Instituto Tecnológico de Costa Rica

Compiladores e Intérpretes

Primera Tarea:

Gramática BNF

Profesor:

Allan Rodríguez Dávila

Estudiantes:

Bryan Londoño Marchena

Dylan Rodríguez Chavarría

Segundo Semestre 2025

Descripción del Problema

Se presenta la asignación de generar una gramática BNF basada en un lenguaje imperativo de tipado fuerte y explícito. Para esto se deben de generar todas las producciones necesarias para llevar a cabo las tareas básicas de un lenguaje imperativo como lo son: creación de identificadores, asignación de valores a los identificadores, operaciones aritméticas, operaciones relacionales, manejo de estructuras de datos como las listas, uso de estructuras de control, entrada de datos, salida de datos, creación y ejecución de funciones.
Estas producciones generadas deben de apoyarse con el establecimiento de los elementos terminales y no terminales.

Diseño del Programa

Terminales:

Símbolos, palabras reservadas y agrupaciones de caracteres:

<left_parenthesis> ::= “є”
<right_parenthesis> ::= “ɜ”
<left_brace> ::= "{"
<right_brace> ::= "}"
<wall_comment> ::= “|”
<left_exclamation> ::= “!”
<right_exclamation> ::= “¡”
<left_block> ::= “¿”
<right_block> ::= “?”
<plus_operator> ::= “+”
<minus_operator> ::= “-“
<multiplication_operator> ::= “*”
<division_operator> ::= “/”
<int_division_operator> ::= “//”
<modulo_operator> ::= “%”
<power_operator> ::= “^”
<increment_operator> ::= "++"
<decrement_operator> ::= "--"
<assignment_operator> ::= "="
<digit1to9_literal> ::= [1-9]

<decimal_digit_literal> ::= [0-9]
<zero_literal> ::= 0
<dot_literal> ::= “.”
<letter_or_underscore> ::= [a-zA-Z_]
<identifier_char> ::= [a-zA-Z0-9_]
<int_keyword> ::= “int”
<float_keyword> ::= “float”
<bool_keyword> ::= “bool”
<char_keyword> ::= “char”
<string_keyword> ::= “string”
<void_keyword> ::= "void"
<principal_keyword> ::= "principal"
<let_keyword> ::= “let”
<input_keyword> ::= “input”
<output_keyword> ::= “output”
<comma_keyword> ::= “,”
<loop_keyword> ::= "loop"
<exit_when_keyword> ::= "exit when"
<end_loop_$_keyword> ::= "end loop$
<for_keyword> ::= "for"
<step_keyword> ::= "step"
<to_keyword> ::= "to"
<downto_keyword> ::= "downto"
<do_keyword> ::= "do"
<return_keyword> ::= "return"
<break_keyword> ::= "break"
<greater_operator> ::= “>”
<less_operator> ::= “<”
<greater_equal_operator> ::= “>=”
<less_equal_operator> ::= “<=”
<equal_operator> ::= “==”
<not_equal_operator> ::= “!=”
<delimiter> ::= “$”
<line_break> ::= “\n”
<or_operator> ::= “~”
<and_operator> ::= “@”
<not_operator> ::= “Σ”
<decide_of_keyword> ::= “decide of”
<elseif_keyword> ::= “elif”
<else_keyword> ::= “else”
<single_quote> ::= “‘”
<double_quote> ::= “””
<type> ::= <int_keyword> | <float_keyword> | <bool_keyword> | <char_keyword> |
<string_keyword>

Tipos de datos literales

<int_literal> ::= <minus_operator>? <digit1to9_literal> <decimal_digit_literal>*
<int_literal> ::= <zero_literal>
<float_literal> ::= <zero_literal> <dot_literal> <zero_literal>
<float_literal> ::= <minus_operator>? <zero_literal> <dot_literal>
<decimal_digit_literal>* <digit1to9_literal>+
<float_literal> ::= <minus_operator>? <digit1to9_literal> <decimal_digit_literal>*
<dot_literal> (<decimal_digit_literal>* <digit1to9_literal>+ | <zero_literal>)
<bool_literal> ::= “True” | “False”
<char_literal> ::= <single_quote> “.” <single_quote>
<string_literal> ::= <double_quote> (.| <line_break>)* <double_quote>
<literal> ::= <int_literal> | <float_literal> | <bool_literal> | <char_literal> |
<string_literal>

Identificador

<identifier> ::= <letter_or_underscore> <identifier_char>*

No Terminales

Operaciones Aritméticas

<unary_negative>
<postfix_expression>
<arithmetic_expression>
<term>
<power>
<factor>
<arithmetic_operands>

Operaciones Lógicas

<condition>
<condition_simple>
<relational_expression>
<relational_operator_numeric>
<equality_expression>
<equality_operator>
<logical_operator>
<not_condition>

Operaciones Relacionales

<conditionR>
<equality_condition>

<relational_condition>
<simple_condition>

Declaración de variables

<declaration>
<reassignment>

Declaración, modificación de arreglos y acceso a sus elementos

<array_declaration>
<assign_elements_array>
<array_access>

Comentarios

<simple_comment>
<multiple_comment>

Funciones Input/Output

<input_statement>
<output_statement>

Bloques de código

<block>
<statements>
<statement>

Decide Of

<decide_of>
<else_if>
<else>
<multiple_condition>

Loop

<loop>
<break>

For

<for>

Funciones

<function>

<functions>
<params>
<return>

Main y Variables Globales

<principal>
<global_variables>

Símbolo Inicial

<program>

Producciones

Las producciones que se generaron fueron a partir de los requerimientos indicados. Estas se pueden encontrar en el archivo adjunto “producciones.txt” en el cual se puede encontrar el contenido de cada una.

Sobre esto también se desea aclarar que las producciones para las operaciones aritméticas fueron tomadas del siguiente documento encontrado en internet donde se explicaba cómo aplicar el orden de precedencia: https://athena.ecs.csus.edu/~gordonvs/135/resources/04bnfParseTrees.pdf

Análisis de Resultados

Objetivo	¿Logrado?
Permitir la creación de funciones y dentro de ellas ubicar estructuras de control y otras sentencias de código como las asignaciones.	Sí
Crear variables globales	Sí
Manejar los tipos de variables enteras, flotantes, booleanas, caracteres, cadenas de caracteres (string) y arreglo estático unidimensional.	Sí
Se permite crear arreglos de tipo char y entero. Además, se permite obtener y modificar sus elementos, y ser utilizados en expresiones. Se permite creación con asignación (¿ y ?).	Sí
Creación y asignación de variables.	Sí
Combinar literales, variables, arreglos y funciones en una expresión	Sí

Expresiones aritméticas respetando el orden usual de precedencia matemática. (+, -, *, //, / % y ^)	Sí
Expresiones aritméticas unarias de negativo, ++ y - -	Sí
Expresiones relacionales sobre enteros y flotantes.	Sí
Operador igual y diferente.	Sí
Expresiones lógicas de conjunción, disyunción y negación.	Sí
Combinación de expresiones aritméticas, relacionales y lógicas.	Sí
Estructura decide of con return.	Sí
Estructura loop con return y break.	Sí
Estructura for con return y break.	Sí
Función Input/Output para leer e imprimir	Sí
Creación de funciones indicando su tipo de retorno y los parámetros	Sí
Comentarios de una y múltiples líneas.	Sí
Definir un único procedimiento inicial por medio del cuál se da la ejecución de los programas y la combinación de todos los elementos.	Sí

Bitácora

Se encuentra en git en el siguiente repositorio:

https://github.com/DylanRodriguez22/Tarea1_Compiladores.git