis del until do	until 2=2 do pass;pass end	Muestra un error ya que falta el loop	Se observó el resultado deseado
Until do 2	Comprueba la sintaxis del until do	loop until 2=2 pass; pass end	Muestra un error ya que falta el do	Se observó el resultado deseado
Until do 3	Comprueba la sintaxis del until do	loop until 2=2 do pass end --- loop until 2=2 do pass; pass end	No muestran un error ya que están correctos	Se observó el resultado deseado
While do 1	Comprueba la sintaxis del while do	while 1=1 do pass; pass end	Muestra un error ya que esta es la sintaxis vieja	Se observó el resultado deseado
While do 2	Comprueba la sintaxis del while do	loop while 1=1 pass; pass end	Muestra un error ya que falta el do	Se observó el resultado deseado
While do 3	Comprueba la sintaxis del while do	loop while 1=1 do pass end --- loop while 1=1 do pass; pass end	No muestran un error ya que están correctos	Se observó el resultado deseado
Packag e 1	Comprueba la sintaxis del package	myP ~ var a: Integer end; let	Muestra un error ya que no tiene el package	Se observó el resultado deseado

		var b: Integer; b:= myP $ a in pass end		
Package 2	Comprueba la sintaxis del package	package myP var a: Integer end; let var b: Integer; b:= myP $ a in pass end	Muestra un error ya que hace falta el ~	Se observó el resultado deseado
Package 3	Comprueba la sintaxis del package	package myP ~ var a: Integer ; let var b: Integer; b:= myP $ a in pass end	Muestra un error ya que hace falta el end del package	Se observó el resultado deseado
Package 4	Comprueba la sintaxis del package	package myP ~ var a: Integer end; let var b: Integer in putint (myP $ a) end	No muestra un error ya que está correcto	Se observó el resultado deseado

Package Invocation 1	Comprueba la sintaxis del package invocation	package myP ~
		var a: Integer
		end;
		let
		var b: Integer
		in
		b:= myP a;
		putint(b)
		
		end
			Muestra un error ya que hace falta el $	Se observó el resultado deseado

Package Invocation 2	Comprueba la sintaxis del package invocation	package myP ~
		var a: Integer
		end;
		let
		var b: Integer
		in
		b:= myP $ a;
		putint(b)
		
		end
			No muestra un error ya que está correcto	Se observó el resultado deseado

Par 1	Comprueba la sintaxis del par	let var i: Integer | var k: Integer end in putint(i); put(i) end	Muestra un error ya que falta el par	Se observó el resultado deseado
Par 2	Comprueba la sintaxis del par	let par var i: Integer var k: Integer end in put(i); put(i) end	Muestra un error ya que falta el |	Se observó el resultado deseado
Par 3	Comprueba la sintaxis del par	let par var i: Integer | var k: Integer in put(i); putint(k) end	Muestra un error porque falta el end	Se observó el resultado deseado
Par 4	Comprueba la sintaxis del par	let par var i: Integer | var k: Integer | var t: Char end in putint(i); put(t) end --- let par var i: Integer | var k: Integer | var t: Char end	No muestran un error ya que están correctos	Se observó el resultado deseado

		in putint(i); put(t) end		
Pass 1	Comprueba la sintaxis del pass	puteol();	Muestra un error ya que después del ; falta un pass	Se observó el resultado deseado
Pass 2	Comprueba la sintaxis del pass	pass ---------- pass; puteol(); pass	No muestran un error ya que están correctos	Se observó el resultado deseado

Private 1	Comprueba la sintaxis del private	let
		private
		var a::=5;
		var b::=15
		! in
		var c::=a*b*2
		end ;
		var d::= 5
		in
		putint(c)
		end
			Muestra un error ya que hace falta el in del private	Se observó el resultado deseado

Private 2	Comprueba la sintaxis del private	let private var a::=5; var b::=15 in var c::=a*b*2 ! end ; var d::= 5 in putint(c) end	Muestra un error ya que falta el end del private	Se observó el resultado deseado
Private 3	Comprueba la sintaxis del private	let private ! var a::=5; ! var b::=15 in var c::=a*b*2	Muestra un error ya que falta la declaración local en el private	Se observó el resultado deseado

		end; var d::= 5 in putint(c) end		

Private 4	Comprueba la sintaxis del private	let
		private
		var a::=5;
		var b::=15
		in
		! var c::=a*b*2
		end;
		var d::= 5
		in
		putint(c)
		end
			Muestra un error ya que falta la declaración en private que usa la local	Se observó el resultado deseado

Private 5	Comprueba la sintaxis del private	let
		private
		var a::=5;
		var b::=15
		in
		var c::=a*b*2
		end;
		var d::= 5
		in
		putint(c)
		end
			No muestra un error ya que está correcto	Se observó el resultado deseado

Recursive 1	Comprueba la sintaxis de recursive	let
		  recursive
		  proc doble (var i : Integer) ~
		  i := i*2
		  end
		  end
		in
		  pass
		end
		--- 	
		let
		  recursive
		  func doble (var i : Integer) : Integer ~
		  i*2
		  end
		in
		  pass
		end
			Muestran un error porque solo existe una alternativa	Se observó el resultado deseado

Recursive 2	Comprueba la sintaxis de recursive	let
		  recursive
		  func doble(var i : Integer) : Integer ~
		  i*2
		  |
		  var i : Integer
		  end
		in
		  pass
		end
			Muestra un error ya que una de las alternativas no es proc o func	Se observó el resultado deseado

Recursive 3	Comprueba la sintaxis de recursive	let
		  recursive
		
		  end
		in
		  pass
		end
			Muestra un error porque no hay alternativas	Se observó el resultado deseado

Recursive 4	Comprueba la sintaxis de recursive	let
		  recursive
		  func doble(var i : Integer) : Integer ~
			Muestra un error ya que hace falta el | que separa los procs-funcs	Se observó el resultado deseado

		triplicar(i)/3*2 proc triplicar(var i : Integer) ~ i ::= i + doble(i) end end in pass end		
Recursive 5	Comprueba la sintaxis de recursive	let recursive func doble(var i : Integer) : Integer ~ triplicar(i)/3*2 | proc triplicar(var i : Integer) ~ i := i + doble(i) end in pass end	Muestra un error ya que hace falta el end del recursive	Se observó el resultado deseado
Recursive 6	Comprueba la sintaxis de recursive	let recursive func doble(var i : Integer) : Integer ~ triplicar(i)/3*2 | proc triplicar(var i : Integer) ~ i := i + doble(i) end | proc duplicar(var i : Integer) ~ i := doble(i) end	No muestra un error ya que está correcto	Se observó el resultado deseado

		end in pass end		
Var Init 1	Comprueba la sintaxis de var init	var j::=5	Muestra un error ya que no se puede declarar afuera de un let o un local	Se observó el resultado deseado
Var Init 2	Comprueba la sintaxis de var init	let var j ::= 'c'; var i ::= 4 : Integer; var y : Integer in pass end	Muestra un error porque i no debería de tener una declaración de tipo	Se observó el resultado deseado
Var Init 3	Comprueba la sintaxis de var init	let var j ::= 'c'; var i : Integer ::= 0; var y : Integer in pass end	Muestra un error ya que se inicializa después de denotar el tipo	Se observó el resultado deseado
Var Init 4	Comprueba la sintaxis de var init	let var j := 'c'; var i : Integer; var y : Integer in pass end	Muestra un error ya que se usó := en vez de ::=	Se observó el resultado deseado
Var Init 5	Comprueba la sintaxis de var init	let var j ::= 5; var i ::= 'c'; var y : Integer in pass end	No muestra un error ya que está correcto	Se observó el resultado deseado

Análisis de la cobertura del plan de pruebas

Después de ejecutar todos los códigos de prueba que brindó el profesor en los recursos estos los usamos como base para el plan de pruebas. Fueron muchas pruebas diferentes todos con un objetivo diferente, en el caso de los que tiran error a propósito es para observar que se atrapan bien los incoherencias sintácticas además de que el código eliminado como los begin fueron correctamente re implementados como deben. Con respecto al código “ok” el objetivo es ver que todos los códigos que siguen las reglas sintácticas logren pasar por el compilador correctamente. Este plan de pruebas también tiene como objetivo en el código que si compila poder observar bien la creación de los archivos HTML y XML que son parte del proceso de compilación.

Discusión y análisis sobre los resultados

Al terminar todas las pruebas y debugeo correspondiente logramos obtener un gran margen de éxito, sin embargo si existen errores encontrados en los resultados que requieren de mucho más tiempo y cambios serios dentro del various packages del código java, tanto así que no se logró cambiar estos errores antes de la fecha de entrega. Durante la compilación de los archivos con código de “choose” se genera un error a nivel del código de Java que interpreta el lenguaje Triangle. El error existe en las clases y métodos que generamos para organizar y parsear los comandos de “choose”. Además de este error todos los demas pruebas en el plan fueron exitosos y se obtuvieron los resultados deseados.

Reflexión

Haber realizado las modificaciones a una parte de un compilador/ambiente que fue escrito por terceras personas nos ayudó mucho a comprender la parte teórica que fue vista en clase, ya que es muy diferente ver la materia en una presentación o en la pizarra, a tener que crear nosotros todos los cambios. Para poder realizar de manera exitosa los cambios solicitados se revisó la teoría del libro de Watt y Brown para observar la gramática original de Triangle y saber exactamente a donde y como se debían de realizar los cambios solicitados.

Fue una excelente experiencia para todos los integrantes del grupo, ya que ninguno antes había trabajado con o en un compilador por lo que experimentar con el código de este fue un reto, ya que se tenían que seguir las reglas dadas por la gramática que se asignó. Por lo que, al cambiar una sección del código o cambiarle el nombre a una clase podía afectar varias partes del código y este se tenía que cambiar. Gracias a los problemas que se enfrentaron a lo largo del proyecto se aprendió mucho sobre cómo funcionan los analizadores sintácticos y léxicos de los compiladores, algo que en el futuro nos va a servir en nuestra carrera profesional.

Asignación de tareas

Tareas asignadas a Gabriel Quesada:

- Parser
- TreeVisitor
- Token
- LayoutVisitor

Tareas asignadas a Josue Daniel Canales:

- Parser
- HTML y XML
- Clases AST
- Token

Tareas asignadas a Jose Fabio Perez:

- XML y HTML
- FileCreator
- Visitor
- Parser

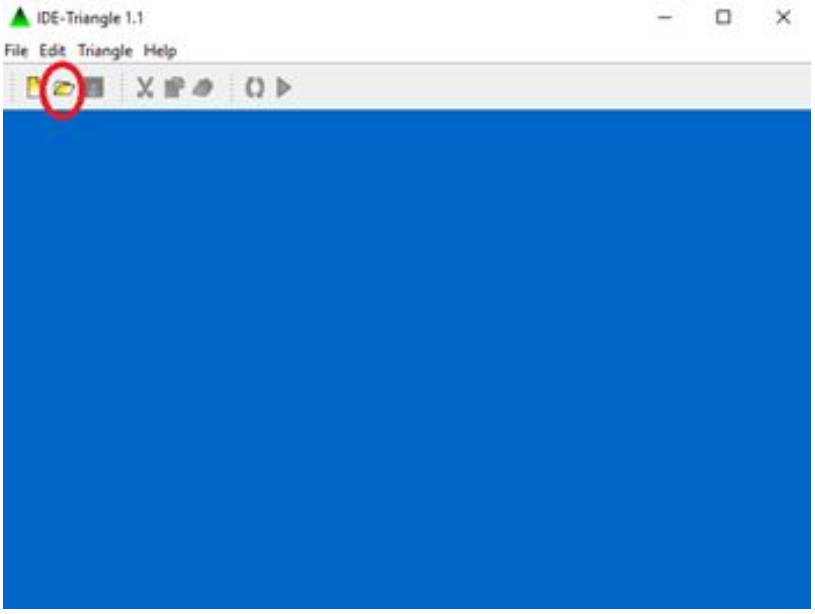
Tareas asignadas a Bárbara Gutiérrez:

- Scanner
- Clases del AST
- LayoutVisitor
- Tree y Table Visitor

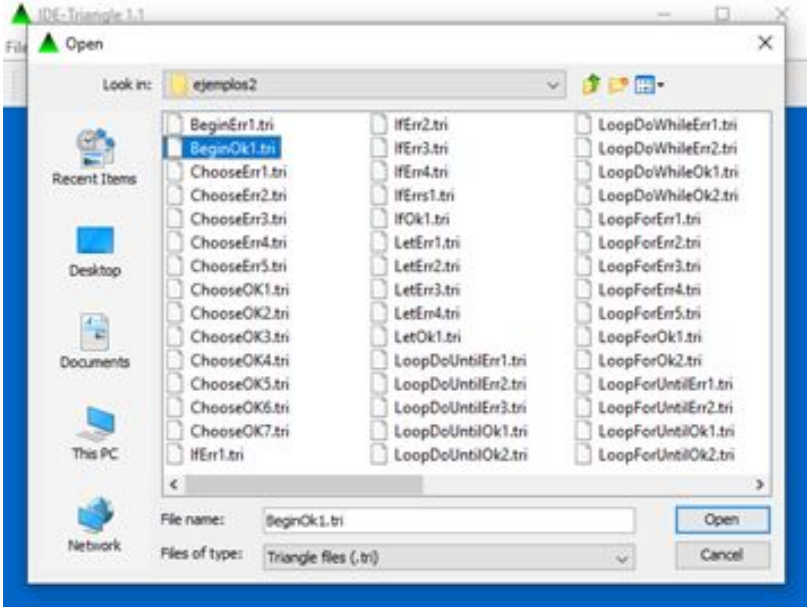
Todos los integrantes participaron en la creación de la documentación del proyecto.

Cómo debe compilar el programa

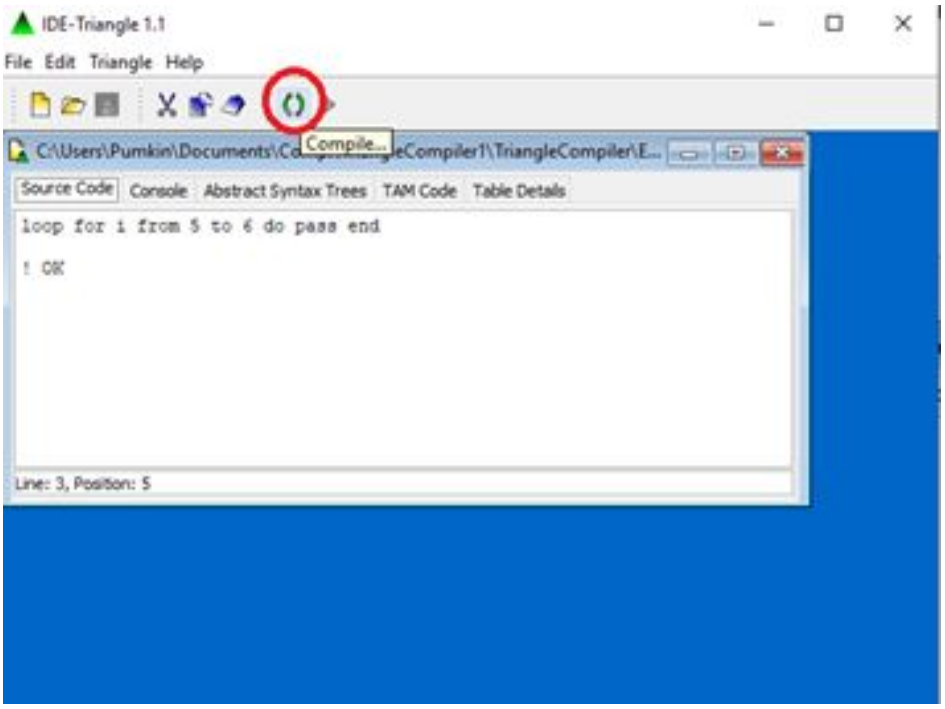
1. Al abrir el IDE de Triangle en la ventana se le da click al siguiente icono para abrir un archivo .tri.



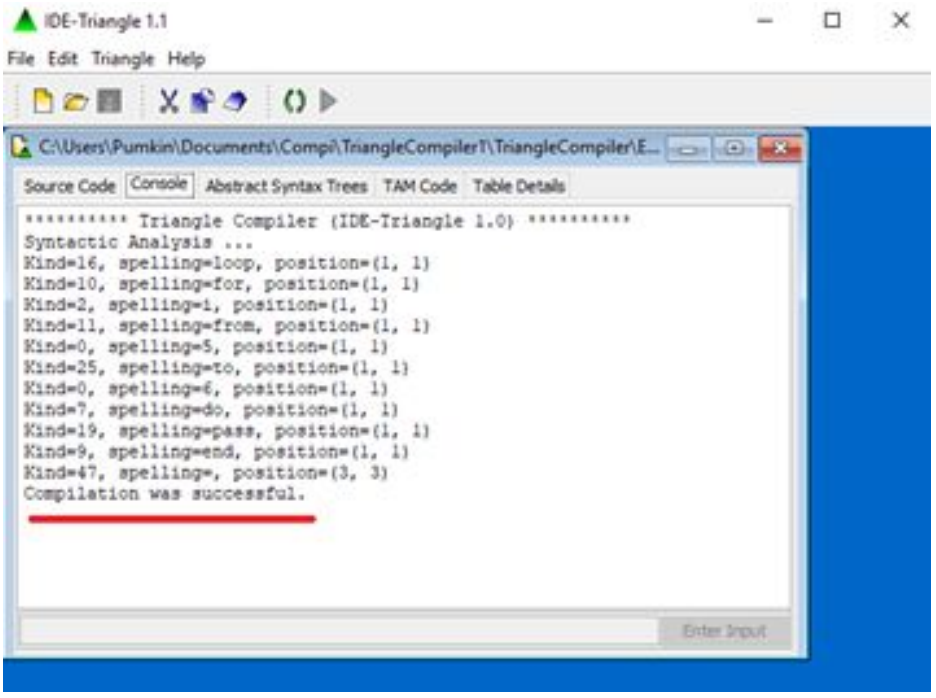
2. En la siguiente ventana se navega al folder destino del archivo, se selecciona y le da click “Open”.



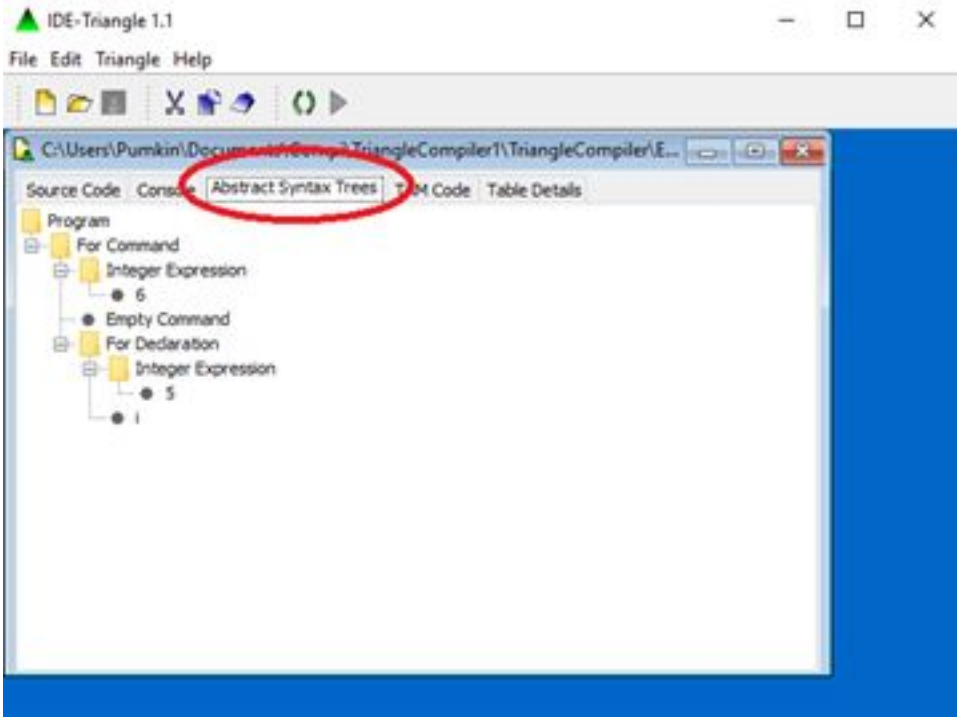
3. Luego de abrir el código se compila dándole click al siguiente icono indicado.



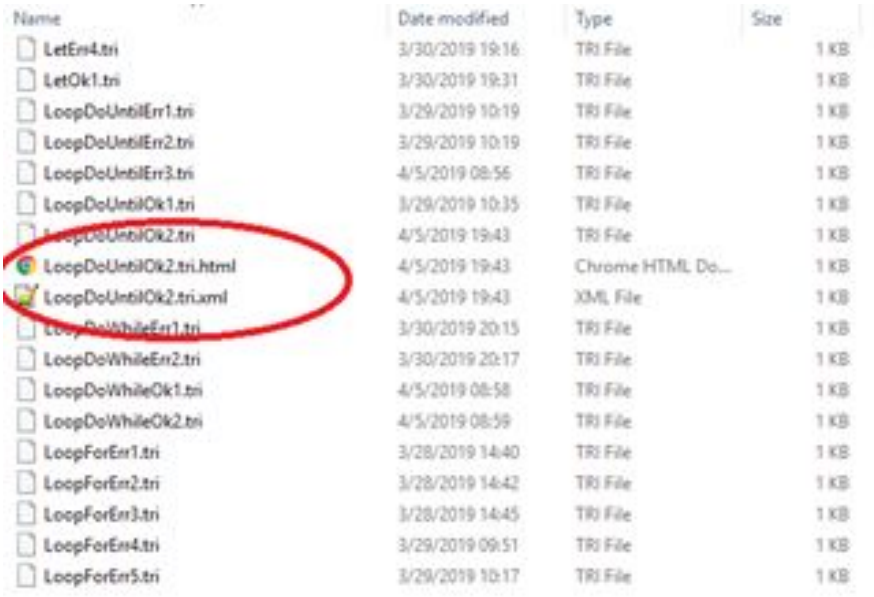
4. En el tab de “Console” del IDE veremos si la compilación fue exitosa.



5. En el tab de “Abstract Syntax Tree” se ve el desglose del árbol sintáctico del código compilado.

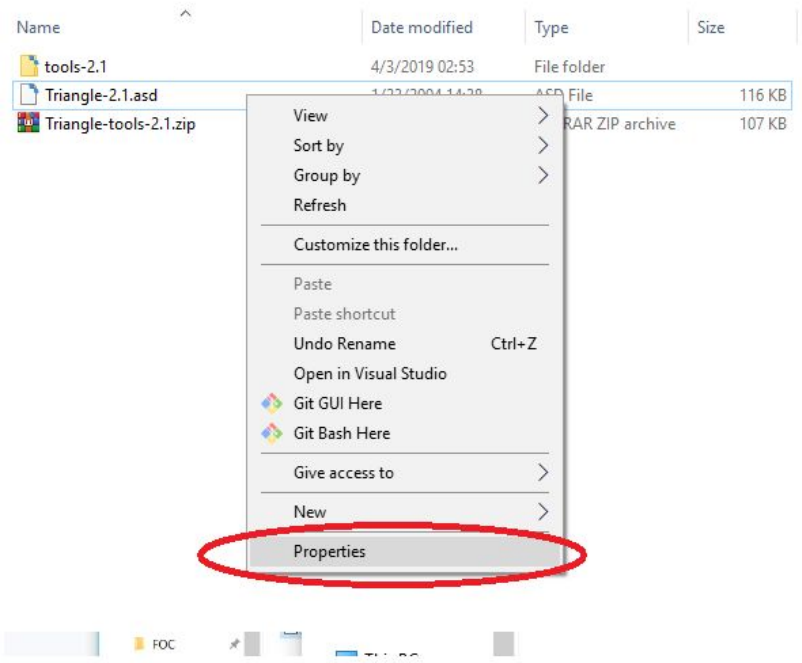


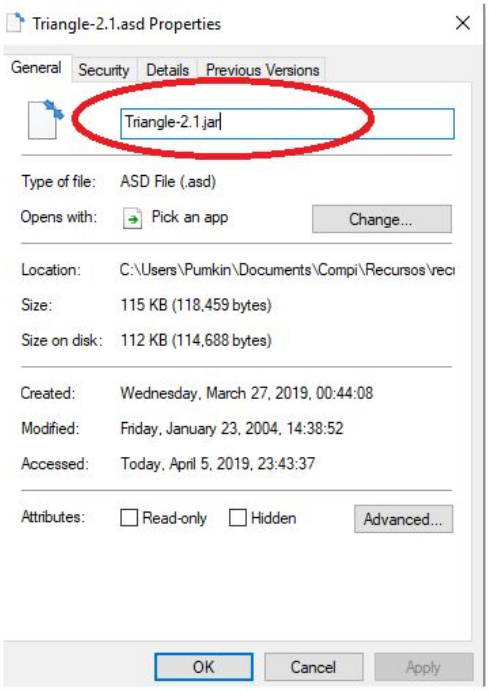
6. Después de compilar un archivo .tri se creará un archivo .XML y otro .HTML para visualizar el código y el árbol sintáctico, estos archivos tendrán el mismo nombre del .tri.



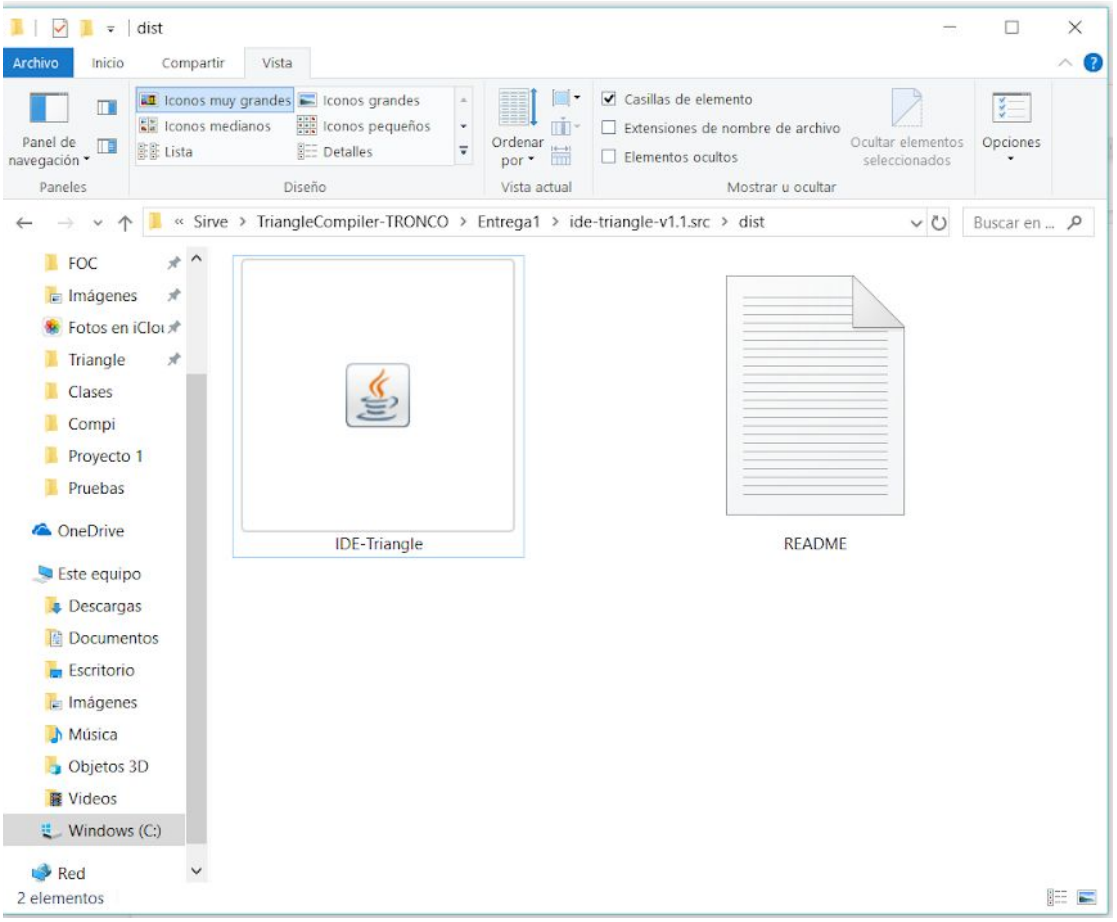
Cómo se ejecuta el programa

1. **Seleccione el .proyecto y entre a propiedades, se cambia la extensión a un .jar.**





2. **Ejecutar el .jar**


dist
Archivo
Inicio
Compartir
Vista
Panel de navegación
Paneles
Iconos muy grandes
Iconos grandes
Iconos medianos
Iconos pequeños
Lista
Detalles
Diseño
Ordenar por
Vista actual
Casillas de elemento
Extensiones de nombre de archivo
Elementos ocultos
Ocultar elementos seleccionados
Mostrar u ocultar
Opciones
« Sirve › TriangleCompiler-TRONCO › Entrega1 › ide-triangle-v1.1.src › dist
Buscar en ...
FOC
Imágenes
Fotos en iClou
Triangle
Clases
Compi
Proyecto 1
Pruebas
OneDrive
Este equipo
Descargas
Documentos
Escritorio
Imágenes
Música
Objetos 3D
Videos
Windows (C:)
Red
2 elementos
IDE-Triangle
README