Instituto Tecnológico de Costa Rica

Ingeniería en Computación

Compiladores e Intérpretes

Tarea 1

Profesor: Ing. Allan Rodriguez Davila

Estudiante: Carnet:

Jordano Escalante 2018161994

Segundo Semestre

Fecha: 11 de Septiembre

Año:2025



**Descripción del problema**

Se requiere desarrollar una Gramática BNF para un lenguaje imperativo con las siguientes características:

1. Permitir la creación de funciones, y dentro de ellas, estructuras de control, bloques de código ( ¿ y ? ) y sentencias de código.
2. Se permite crear variables globales.
3. Manejar los tipos de variables enteras, flotantes, booleanas, caracteres, cadenas de caracteres (string) y arreglo estático unidimensional.
4. 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 ? ).
5. Permitir sentencias para creación de variables (locales y globales), creación y asignación de expresiones y asignación de expresiones a variables, y algunos casos, sólo expresiones sin asignación. En el caso de creaciones utilizar la palabra reservada let.
6. Las expresiones permiten combinar literales, variables, arreglos y/o funciones, de los tipos reconocidos en la gramática.
7. Debe permitir operadores y operandos, respetando precedencia (usual matemática) y permitiendo el uso de paréntesis ( є y э ).
8. Permitir expresiones aritméticas binarias de suma (+), resta (-), división (// y /) entera o decimal según el tipo--, multiplicación (\*), módulo (%) y potencia (^). Para enteros o reales. Permitir expresiones aritméticas unarias de negativo (-), ++ y -- después del operando; el negativo se puede aplicar a literales enteros y flotantes, el ++ y -- se aplica a variables enteres y flotantes.
9. Permitir expresiones relacionales (sobre enteros y flotantes) de menor, menor o igual, mayor, mayor o igual, igual y diferente. Los operadores igual y diferente permiten adicionalmente tipo booleano.
10. Permitir expresiones lógicas de conjunción (@), disyunción (~) y negación (ésta debe ser de tipo caracter (Σ)).
11. Debe permitir sentencias de código para las diferentes expresiones mencionadas anteriormente y su combinación, el delimitador de final de expresión será el carácter dólar ($). Además, dichas expresiones pueden usarse en las condicionales y bloques de las siguientes estructuras de control.
12. Debe permitir el uso de tipos y la combinación de expresiones aritméticas (binarias y unarias), relacionales y lógicas, según las reglas gramaticales, aritméticas, relacionales y lógicas del Paradigma Imperativo, por ejemplo, tomando como referencia el lenguaje
13. Cualquier duda con respecto al comportamiento de la funcionalidad debe validarlo con el profesor.
14. La gramática genera un lenguaje con tipado explícito y fuerte.
15. Debe permitir las funciones de leer (enteros y flotantes) y escribir en la salida estándar (cadena carácter, enteros, boolean y flotantes), se pueden escribir literales o variables. Utilizar palabras reservadas output e input.
16. Debe permitir la creación y utilización de funciones, estos deben retornar valores (entero, flotantes, char y booleanos) y recibir parámetros (con tipo).
17. Debe definir un único procedimiento inicial main (llamado principal), por medio de la cual se inicia la ejecución de los programas, este es de tipo void y no recibe parámetros.
18. Además, debe permitir comentarios de una línea (|) o múltiples líneas (¡ y ! ).
19. Debe permitir las funciones de leer (enteros y flotantes) y escribir en la salida estándar (cadena carácter, enteros, boolean y flotantes), se pueden escribir literales o variables. Utilizar palabras reservadas output e input.
20. Debe permitir la creación y utilización de funciones, estos deben retornar valores (entero, flotantes, char y booleanos) y recibir parámetros (con tipo).
21. Debe definir un único procedimiento inicial main (llamado principal), por medio de la cual se inicia la ejecución de los programas, este es de tipo void y no recibe parámetros.
22. Debe permitir las estructuras de control:

decide of, de la forma:

decide of

(condicion -> bloque)\*

(else -> bloque)?

end decide$

loop (tipo Ada, no usa llaves de bloque)

loop instrucciones…

exit when condicion$

end loop$

for (variación de Pascal con to y downto, utiliza llaves después del do para definir un bloque)

for asignacion step valor (to o downto) expresion do

bloque

Además, permitir return y break. Las expresiones de las condiciones deberán ser valores booleanos combinando expresiones aritméticas, lógicas y relacionales.

**Diseño del programa**

La gramática se desarrolló usando la notación Backus Naur estudiada en el curso.

Se siguió el siguiente estándar:

nombreElemento -> expresión/símbolos/otros para los no terminales, terminales, símbolo inicial y producciones, además se decidió evitar el uso de caracteres especiales propios del español ya que puede incluir problemas de notación que conviene evitar en el desarrollo de una gramática para mantenerla lo más sencilla posible.

La lista completa de terminales, no terminales y producciones se encuentra en el documento adjunto llamado “gramatica BNF.txt” que se encuentra en el repositorio del proyecto, pueden ser consultadas con mejor detalle en el mismo y además se encuentran agrupadas por su tipo y uso.

**Análisis de resultados**

|  |  |  |
| --- | --- | --- |
| **Objetivo** | **Estado** | **Lección** |
| Creación de funciones | 100% |  |
| Manejar tipos | 100% |  |
| Manejar Variables | 100% |  |
| Manejar expresiones | 100% |  |
| Creación de variables | 100% |  |
| Crear arreglos | 100% |  |
| Definir alcance de variables | 100% |  |
| Separador 100% | 100% |  |
| Combinación de expresiones | 100% |  |
| Operadores y operandos | 100% |  |
| Operaciones aritméticas | 100% |  |
| Operaciones aritméticas Unarias | 100% |  |
| Expresiones relacionales | 100% |  |
| Expresiones lógicas | 100% |  |
| Tipado | 100% |  |
| Estructuras de control | 100% |  |
| Entrada y salida estándar | 100% |  |
| Procedimiento inicial main | 100% |  |
| Comentarios | 100% |  |

**Bitácora**

La bitácora fue generada de manera automática usando el sistema de control de versiones Git, puede consultar el repositorio online de la herramienta accediendo al siguiente enlace:

https://github.com/EscalanteWizard/Compilador2025.git