**ANALIZADOR LÉXICO,**

**PRIMERA ENTREGA COMPILADORES**

**Realizado por:**

**Luis Miguel Marulanda Torres**

**Angel David Santa Giraldo**

**Juan Camilo Fernández**

**Presentado a:**

**Fernando Galindres**

**Compiladores**

**Ingeniería de sistemas y computación**

**2019-1**

**UTP**

**Analizador Sintáctico**

**Descripción del proyecto**

En este trabajo, se realizó la implantación de un analizador sintáctico para el lenguaje Mini Java utilizando la librería ply.yacc en el lenguaje de programación Python. Adicionalmente, cabe mencionar que entre las herramientas utilizadas está el analizador léxico implementado para la primera entrega del semestre. Para la realización del proyecto fue necesario proceder de la siguiente manera:

1. Depurar la gramática entregada por el profesor (la cual reconoce el lenguaje MiniJava) con el fin de que dejara de estar en la forma EBNF y estuviera en BNF: Tras la primera depuración, la gramática quedó con aproximadamente 95 reglas, todas en su forma BNF sin presentar problemas.
2. Tratar de corregir los conflictos que se pudieran identificar en la gramática: Tras analizar, se identificaron problemas como el siguiente: la regla (Location : ID | EXPR ‘.’ ID | ‘[‘ EXPR ‘]’) tiene conflicto con la regla (Method : ID | EXPR ‘.’ ID) ya que en determinados casos se elimina el determinismo y existe la ambigüedad. Además de otros problemas como el menos unario y el menos entre dos expresiones numéricas.
3. Definir la gramática en Python en la notación que corresponde, habiendo especificado la precedencia.
4. Definir el análisis que hará el ply.yacc como el LALR.
5. Ejecutar el código para determinar los conflictos cuando se genere el Parser.out: Después de ejecutar por primera vez, se encontraron más de 50 conflictos, (conflictos desplaza reduce) y conflictos críticos reduce/reduce como los mencionados en el literal 2 de esta descripción.
6. Tratar de corregir los conflictos: Tras investigar se logró reducir significativamente el número de conflictos determinando como el más crítico el de menos unario y menos entre dos expresiones numéricas.
7. Implementar el árbol AST identificando las características comunes en la procedencia con cada una de las reglas de la gramática.

Finalmente, tras el proceso anteriormente descrito la gramática final utilizada para reconocer el lenguaje fue la siguiente:

Rule 0 S' -> PROGRAMA

Rule 1 PROGRAMA -> CLASS\_DECL\_LIST

Rule 2 PROGRAMA -> epsilon

Rule 3 CLASS\_DECL\_LIST -> CLASS\_DECL

Rule 4 CLASS\_DECL\_LIST -> CLASS\_DECL CLASS\_DECL\_LIST

Rule 5 CLASS\_DECL -> CLASS ID EXTENDS ID LLAVEIZQ FIELD\_OR\_METHOD\_DECL\_LIST LLAVEDER

Rule 6 CLASS\_DECL -> CLASS ID EXTENDS ID LLAVEIZQ LLAVEDER

Rule 7 CLASS\_DECL -> CLASS ID LLAVEIZQ FIELD\_OR\_METHOD\_DECL\_LIST LLAVEDER

Rule 8 CLASS\_DECL -> CLASS ID LLAVEIZQ LLAVEDER

Rule 9 FIELD\_OR\_METHOD\_DECL\_LIST -> FIELD\_DECL

Rule 10 FIELD\_OR\_METHOD\_DECL\_LIST -> FIELD\_DECL FIELD\_OR\_METHOD\_DECL\_LIST

Rule 11 FIELD\_OR\_METHOD\_DECL\_LIST -> METHOD\_DECL

Rule 12 FIELD\_OR\_METHOD\_DECL\_LIST -> METHOD\_DECL FIELD\_OR\_METHOD\_DECL\_LIST

Rule 13 FIELD\_DECL -> TYPE ID FINALIZADOR

Rule 14 FIELD\_DECL -> TYPE ID LIST\_AUX\_IDS ;

Rule 15 LIST\_AUX\_IDS -> COMA ID

Rule 16 LIST\_AUX\_IDS -> COMA ID LIST\_AUX\_IDS

Rule 17 METHOD\_DECL -> TYPE ID PARENTIZQ PARENTDER BLOCK

Rule 18 METHOD\_DECL -> TYPE ID PARENTIZQ FORMALS PARENTDER BLOCK

Rule 19 METHOD\_DECL -> VOID ID PARENTIZQ PARENTDER BLOCK

Rule 20 METHOD\_DECL -> VOID ID PARENTIZQ FORMALS PARENTDER BLOCK

Rule 21 FORMALS -> TYPE ID

Rule 22 FORMALS -> TYPE ID COMA FORMALS

Rule 23 TYPE -> INT

Rule 24 TYPE -> BOOLEAN

Rule 25 TYPE -> STRING

Rule 26 TYPE -> ID

Rule 27 TYPE -> TYPE CORCHIZQ CORCHDER

Rule 28 BLOCK -> LLAVEIZQ LLAVEDER

Rule 29 BLOCK -> LLAVEIZQ VAR\_DECL\_STATEMENTS\_LIST LLAVEDER

Rule 30 VAR\_DECL\_STATEMENTS\_LIST -> VAR\_DECL

Rule 31 VAR\_DECL\_STATEMENTS\_LIST -> VAR\_DECL VAR\_DECL\_STATEMENTS\_LIST

Rule 32 VAR\_DECL\_STATEMENTS\_LIST -> STATEMENT

Rule 33 VAR\_DECL\_STATEMENTS\_LIST -> STATEMENT VAR\_DECL\_STATEMENTS\_LIST

Rule 34 VAR\_DECL -> TYPE ID LIST\_IDS\_EXPRESSIONS FINALIZADOR

Rule 35 VAR\_DECL -> TYPE ID FINALIZADOR

Rule 36 VAR\_DECL -> TYPE ID ASIGNAR EXPRESSION LIST\_IDS\_EXPRESSIONS FINALIZADOR

Rule 37 VAR\_DECL -> TYPE ID ASIGNAR EXPRESSION FINALIZADOR

Rule 38 LIST\_IDS\_EXPRESSIONS -> COMA ID

Rule 39 LIST\_IDS\_EXPRESSIONS -> COMA ID ASIGNAR EXPRESSION

Rule 40 LIST\_IDS\_EXPRESSIONS -> COMA ID LIST\_IDS\_EXPRESSIONS

Rule 41 LIST\_IDS\_EXPRESSIONS -> COMA ID ASIGNAR EXPRESSION LIST\_IDS\_EXPRESSIONS

Rule 42 STATEMENT -> ASSIGN FINALIZADOR

Rule 43 STATEMENT -> CALL FINALIZADOR

Rule 44 STATEMENT -> RETURN EXPRESSION FINALIZADOR

Rule 45 STATEMENT -> RETURN FINALIZADOR

Rule 46 STATEMENT -> IF PARENTIZQ EXPRESSION PARENTDER STATEMENT ELSE STATEMENT

Rule 47 STATEMENT -> IF PARENTIZQ EXPRESSION PARENTDER STATEMENT

Rule 48 STATEMENT -> WHILE PARENTIZQ EXPRESSION PARENTDER STATEMENT

Rule 49 STATEMENT -> BREAK FINALIZADOR

Rule 50 STATEMENT -> CONTINUE FINALIZADOR

Rule 51 STATEMENT -> BLOCK

Rule 52 ASSIGN -> LOCATION ASIGNAR EXPRESSION

Rule 53 LOCATION -> ID

Rule 54 LOCATION -> EXPRESSION ACCESO ID

Rule 55 LOCATION -> ID CORCHIZQ EXPRESSION CORCHDER

Rule 56 CALL -> METHOD PARENTIZQ ACTUALS PARENTDER

Rule 57 CALL -> METHOD PARENTIZQ PARENTDER

Rule 58 METHOD -> ID

Rule 59 METHOD -> EXPRESSION ACCESO ID

Rule 60 ACTUALS -> EXPRESSION

Rule 61 ACTUALS -> EXPRESSION COMA ACTUALS

Rule 62 EXPRESSION -> LOCATION

Rule 63 EXPRESSION -> CALL

Rule 64 EXPRESSION -> THIS

Rule 65 EXPRESSION -> NEW ID PARENTIZQ PARENTDER

Rule 66 EXPRESSION -> NEW TYPE CORCHIZQ EXPRESSION CORCHDER

Rule 67 EXPRESSION -> EXPRESSION ACCESO LENGTH

Rule 68 EXPRESSION -> BINARY\_EXPRESSION

Rule 69 EXPRESSION -> NEGLOGICA EXPRESSION

Rule 70 EXPRESSION -> MINUS EXPRESSION

Rule 71 EXPRESSION -> LITERAL

Rule 72 EXPRESSION -> PARENTIZQ EXPRESSION PARENTDER

Rule 73 BINARY\_EXPRESSION -> EXPRESSION PLUS EXPRESSION

Rule 74 BINARY\_EXPRESSION -> EXPRESSION MINUS EXPRESSION

Rule 75 BINARY\_EXPRESSION -> EXPRESSION POR EXPRESSION

Rule 76 BINARY\_EXPRESSION -> EXPRESSION DIVIDE EXPRESSION

Rule 77 BINARY\_EXPRESSION -> EXPRESSION MODULO EXPRESSION

Rule 78 BINARY\_EXPRESSION -> EXPRESSION YAND EXPRESSION

Rule 79 BINARY\_EXPRESSION -> EXPRESSION OOR EXPRESSION

Rule 80 BINARY\_EXPRESSION -> EXPRESSION MENORQUE EXPRESSION

Rule 81 BINARY\_EXPRESSION -> EXPRESSION MENORIGUAL EXPRESSION

Rule 82 BINARY\_EXPRESSION -> EXPRESSION MAYORQUE EXPRESSION

Rule 83 BINARY\_EXPRESSION -> EXPRESSION MAYORIGUAL EXPRESSION

Rule 84 BINARY\_EXPRESSION -> EXPRESSION COMPIGUAL EXPRESSION

Rule 85 BINARY\_EXPRESSION -> EXPRESSION COMPDIFERENTE EXPRESSION

Rule 86 LITERAL -> NUMERO

Rule 87 LITERAL -> BIN

Rule 88 LITERAL -> NUMEROC

Rule 89 LITERAL -> CADENA

Rule 90 LITERAL -> TRUE

Rule 91 LITERAL -> FALSE

Rule 92 LITERAL -> NULL

Rule 93 epsilon -> <empty>

Donde cada uno de los símbolos no terminales salieron de la definición de Tokens de la primera entrega.

**Descripción del proceso del AST:**

Para la construcción del árbol hubo diferentes inconvenientes, en primer lugar, existía una confusión con el proceso de creación de clases, es decir, se creía que había que crear una clase diferente para proceder con cada grupo de elementos comunes en la gramática. Sin embargo, se logro llegar al resultado final de la siguiente manera:

Se identificó que todos los nodos tienen la posibilidad de tener hijos, los cuales podrán imprimirse por conveniencia para mostrar el recorrido deseado. La clase “Node” la definimos con el fin de darle tratamiento similar a las reglas gramaticales, es decir, que se puedan imprimir o desplazar según el caso.

La clase nodo contiene entonces un constructor y un imprimir, en la cual, el constructor reúne los atributos (hijos y tipo) además, tiene un método imprimir el cual procede de la siguiente manera:

Sí el nodo es de clase padre, es decir, tiene hijos, entonces de manera recursiva el método se va llamando, hasta imprimir incluso los nodos que no tienen hijos (que serían hojas).

Finalmente cabe mencionar que el AST se llama mediante la función evaluar, la cual asigna los valores de la producción, excluyendo el No terminal que la produce, ya que estos serían los hijos del nodo.

**Retroalimentación**

Para la entrega, se necesitó de más de una semana de trabajo, en la cual el proceso más demorado fue el de tratar de arreglar la gramática para reducir conflictos de esta. La parte del AST fue también compleja, pero más trabajable en el sentido de que se hizo una misma clase para todas las producciones y se tuvieron en cuenta las recomendaciones del profesor. La comprensión del AST fue difícil, y la parte más fácil fue la de poner la gramática ya depurada en Python.