Instituto Tecnológico de Costa Rica

Documentación Externa - Tarea #1

Compiladores e Intérpretes

Grupo 60

Integrantes:

Bayron Rodríguez Centeno - 2020114659

Gadir Calderon Diáz - 2022327328

Profesor:

Allan Rodríguez Davila

I Semestre

2025

# Tabla de contenidos

[**Tabla de contenidos 2**](#_71xdo0dux968)

[**Descripción del problema 2**](#_tea4i9l8jteg)

[**Diseño del programa 2**](#_4vp4yyst4g8r)

[Lista de Terminales 2](#_fslsw4za444k)

[Palabras reservadas 2](#_8i3xgxt21ln9)

[Operadores aritméticos 3](#_o9i9lqkqdrfn)

[Operadores relacionales 3](#_e50mtt1fx0xt)

[Operadores lógicos 3](#_x779jttqehfq)

[Delimitadores y otros símbolos 3](#_d2ia7d9ydcn)

[Elementos léxicos 4](#_if4w9iu6qe93)

[Lista de No Terminales 4](#_2rn9tat84hjw)

[Símbolo Inicial 6](#_ynrxlvnc13rt)

[**Análisis de resultados 6**](#_binaj9ppp69q)

[Objetivos 6](#_jl976ubcphy)

[Lecciones Aprendidas 6](#_fdykncdxplt9)

[**Bitácora 7**](#_gtqgo2n1f40l)

# Descripción del problema

Un grupo de desarrolladores desea crear un nuevo lenguaje imperativo, ligero, que le permita realizar operaciones básicas para la configuración de chips, ya que esta es una industria que sigue creciendo constantemente, y cada vez estos chips necesitan ser configurados por lenguajes más ligeros y potentes. Es por esto que este grupo de desarrolladores requiere desarrollar su propio lenguaje para el desarrollo de sistemas empotrados, y como primer paso necesitan desarrollar una gramática simple y poderosa.

# Diseño del programa

## Lista de Terminales

### Palabras reservadas

if, elif, else, do, while, for, break, return, int, float, bool, char, string, void, read, write, main, true, false

### Operadores aritméticos

| + | suma |
| --- | --- |
| - | resta |
| \* | multiplicación |
| // | división |
| ~ | módulo |
| \*\* | potencia |
| ++ | incremento |
| -- | decremento |

### Operadores relacionales

| < | menor que |
| --- | --- |
| <= | menor o igual que |
| > | mayor que |
| >= | mayor o igual que |
| == | igual a |
| != | diferente de |

### Operadores lógicos

| ^ | conjunción |
| --- | --- |
| # | disyunción |
| ! | negación |

### Delimitadores y otros símbolos

| ( ) | paréntesis |
| --- | --- |
| \ / | delimitador de bloque |
| ? | delimitador de expresión |
| , | separador |
| | | asignación |
| @ | inicio de comentario de línea |
| { } | delimitador de comentario multilínea |
| [ ] | delimitadores para arreglos |

### Elementos léxicos

| Id | identificadores: Nombres de variables/funciones |
| --- | --- |
| entero | constantes numéricas enteras |
| flotante | constantes numéricas de punto flotante |
| caracter | constantes de tipo caracter |
| cadena | constantes de tipo string |
| booleano | constantes booleanas: true, false |

## Lista de No Terminales

| program | Estructura principal del programa |
| --- | --- |
| block | Bloques de código delimitados |
| decl | Declaraciones generales |
| decl\_var | Declaración de variables |
| decl\_fun | Declaración de funciones |
| stmt | Sentencias de código |
| assign | Sentencias de asignación |
| exp | Expresiones de código |
| exp\_arit | Expresiones aritméticas |
| exp\_unary | Expresiones aritméticas unarias |
| exp\_rel | Expresiones relacionales |
| exp\_log | Expresiones lógicas |
| exp\_bool | Expresiones booleanas |
| exp\_comb | Combinación de expresiones |
| ctrl | Estructuras de control |
| if | Estructura condicional if-elif-else |
| do\_while | Estructura iterativa do-while |
| for | Estructura iterativa for |
| break | Sentencia break |
| return | Sentencia return |
| read | Funciones de entrada |
| write | Funciones de salida |
| tipo | Tipos de dato |
| tipo\_arr | Tipo de arreglo |
| acc\_arr | Acceso a elementos de arreglo |
| param | Parámetros de función |
| arg | Argumentos de llamada a función |
| comment | Comentario de una línea |
| comment\_block | Comentario de múltiples líneas |
| lit | Valores literales |
| term | Términos para expresiones aritméticas |
| op\_log | operadores lógicos |
| op\_rel | operadores relacionales |
| tipo\_r | tipo retorno |
| t\_entry | tipo entrada |
| exp\_out | expresión de salida |
| lit\_int | literal entero |
| lit\_float | literal flotante |
| lit\_bool | literal booleano |
| lit\_char | literal caracter |
| lit\_string | literal cadena |
| digit | entero |
| letter | caracter |

## Símbolo Inicial

program

# Análisis de resultados

## Objetivos

| **Objetivos** | **Alcanzado** | **No alcanzado** |
| --- | --- | --- |
| Permitir la creación de funciones, y dentro de ellas, estructuras de control, bloques de código ( \ y / ) y sentencias de código. | X |  |
| Manejar los tipos de variables enteras, flotantes, booleanas, caracteres, cadenas de caracteres (string) y arreglo estático bidimensional. | X |  |
| Se permite crear arreglos de tipo entero. Además, se permite obtener y modificar sus elementos, y ser utilizados en expresiones. Usar | en creación y asignación. | X |  |
| Permitir sentencias para creación de variables, creación y asignación de expresiones y asignación de expresiones a variables, y algunos casos, sólo expresiones sin asignación. | X |  |
| Las expresiones permiten combinar literales, variables y/o funciones, de los tipos reconocidos en la gramática. | X |  |
| Debe permitir operadores y operandos, respetando precedencia (usual matemática) y permitiendo el uso de paréntesis. | X |  |
| Permitir expresiones aritméticas binarias de suma (+), resta (-), división (//) –entera o decimal según el tipo--, multiplicación (\*), módulo (~) y potencia (\*\*). Para enteros o reales. | X |  |
| Permitir expresiones aritméticas unarias de negativo (-), ++, -- , antes del operando; el negativo se puede aplicar a literales enteros y flotantes, el ++ y -- se aplica a variables enteros y flotantes. | X |  |
| 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. | X |  |
| Permitir expresiones lógicas de conjunción (^), disyunción (#) y negación (ésta debe ser de tipo caracter (!)). | X |  |
| Debe permitir sentencias de código para las diferentes expresiones mencionadas anteriormente y su combinación, el delimitador de final de expresión será el carácter pregunta (?). Además, dichas expresiones pueden usarse en las condicionales y bloques de las siguientes estructuras de control. | X |  |
| 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 C. Cualquier duda con respecto al comportamiento de la funcionalidad debe validarlo con el profesor. | X |  |
| La gramática genera un lenguaje con tipado explícito y fuerte. | X |  |
| Debe permitir las estructuras de control if-[elif]-[else], do-while y for, además, permitir return y break. Las expresiones de las condiciones deberán ser valores booleanos combinando expresiones aritméticas, lógicas y relacionales. | X |  |
| 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. | X |  |
| Debe permitir la creación y utilización de funciones, estos deben retornar valores (entero, flotantes, char y booleanos) y recibir parámetros (con tipo). | X |  |
| Debe definir un único procedimiento inicial main, por medio del cual se inicia la ejecución de los programas, este es de tipo void. | X |  |
| Además, debe permitir comentarios de una línea (@) o múltiples líneas ({ }). | X |  |

## Lecciones Aprendidas

1. **Complejidad de diseño de lenguajes:** Para lograr el diseño de una gramática formal, se ocupan muchos casos a evaluar y además de eso tiene que ser consistente en todas las reglas, así que con cada característica que se añadía se aumentaba la complejidad.

# Bitácora

<https://github.com/Bayronjrc/tarea1_Compi_Bayron_Gadyr.git>