

# Proyecto de Programación

## Introducción.

En este documento se describen las especificaciones de los lenguajes ficticios **S** y **P**; el lenguaje **S** es usado para ejemplificar la implementación las distintas partes de un lenguaje interpretado. El lenguaje **P** es un lenguaje obtenido a partir de la ampliación de la gramática del lenguaje **S** y es el lenguaje que será implementado como proyecto del curso. La gramática del lenguaje **P** es presentada en la Fig. 1; la gramática del lenguaje **S** se obtiene al eliminar de la gramática del lenguaje **P** algunas de sus reglas. En casos de gramáticas muy simples en las que el analizador sintáctico se va a codificar manualmente (o en la documentación de un lenguaje), como es el caso del lenguaje **S**, se pueden utilizar la misma notación de las expresiones regulares para simplificar el diseño de la gramática y su posterior implementación. En este caso, los corchetes indican componentes opcionales en la gramática.

```
prog
-> fn_decl_list main_prog

var_decl
-> ID ':' DATATYPE [' , ' ID ':' DATATYPE]*

fn_decl_list
-> [ FUNCTION FID ':' DATATYPE '(' [var_decl] ')'
    [VAR var_decl ';' ]
    stmt_block ]*

stmt_block
-> '{' stmt+ '}'
    | stmt

stmt
-> WRITE [STRING ':' ] lexpr ';'
    | INPUT [STRING ':' ] ID ';'
    | WHEN '(' lexpr ')' DO stmt_block
    | IF '(' lexpr ')' DO stmt_block ELSE stmt_block
    | UNLESS '(' lexpr ')' DO stmt_block
    | WHILE '(' expr ')' DO stmt_block
    | RETURN expr ';'
    | UNTIL '(' lexpr ')' DO stmt_block
    | LOOP stmt_block
    | DO stmt_block WHILE '(' lexpr ')'
    | DO stmt_block UNTIL '(' lexpr ')'
    | REPEAT NUM ':' stmt_block
    | FOR ID ':' lexpr TO lexpr DO stmt_block
    | END ';'
    | NEXT ';'
    | BREAK ';'
    | ID ':' lexpr ';'
    | ID '+=' lexpr ';'
    | ID '-=' lexpr ';'
    | ID '*=' lexpr ';'
    | ID '/=' lexpr ';'
    | ID '%=' lexpr ';'
    | ID '++' ';'
    | ID '--' ';'

lexpr
-> nexpr [[AND nexpr]* | [OR nexpr]*]
```

Fig. 1. Gramática del Lenguaje **P**. La gramática del lenguaje **S** se obtiene al eliminar las partes de las reglas resaltadas en amarillo.

```
nexpr
-> NOT '(' lexpr ')'
    | rexpr

rexpr
-> simple_expr [ ('<' | '==' | '<=' | '>' | '>=' | '!=')
    simple_expr ]

simple_expr
-> term [ ('+' | '-') term]*

term
-> factor [ ('*' | '/') factor]*

factor
-> NUM
    | ID
    | BOOL
    | '(' expr ')'
    | FID '(' [lexpr [',' lexpr]*] ')'

main_prog
-> [VAR var_decl ';' ] stmt* END
```

Fig. 1. Continuación.

## Lenguaje **S**.

El lenguaje **S** es un lenguaje imperativo extremadamente simple, fuertemente tipado, con un ambiente de ejecución completamente estático. Las características del lenguaje son enumeradas a continuación.

- 1.- Soporta únicamente tipos numéricos (num) y booleanos (bool).
- 2.- El lenguaje soporta cadenas de caracteres únicamente para fines de impresión y entrada de datos. Las cadenas de caracteres se colocan entre comillas simples.
- 3.- Sólo existen los operadores de suma (+) y multiplicación (\*); estas operaciones son del tipo  $\text{num} \times \text{num} \rightarrow \text{num}$ .
- 4.- Solo existen los operadores relacionales < y ==. El operador relacional < es del tipo  $\text{num} \times \text{num} \rightarrow \text{bool}$ . El operador relacional == soporta  $\text{bool} \times \text{bool} \rightarrow \text{bool}$  y  $\text{num} \times \text{num} \rightarrow \text{bool}$ .
- 5.- Las variables y las funciones son fuertemente tipadas; su tipo es asignado al momento de su declaración. Solo existen los tipos num y bool.
- 6.- Todas las variables son locales a las funciones en que son definidas. Los nombres válidos de las variables están conformados por una letra seguida de cero o más letras o números. Los nombres válidos de funciones están conformados por el símbolo @ seguido de una letra seguida de cero o más letras o números.
- 7.- El lenguaje provee funciones internas implementadas en el intérprete. Pueden tener cero o más argumentos. Son del tipo  $\text{num} \times \dots \times \text{num} \rightarrow \text{num}$ .
- 8.- Se pueden crear funciones de usuario. Las funciones pueden devolver un número (num) o un booleano (bool). Todas sus variables son locales. Las funciones no pueden ser llamadas recursivamente ni de forma directa ni indirecta. Sus argumentos pueden ser del tipo num o bool. Una función debe ser creada antes de que pueda ser llamada por otra función o por el programa principal.

9.- Solo hay soporte para la asignación de valores booleanos o numéricos a variables.

10.- Los condicionales en los comandos `if`, `when` y `while` solo admiten booleanos.

11.- La expresión `return` permite retornar valores numéricos o booleanos.

## Lenguaje P

A continuación se describen las variaciones respecto al lenguaje S.

1.- Se agregó el condicional `UNLESS`; este es equivalente a `WHEN NOT`.

2.- El comando `UNTIL` es equivalente a `WHILE NOT`.

4.- El comando `LOOP` realiza un ciclo infinito.

5.- `REPEAT` ejecuta el bloque de código `NUM` veces.

6.- El comando `FOR` ejecuta el ciclo desde `lexpr` `TO` `lexpr`; el incremento en cada ciclo es unitario.

7.- Los comandos `NEXT` y `BREAK` modifican la ejecución de los comandos de iteración; `BREAK` forza la terminación del ciclo, mientras que `NEXT` es equivalente a un `GOTO` al inicio.

8.- El operador `ID += lexpr` se interpreta como `ID := ID + (lexpr)`; los restantes se interpretan de forma similar.






9.- Los operadores `ID++` y `ID--` equivalen a `ID := ID + 1` y `ID := ID - 1` respectivamente.

10.- Sólo es posible construir expresiones con múltiples `AND` o múltiples `OR`; no es posible construir expresiones que mezclen directamente `AND` y `OR`, tal como es indicado en la sintaxis.

14.- El lenguaje **P**, al igual que el lenguaje **S**, es completamente estático, tal que las funciones no pueden llamarse recursivamente.

**JUAN DAVID VELÁSQUEZ HENAO**, MSc, PhD  
**Profesor Titular**

Departamento de Ciencias de la Computación y la Decisión  
Facultad de Minas  
Universidad Nacional de Colombia, Sede Medellín

 [jdvelasq@unal.edu.co](mailto:jdvelasq@unal.edu.co)  
 [@jdvelasquezh](https://twitter.com/jdvelasquezh)  
 <https://github.com/jdvelasq>  
 <https://goo.gl/prkJAq>  
 <https://goo.gl/vXH8jy>