**Rapport Projet Compilation GL**

# Compilateur du langage

***ONE FOR ALL***

Encadr´e par :

**Professeur** Youness **TABII**

Elabor´e par :

Hamza **TAMRY** Abdelwadoud **TAMTAOUI** Zakaria **SABOUR**

Rida **TAZI** Said **EL ABOUDI** Yasser **FALEH**

# Table des mati`eres

[A Description](#_bookmark0) . . . . . . . . . . . . . . . . . . . . . . 3

[B Grammaire](#_bookmark1) . . . . . . . . . . . . . . . . . . . . . . 4

## Description

Vous avez sans doute eu l’occasion d’apprendre plusieurs **langages de programmation**, et vous vous ˆetes tromp´es de syntaxe.

Trop de langages rendent la vie des d´eveloppeurs moins plaisante c’est pour cela qu’on a d´ecid´e de cr´eer un **langage universel unique** ONE FOR ALL

**compatible** avec la plupart des langages de programmation a` savoir **C**,

**javaScript**, **pascal**, **typeScript** mais aussi adapt´e a` de nouvelles r`egles grammaticales .

## Grammaire

PROGRAM → INSTRUCTIONS $

INSTRUCTIONS → { INSTRUCTION INSTRUCTIONS }

| INSTRUCTION FINSTRUCTION

FINSTRUCTION → INSTRUCTIONS

| e ps i l o n

INSTRUCTION → AFFECTATION ;

| APPEL FONCTION ;

| RETURN ;

| BOUCLE

| INPUT OUTPUT

| FONCTION

| CONTROLE

| VAR DECLARATION ;

AFFECTATION → id Fid

Fid → := EXPRESSION ;

| = EXPRESSION ;

| <= EXPRESSION ;

EXPRESSION → TERM FTERM

| ( EXPRESSION )

|  |  |  |
| --- | --- | --- |
| *FTERM* | *→*  *|*  *|* | *e ps i l o n*  *+ EXPRESSION*  = *EXPRESSION* |
| *TERM* | *→*  *|*  *|* | *FACTEUR FFACTEUR*  *+ FACTEUR*  = *FACTEUR* |
| *FFACTEUR* | *→*  *|*  *|* | *OPERATEURMULT FACTEUR*  *e ps i l o n OPERATEURSPECIAUX* |
| *OPERATEURMULT* | *→*  *|*  *|*  *|*  *|*  *|*  *|* | \*  *mult*  */ div*  *%*  *mod modulo* |
| *OPERATEURSPECIAUX* | *→*  *|* | *++*  == |

FACTEUR → id

| number

| boolean

| APPEL FONCTION

| s t r i n g

APPEL FONCTION → c a l l id ( APPEL FONCTION ARG

APPEL FONCTION ARG → ARGUMENT )

| )

ARGUMENT → id ARGUMENT1

ARGUMENT1 → , id ARGUMENT1

| e ps i l o n

RETURN → return EXPRESSION

BOUCLE → FORLOOP STATEMENT

| DOWHILELOOP STATEMENT

| WHILELOOP STATEMENT

FORLOOP STATEMENT → f o r Ffor

Ffor → ( FOR1

| id Fid3

FOR1 → VAR DECLARATION FVAR DECLARATION

FVAR DECLARATION → ; FVAR DECLARATION2

| : FVAR DECLARATION3

FVAR DECLARATION2 → CONDITIONS FCONDITIONS1

FCONDITIONS1 → ; FCONDITIONS2

FCONDITIONS2 → INSTRUCTION FINSTRUCTION1

FINSTRUCTION1 → ) FINSTRUCTION2

FINSTRUCTION2 → { INSTRUCTIONS }

| INSTRUCTION

FVAR DECLARATION3 → id Fid4

Fid4 → ) Fid5

Fid5 → { INSTRUCTIONS }

| INSTRUCTION

Fid3 → in id { INSTRUCTIONS }

|  |  |  |  |
| --- | --- | --- | --- |
| *WHILELOOP STATEMENT* | *|*  *→* | *=* | *number to number do INSTRUCTIONS ;*  *while Fwhile* |
| *Fwhile* | *→* |  | *( Fwhile 2* |
| *Fwhile 2* | *→* |  | *CONDITIONS FCONDITIONS* |
| *FCONDITIONS* | *→* |  | *) FCONDITIONS2* |
| *FCONDITIONS2* | *→*  *|* |  | *INSTRUCTION*  *{ INSTRUCTIONS }* |
| *DOWHILELOOP STATEMENT CONDITIONS ) ;* | *→* | *do { INSTRUCTIONS } while (* | |
|  | *|* | *repeat INSTRUCTIONS u n t i l CONDITIONS ;* | |
| *CONDITIONS* | *→*  *|*  *|* | *CONDITION FCONDITION*  *! ( CONDITION ) not ( CONDITION )* | |

|  |  |  |
| --- | --- | --- |
| *FCONDITION* | *→*  *|*  *|*  *|* | *&& CONDITIONS*  *| | CONDITIONS and CONDITIONS or CONDITIONS* |
|  | *|* | *e ps i l o n* |
| *CONDITION* | *→* | *EXPRESSION comparator EXPRESSION* |
| *INPUT OUTPUT* | *→*  *|*  *|*  *|*  *|*  *|*  *|*  *|*  *|*  *|*  *|*  *|*  *|* | *print ( ARGUMENT ) ; p r i n t f ( ARGUMENT ) ; s canf ( ARGUMENT ) ; input ( ARGUMENT ) ; log ( ARGUMENT ) ;*  *f p r i n t f ( ARGUMENT ) ; f s c a n f ( ARGUMENT ) ; f read ( ARGUMENT ) ; f w r i t e ( ARGUMENT ) ; write ( ARGUMENT ) ; read ( ARGUMENT ) ; puts ( ARGUMENT ) ;*  *gets ( ARGUMENT ) ;* |

FONCTION → def type FONCTION2

| f unction type FONCTION2

FONCTION2 → id ( PARAMETER ) { INSTRUCTIONS }

PARAMETER → id type PARAMETER1

PARAMETER1 → , id type PARAMETER1

|

CONTROLE → IF

| CASE

| SHORTHAND

IF → i f Fif

Fif → ( Fif 2

Fif 2 → CONDITION FCONDITION1

FCONDITION1 → ) FCONDITION2

FCONDITION2 → BLOCK IF FBLOCK IF

FBLOCK IF →

| e l s e BLOCK IF

| e l i f BLOCK IF e l s e BLOCK IF

BLOCK IF → { INSTRUCTIONS }

CASE → switch ( EXPRESSION ) { BLOCK CASE

}

BLOCK CASE → case Fcase

| de f a u l t : INSTRUCTIONS

Fcase → FACTEUR FFACTEUR1

FFACTEUR1 → : FFACTEUR2

FFACTEUR2 → INSTRUCTIONS FINSTRUCTIONS

FINSTRUCTIONS → e ps i l o n

| BLOCK CASE

SHORTHAND → ( CONDITION ) ? INSTRUCTION : INSTRUCTION VAR DECLARATION → const TYPE IDS CONST

| l e t VARS2

VARS2 → id Fid1

Fid1 → VARS TYPE FVARS TYPE

FVARS TYPE → symbole a f f Fsymbole aff 1

| , VARS2

| e ps i l o n

Fsymbole aff 1 → EXPRESSION FEXPRESSION1

FEXPRESSION1 → , VARS2

| e ps i l o n

VARS TYPE → : type

| i s type

IDS CONST → id Fid2

Fid2 → symbole a f f Fsymbole aff

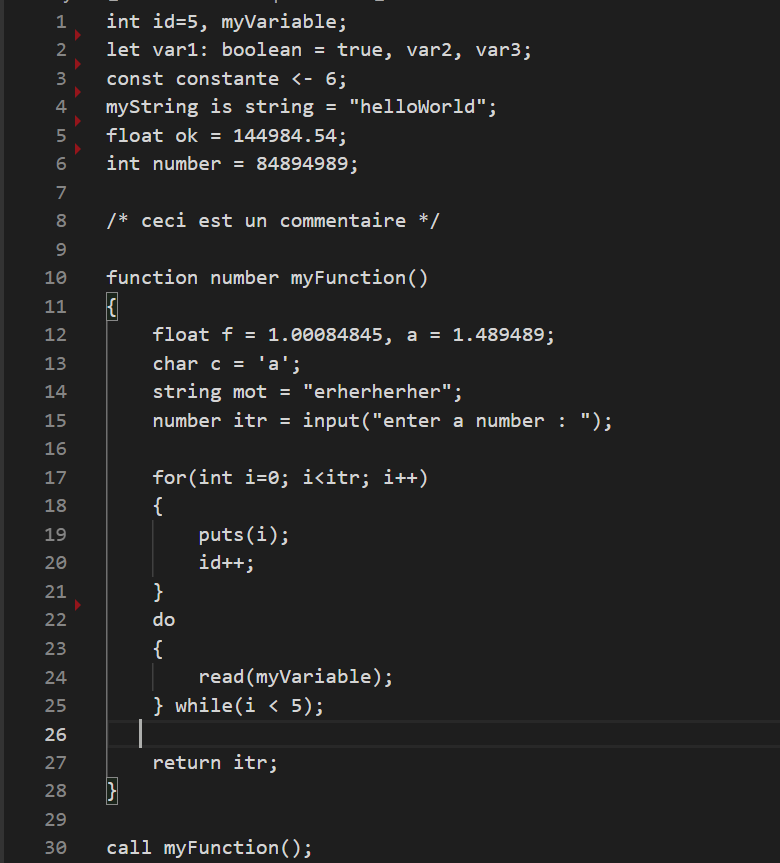
Fsymbole aff → EXPRESSION FEXPRESSION

FEXPRESSION → , IDS CONST

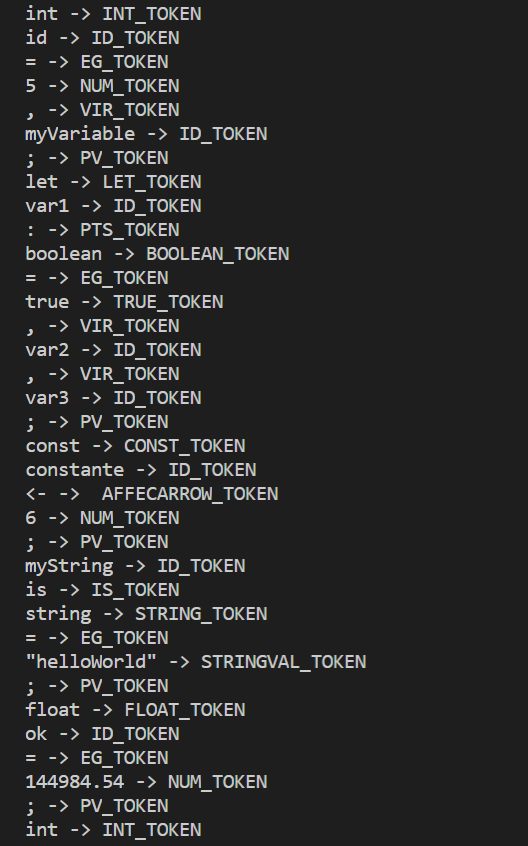
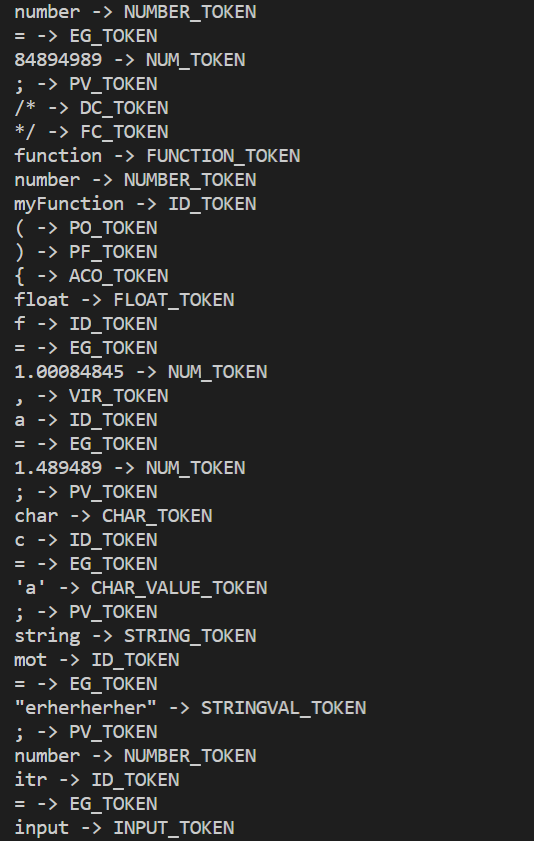
| e ps i l o n

1. **FIRST ET FOLLOW**
2. **ANALYSEUR LEXICAL**

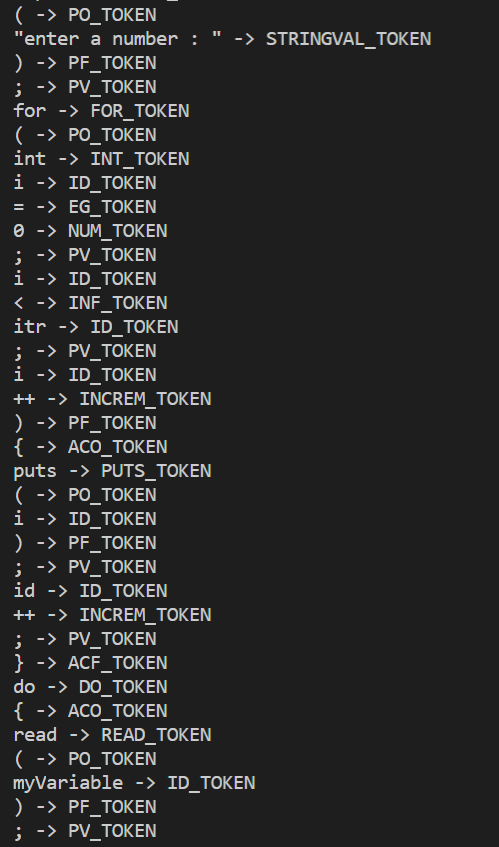
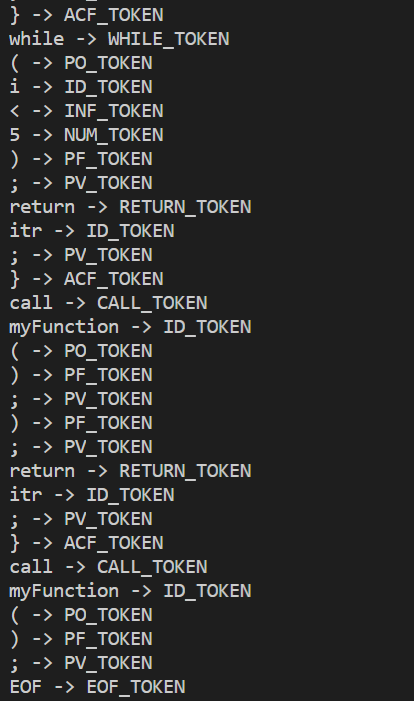
**Code source**



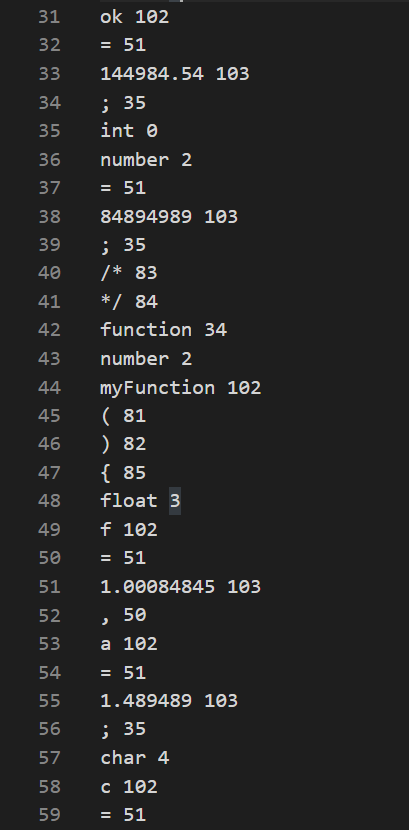
**SORTIE STANDARD (TERMINAL)**

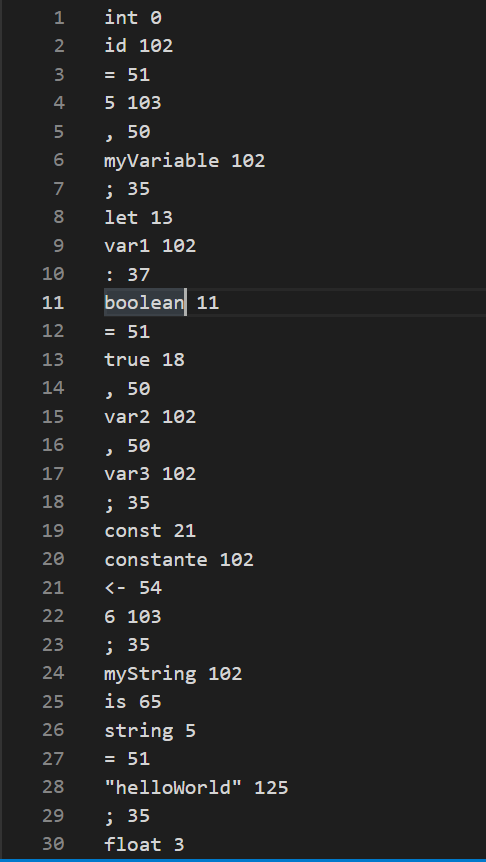
**1**  **2**

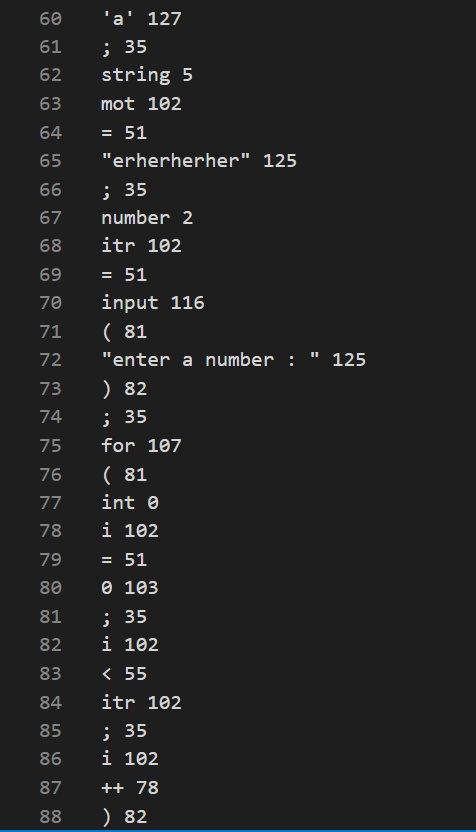
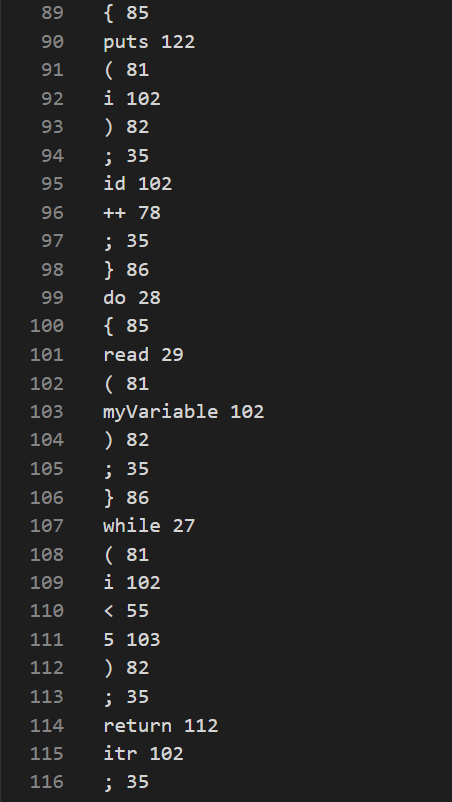
**3**  **4**



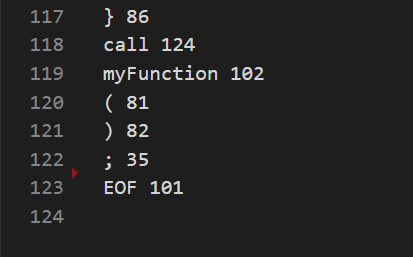
**SORTIE (FICHIER ENTRÉE DE L’ANALYSEUR SYNTAXIQUE)**

 **1**  **2**

****

 **3** **4**

**5**



1. **ANALYSEUR SYNTAXIQUE**