Universidad de San Carlos de Guatemala

Facultad de Ingeniería

Escuela de Ciencias y Sistemas

Organización de lenguajes y compiladores 1

Manual técnico – Proyecto 2

Nombre: Xhunik Nikol Miguel Mutzutz

Sección: C

Carné: 201900462

MANUAL TÉCNICO

Gramática

```
ini-> main_statements EOF ;


main_statements-> main_statements main_statement
    | main_statement ;


main_statement-> run_st END_SENTENCE
    | function
    | method ;


standard_statements-> standard_statements standard_statement
    | standard_statement ;


standard_statement-> declare_array_1 END_SENTENCE
    | declare_array_2 END_SENTENCE
    | declaration END_SENTENCE
    | assign END_SENTENCE
    | print_st END_SENTENCE
    | println_st END_SENTENCE
    | if
    | while
    | do_while
    | do_until
    | for
    | switch
    | call END_SENTENCE
    | increment END_SENTENCE
    | decrement END_SENTENCE
    | BREAK END_SENTENCE
    | CONTINUE END_SENTENCE
    | RETURN expr END_SENTENCE
     | RETURN END_SENTENCE ;



expr-> arithmetic
    | relational
    | logical
    | ternary
    | group
    | value
    | cast
    | increment
    | decrement
    | call
    | access_array
    | access_matrix
    | to_lower_st
    | to_upper_st
    | round_st
```

```
    | typeof_st
    | tostring_st ;



relational-> expr LESS expr
    | expr GREATER expr
    | expr LESS_EQUAL expr
    | expr GREATER_EQUAL expr
    | expr EQUAL expr
    | expr NOT_EQUAL expr ;


arithmetic-> expr ADD expr
    | expr MINUS expr
    | expr PRODUCT expr
    | expr DIVISION expr
    | expr MODULE expr
    | expr POWER expr
    | MINUS expr ;


logical-> expr AND expr
    | expr OR expr
    | NOT expr ;


value-> DECIMAL
    | INTEGER
    | LOGICAL
    | STRING
    | CHAR
    | IDENTIFIER ;


ternary-> expr TERNARY_IF expr TERNARY_ELSE expr ;


group-> OPEN_PARENTHESIS expr CLOSE_PARENTHESIS ;


cast-> OPEN_PARENTHESIS TYPE CLOSE_PARENTHESIS expr ;


increment-> IDENTIFIER INCREMENT ;


decrement-> IDENTIFIER DECREMENT ;


list_identifiers-> list_identifiers COMMA IDENTIFIER
    | IDENTIFIER ;


declaration-> TYPE list_identifiers
    | TYPE list_identifiers ASSIGNMENT expr ;


assign-> list_identifiers ASSIGNMENT expr ;



if-> IF OPEN_PARENTHESIS expr CLOSE_PARENTHESIS OPEN_BRACE
standard_statements CLOSE_BRACE
```

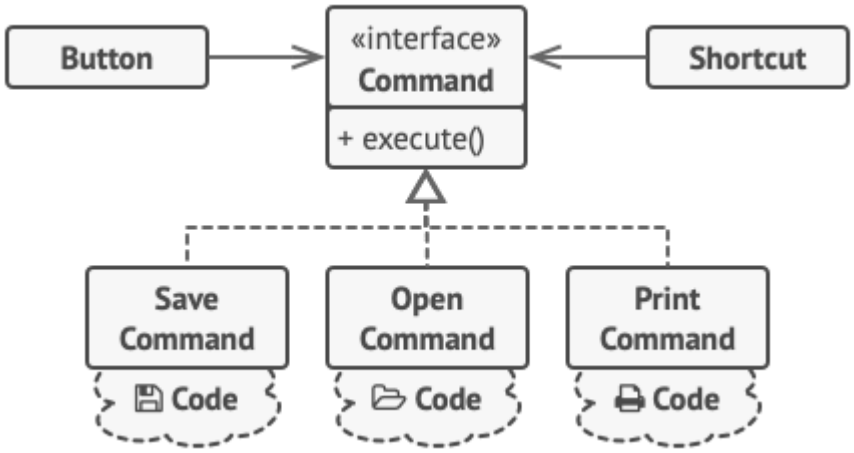
```
    | IF OPEN_PARENTHESIS expr CLOSE_PARENTHESIS OPEN_BRACE
standard_statements CLOSE_BRACE ELSE OPEN_BRACE standard_statements
CLOSE_BRACE
    | IF OPEN_PARENTHESIS expr CLOSE_PARENTHESIS OPEN_BRACE
standard_statements CLOSE_BRACE elifs
    | IF OPEN_PARENTHESIS expr CLOSE_PARENTHESIS OPEN_BRACE
standard_statements CLOSE_BRACE elifs ELSE OPEN_BRACE
standard_statements CLOSE_BRACE ;


elifs-> elifs ELIF OPEN_PARENTHESIS expr CLOSE_PARENTHESIS OPEN_BRACE
standard_statements CLOSE_BRACE
    | ELIF OPEN_PARENTHESIS expr CLOSE_PARENTHESIS OPEN_BRACE
standard_statements CLOSE_BRACE ;


while-> WHILE OPEN_PARENTHESIS expr CLOSE_PARENTHESIS OPEN_BRACE
standard_statements CLOSE_BRACE ;


do_while-> DO OPEN_BRACE standard_statements CLOSE_BRACE WHILE
OPEN_PARENTHESIS expr CLOSE_PARENTHESIS END_SENTENCE ;


do_until-> DO OPEN_BRACE standard_statements CLOSE_BRACE UNTIL
OPEN_PARENTHESIS expr CLOSE_PARENTHESIS END_SENTENCE ;


parameters-> parameters COMMA TYPE IDENTIFIER
    | TYPE IDENTIFIER ;


function-> IDENTIFIER OPEN_PARENTHESIS parameters CLOSE_PARENTHESIS
TERNARY_ELSE TYPE OPEN_BRACE standard_statements CLOSE_BRACE
    | IDENTIFIER OPEN_PARENTHESIS CLOSE_PARENTHESIS  TERNARY_ELSE TYPE
OPEN_BRACE standard_statements CLOSE_BRACE ;


method-> IDENTIFIER OPEN_PARENTHESIS parameters CLOSE_PARENTHESIS
TERNARY_ELSE VOID OPEN_BRACE standard_statements CLOSE_BRACE
    | IDENTIFIER OPEN_PARENTHESIS parameters CLOSE_PARENTHESIS
OPEN_BRACE standard_statements CLOSE_BRACE
    | IDENTIFIER OPEN_PARENTHESIS CLOSE_PARENTHESIS TERNARY_ELSE VOID
OPEN_BRACE standard_statements CLOSE_BRACE
    | IDENTIFIER OPEN_PARENTHESIS CLOSE_PARENTHESIS OPEN_BRACE
standard_statements CLOSE_BRACE ;



arguments-> arguments COMMA expr
    | expr ;


call-> IDENTIFIER OPEN_PARENTHESIS arguments CLOSE_PARENTHESIS
    | IDENTIFIER OPEN_PARENTHESIS CLOSE_PARENTHESIS ;


for-> FOR OPEN_PARENTHESIS for_init END_SENTENCE expr END_SENTENCE
for_update CLOSE_PARENTHESIS OPEN_BRACE standard_statements CLOSE_BRACE
;

for_init-> assign
    | declaration ;
```

```
for_update-> assign
    | increment
    | decrement ;


switch-> SWITCH OPEN_PARENTHESIS expr CLOSE_PARENTHESIS OPEN_BRACE
cases CLOSE_BRACE
    | SWITCH OPEN_PARENTHESIS expr CLOSE_PARENTHESIS OPEN_BRACE cases
DEFAULT TERNARY_ELSE standard_statements CLOSE_BRACE
    | SWITCH OPEN_PARENTHESIS expr CLOSE_PARENTHESIS OPEN_BRACE DEFAULT
TERNARY_ELSE standard_statements CLOSE_BRACE ;

cases-> cases CASE expr TERNARY_ELSE standard_statements
    | CASE expr TERNARY_ELSE standard_statements ;


declare_array_1-> TYPE OPEN_BRACKET CLOSE_BRACKET IDENTIFIER ASSIGNMENT
NEW TYPE OPEN_BRACKET expr CLOSE_BRACKET
    | TYPE OPEN_BRACKET CLOSE_BRACKET IDENTIFIER ASSIGNMENT OPEN_BRACE
list_expr CLOSE_BRACE ;

list_expr-> list_expr COMMA expr
    | expr ;

list_list_expr-> list_list_expr COMMA OPEN_BRACE list_expr CLOSE_BRACE
    | OPEN_BRACE list_expr CLOSE_BRACE ;


declare_array_2-> TYPE OPEN_BRACKET CLOSE_BRACKET OPEN_BRACKET
CLOSE_BRACKET IDENTIFIER ASSIGNMENT NEW TYPE OPEN_BRACKET expr
CLOSE_BRACKET OPEN_BRACKET expr CLOSE_BRACKET
    | TYPE OPEN_BRACKET CLOSE_BRACKET OPEN_BRACKET CLOSE_BRACKET
IDENTIFIER ASSIGNMENT OPEN_BRACE list_list_expr CLOSE_BRACE ;


access_array-> IDENTIFIER OPEN_BRACKET expr CLOSE_BRACKET ;


access_matrix-> IDENTIFIER OPEN_BRACKET expr CLOSE_BRACKET OPEN_BRACKET
expr CLOSE_BRACKET ;


print_st-> PRINT OPEN_PARENTHESIS expr CLOSE_PARENTHESIS ;


println_st-> PRINTLN OPEN_PARENTHESIS expr CLOSE_PARENTHESIS ;


to_lower_st-> TOLOWER OPEN_PARENTHESIS expr CLOSE_PARENTHESIS ;


to_upper_st-> TOUPPER OPEN_PARENTHESIS expr CLOSE_PARENTHESIS ;


round_st-> ROUND OPEN_PARENTHESIS expr CLOSE_PARENTHESIS ;


typeof_st-> TYPEOF OPEN_PARENTHESIS expr CLOSE_PARENTHESIS ;


tostring_st-> TOSTRING OPEN_PARENTHESIS expr CLOSE_PARENTHESIS ;
```

```
run_st-> RUN IDENTIFIER OPEN_PARENTHESIS list_expr CLOSE_PARENTHESIS
    | RUN IDENTIFIER OPEN_PARENTHESIS CLOSE_PARENTHESIS ;
```

Patrón Interprete

Para modelar la entrada en base a la gramática, se utilizó el patrón interprete, de forma que sea sencillo hacer las traducciones y la generación de graficas.



Herramientas utilizadas

Análisis Léxico y Sintáctico – Jison

Se utilizo esta herramienta para generar un autómata capaz de reconocer la gramática tipo 3 que define los lexemas de entrada del lenguaje.

Enumerables

Se utilizaron varios tipos de datos, definidos como enumerables, los cuales representan, operadores, tipos de datos, etc.

Clases de instrucción

- **Asignación:** define la operación de asignación posterior a la definición.
- **Case:** case de la sentencias switch-case.
- **Declaración:** define la operación de definición de variables.
- **Elif:** Operación de combinar else-if para múltiples casos.
- **Ejecutar:** llamadas a funciones.
- **Para:** Ciclo for, similar al utilizado en lenguajes como c++, con expresiones aritméticas e incrementos.
- **Función:** funciones con instrucciones de retorno.
- **Si:** sentencia if, contiene todos los casos, if simple o con sentencia else, también contempla sentencias elif.
- **Operación:** sentencia base, contiene los terminales, expresiones aritméticas y booleanas, contempla la ejecución de funciones y agrupación en paréntesis.
- **Parámetros:** parámetros de firmas de funciones y procedimientos
- **Imprimir:** funciones que permiten definir la función de imprimir una operación.
- **Procedimiento:** funciones sin retorno.
- **Repetir:** sentencia repetir hasta que una condición se cumpla.
- **Retorno:** retorna una operación.
- **Switch:** sentencia contenedora de múltiples casos.
- **Mientras:** ciclo mientras-hacer

Funciones importantes (definidas en la interface)

- **execute:** Ejecuta la o las instrucciones definidas en el elemento, recibe la tabla de símbolos, no retorna ningún valor.
- **evaluate:** Evalúa una expresión, recibe la tabla de símbolos en caso sea necesario consultar una variable y retorna el valor de la operación final.
- **graph:** Grafica el árbol AST y muestra la grafica.

Explicación de la solución

El primer paso del proceso de traducción es la lectura por parte del analizador léxico, el cual genera un flujo de tokens, posteriormente, este flujo de tokens es pasado al analizador sintáctico el cual, genera de forma ascendente un árbol sintáctico, el cual contiene toda la información relevante, en forma jerárquica.

Una vez obtenida esta información se recorre el árbol utilizando el algoritmo de búsqueda por profundidad, generando las traducciones de los bloques de código de forma recursiva, generando las salidas en los lenguajes de salida.

Búsqueda en profundidad – algoritmo

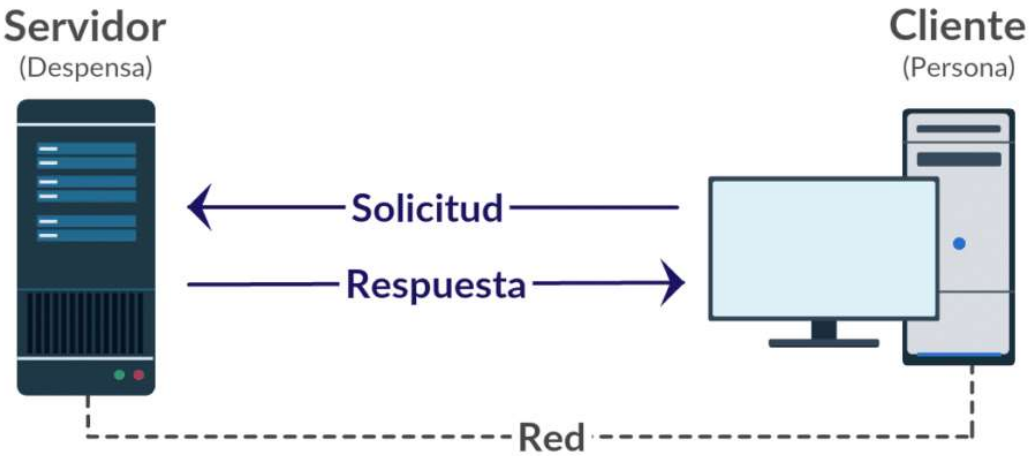
```
DFS(grafo G)
    PARA CADA vértice u ∈ V[G] HACER
            estado[u] ← NO_VISITADO
            padre[u] ← NULO
    tiempo ← 0
    PARA CADA vértice u ∈ V[G] HACER
            SI estado[u] = NO_VISITADO ENTONCES
                    DFS_Visitar(u,tiempo)
```

```
DFS_Visitar(nodo u, int tiempo)
    estado[u] ← VISITADO
    tiempo ← tiempo + 1
    d[u] ← tiempo
    PARA CADA v ∈ Vecinos[u] HACER
            SI estado[v] = NO_VISITADO ENTONCES
                    padre[v] ← u
                    DFS_Visitar(v,tiempo)
    estado[u] ← TERMINADO
    tiempo ← tiempo + 1
    f[u] ← tiempo
```

Cliente Servidor

Para este proyecto implementamos modelo cliente servidor, de manera que se proporciona el servicio de compilación y ejecución desde un servidor web, para este sistema se utilizó:

1. **Frontend:** Se uso el framework de Angular, para manejar la vista web.
2. **Backend:** Se utilizo express y node.js para correr el backend, además por motivos de simplicidad se utilizo el lenguaje **TypeScript**, para manejar mejor la POO.



Link del repositorio: https://github.com/xhuniktzi/OLC1-201900462/