Universidad de San Carlos de Guatemala

Facultad de Ingeniería

Escuela de Ciencias y Sistemas

Organización de lenguajes y compiladores 1

Manual técnico – Proyecto 1

Nombre: Xhunik Nikol Miguel Mutzutz

Sección: C

Carné: 201900462

MANUAL TÉCNICO

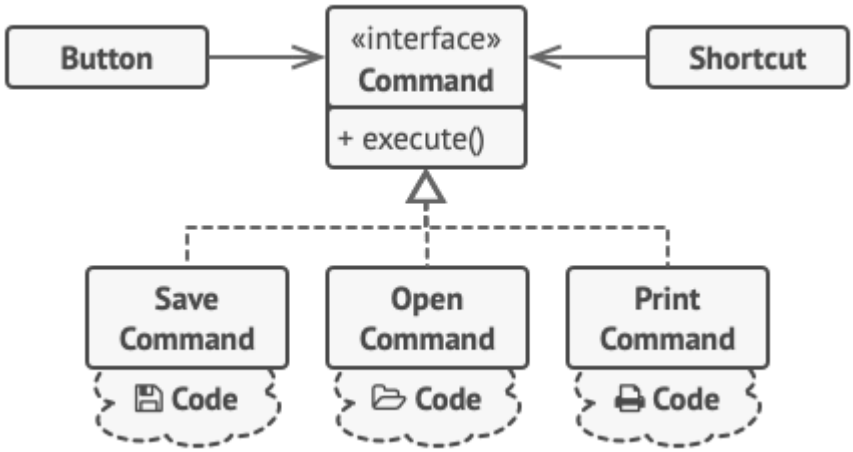
Gramática

```
ini -> START mainstatements END

mainstatements -> mainstatements mainstatement
| mainstatement

mainstatement -> statement
| definitionprocedure

statements -> statements statement
| statement

statement -> ENTER name_list AS TYPEDEF WITH_VALUE expr DOTCOMMA
| name_list ARROW expr DOTCOMMA
| if
| for
| WHILE expr DO statements ENDWHILE
| WHILE expr DO ENDWHILE
| REPEAT statements ENDREPEAT expr
| REPEAT ENDREPEAT expr
| EXEC func DOTCOMMA
| PRINT expr DOTCOMMA
| PRINTLN expr DOTCOMMA
| SWITCH expr DO cases ENDSWITCH
| SWITCH expr DO cases ELSE THEN statements ENDSWITCH

definitionprocedure ->  DEFPROCEDURE ID statements ENDPROCEDURE
| DEFPROCEDURE ID PARAMS PARSTART params_list PAREND statements
ENDPROCEDURE
| FUNCTION ID TYPEDEF funcstatements ENDFUNCTION
| FUNCTION ID TYPEDEF PARAMS PARSTART params_list PAREND
funcstatements ENDFUNCTION

funcstatement -> statement
| RETURN expr DOTCOMMA

funcstatements -> funcstatements funcstatement
| funcstatement

if -> IF expr THEN statements ENDIF
| IF expr THEN statements ELSE statements ENDIF
| IF expr THEN statements elifs ENDIF
| IF expr THEN statements elifs ELSE statements ENDIF

for -> FOR ID ARROW expr TO expr DO statements ENDFOR
| FOR ID ARROW expr TO expr WITHINCREMENTAL NUM DO statements
ENDFOR
| FOR ID ARROW expr TO expr DO ENDFOR
| FOR ID ARROW expr TO expr WITHINCREMENTAL NUM DO ENDFOR
```

```
params_list -> params_list COMMA ID TYPEDEF
| ID TYPEDEF

args_list -> args_list COMMA expr
| expr

name_list -> name_list COMMA ID
| ID

func -> ID PARSTART PAREND
| ID PARSTART args_list PAREND

elifs -> elifs ELIF expr THEN statements
| ELIF expr THEN statements

cases -> cases OPENQUESTION expr CLOSEQUESTION THEN statements
| OPENQUESTION expr CLOSEQUESTION THEN statements

expr -> symbols
| unitary
| aritmetic
| logical
| PARSTART expr PAREND

aritmetic -> expr ADD expr
| expr SUBSTRACT expr
| expr MULTIPLY expr
| expr DIVISION expr
| expr POW SBRACKETOPEN expr SBRACKETCLOSE
| expr MODULE SBRACKETOPEN expr SBRACKETCLOSE
| NUM

logical -> expr MAJOR expr
| expr MINOR expr
| expr MAJOREQUALS expr
| expr MINOREQUALS expr
| expr EQUALS expr
| expr NOTEQUALS expr
| expr AND expr
| expr OR expr
| BOOLEAN

symbols -> ID
| STR
| CHAR

unitary -> SUBSTRACT expr
| NOT expr
```

Patrón Interprete

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



Herramientas utilizadas

Análisis Léxico – JFLEX

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

Análisis Sintáctico – CUP

Se utilizo esta herramienta capaz de generar un programa, capaz de evaluar un flujo de tokens y validar el orden correcto del mismo, este programa utiliza una gramática LALR, la cual es una variante de las gramáticas LR(1), con la diferencia de que la precedencia de las producciones debe ser colocada explícitamente.

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)

Como parte de las funciones obligatorias definidas en la interface tenemos 4, getGuid, traverse, translatePython y translateGolang.

- **getGuid:** retorna el guid, del elemento, sirve para la renderización del graphviz, para el abstract syntax tree.
- **Traverse:** recorre todo el nodo, retorna el string del nodo y de todos los nodos hijos, ejecuta la operación de generación del grafo del árbol.
- **translatePython:** traduce el nodo y todos los nodos hijos, al lenguaje Python.
- **translateGolang:** traduce el nodo y todos los nodos hijos, al lenguaje Golang.

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
```

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