**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 ˆêtes trompé de syntaxe.

Trop de langages rendent la vie des développeurs 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 adapté à de nouvelles règles grammaticales .

**Opérateurs :**

• Opérateur affectation: = <- :=

• Opérateurs arithmétiques : + add - minus \* mult / div % mod modulo

• Opérateurs logiques : && and || or ! not

• Opérateurs de comparaison : < > <= => == is ===

• Autres opérateurs : ++ -- ,

**Types :**

• int • char • number

• float • string • short

• boolean •bool

**Boucles :**

•For

• for ( <insctruction > ; <condition> ; <insctruction > ) { INSTRUCTIONS }

• for (<insctruction >; <condition> ; <insctruction >) INSTRUCTION

• for ( <declaration> : id ) { INSTRUCTIONS }

• for id in id { INSTRUCTIONS }

• for id = number to number do { INSTRUCTIONS }

• While

• while ( CONDITIONS ) INSTRUCTION

• while ( CONDITIONS ) { INSTRUCTIONS }

• DO While

• do { INSTRUCTIONS } while ( CONDITIONS )

• repeat INSTRUCTIONS until ( CONDITIONS )

## 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

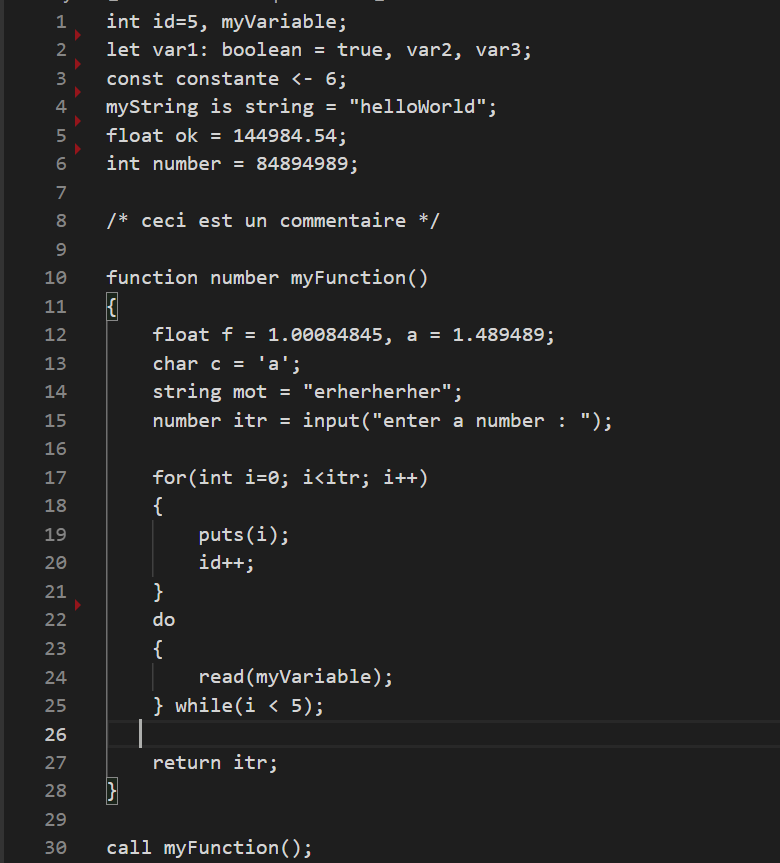
| e ps i l o n

1. **VERIFICATION LL(1) DE LA GRAMMAIRE : FIRST ET FOLLOW**

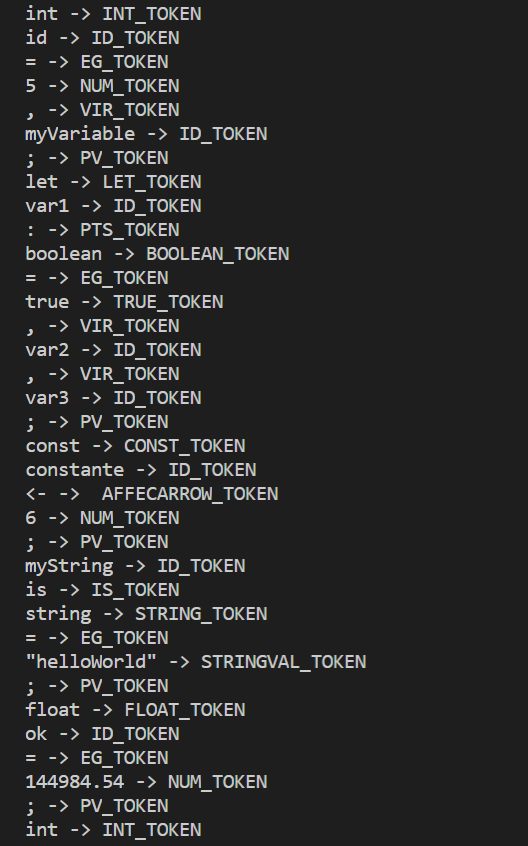
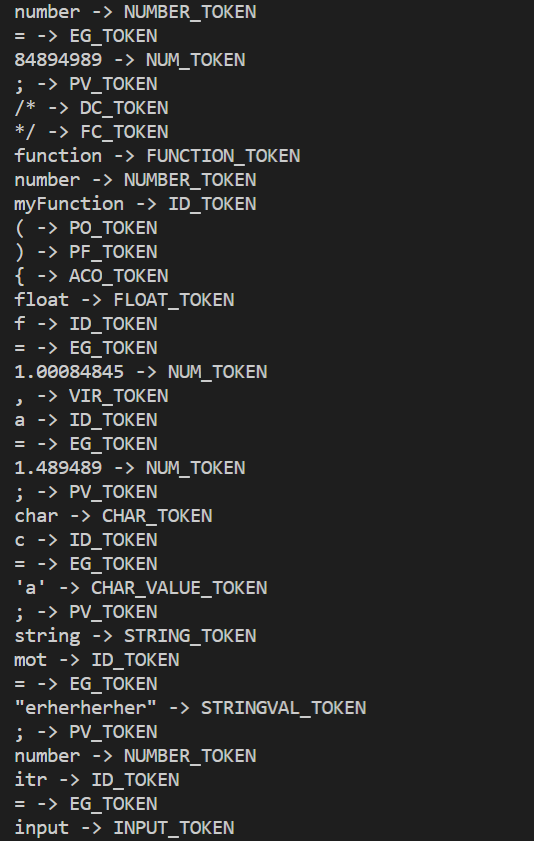
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **nonterminal** | | **first set** | | **follow set** | | |
| INSTRUCTIONS | | { id call return print printf scanf input log fprintf fscanf fread fwrite write read puts gets def  function const let for do repeat while if switch ( | | ; until } case default | | |
| FINSTRUCTION | | { id call return print printf scanf input log fprintf fscanf fread fwrite write read puts gets def  function const let for do repeat while if switch ( | | ; until } case default | | |
| AFFECTATION | | id | | ; | | |
| Fid | | : = <- | | ; | | |
| FTERM | | + - | | && || and or comparator  ) , : ; | | |
| TERM | | + - id number boolean string call | | && || and or + - comparator ) , : ; | | |
| FFACTEUR | | \* mult / div % mod modulo ++ -- | | && || and or + - comparator ) , : ; | | |
| OPERATEURMULT | | \* mult / div % mod modulo | | Id number boolean string call | | |
| OPERATEURSPECIAUX | | ++ -- | | && || and or + - comparator ) , : ; | | |
| APPEL\_FONCTION | | call | | && || and or \* mult / div % mod modulo ++ -- + - comparator ) , : ; | | |
| APPEL\_FONCTION\_ARG | | ) id | | && || and or \* mult / div % mod modulo ++ -- + - comparator ) ,  : ; | | |
| ARGUMENT1 | | , | | ) |
| RETURN | | return | | ; |
| BOUCLE | | for do repeat while | | ; until } case default { id call return print printf scanf input log fprintf fscanf fread fwrite write read puts gets def function const let for do repeat while if  switch ( ) : |
| FORLOOP\_STATEMENT | | for | | ; until } case default { id call return print printf scanf input log fprintf fscanf fread fwrite write read puts gets def function const let for do repeat while if  switch ( ) : |
| Ffor | | ( id | | ; until } case default { id call return print printf scanf input log fprintf fscanf fread fwrite write read puts gets def function const let for do repeat while if switch ( ) : |
| FOR1 | | const let | | ; until } case default { id call return print printf scanf input log fprintf fscanf fread fwrite write read puts gets def function const let for do repeat while if  switch ( ) : |
| FVAR\_DECLARATION | | ; : | | ; until } case default { id call return print printf scanf input log fprintf fscanf fread fwrite write read puts gets def function const let for do repeat while if  switch ( ) : |
| FVAR\_DECLARATION2 | | ! not ( + - id number boolean string call | | ; until } case default { id call return print printf scanf input log fprintf fscanf fread fwrite write read puts gets def function const let for do repeat while if  switch ( ) : |
| FCONDITIONS 1 | | ; | | ; until } case default { id call return print printf scanf input log fprintf fscanf fread fwrite write read puts gets def function const let for do repeat while if  switch ( ) : |
| FCONDITIONS2 | | id call return print printf scanf input log fprintf fscanf fread fwrite write read puts gets def function const let for do repeat while if switch ( | | ; until } case default { id call return print printf scanf input log fprintf fscanf fread fwrite write read puts gets def function const let for do repeat while if  switch ( ) : |
| FINSTRUCTION1 | | ) | | ; until } case default { id call return print printf scanf input log fprintf fscanf fread fwrite write read puts gets def function const let for do repeat while if  switch ( ) : |
| FINSTRUCTION2 | | { id call return print printf scanf input log fprintf fscanf fread fwrite write read puts gets def  function const let for do repeat while if switch ( | | ; until } case default { id call return print printf scanf input log fprintf fscanf fread fwrite write read puts gets def function const let for do repeat while if switch ( ) : |
| FVAR\_DECLARATION3 | | id | | ; until } case default { id call return print printf scanf input log fprintf fscanf fread fwrite write read puts gets def function const let for do repeat while if  switch ( ) : |
| Fid4 | | ) | | ; until } case default { id call return print printf scanf input log fprintf fscanf fread fwrite write read puts gets def function const let for do repeat while if  switch ( ) : |
| Fid5 | | { id call return print printf scanf input log fprintf fscanf fread fwrite write read puts gets def  function const let for do repeat while if switch ( | | ; until } case default { id call return print printf scanf input log fprintf fscanf fread fwrite write read puts gets def function const let for do repeat while if switch ( ) : |
| Fid3 | | in egal | | ; until } case default { id call return print printf scanf input log fprintf fscanf fread fwrite write read puts |
|  | |  | | gets def function const let for do repeat while if  switch ( ) : |
| WHILELOOP\_STATEMENT | | while | | ; until } case default { id call return print printf scanf input log fprintf fscanf fread fwrite write read puts gets def function const let for do repeat while if  switch ( ) : |
| Fwhile | | ( | | ; until } case default { id call return print printf scanf input log fprintf fscanf fread fwrite write read puts gets def function const let for do repeat while if switch ( ) : |
| Fwhile2 | | ! not ( + - id number boolean string call | | ; until } case default { id call return print printf scanf input log fprintf fscanf fread fwrite write read puts gets def function const let for do repeat while if  switch ( ) : |
| FCONDITIONS | | ) | | ; until } case default { id call return print printf scanf input log fprintf fscanf fread fwrite write read puts gets def function const let for do repeat while if  switch ( ) : |
| FCONDITIONS2 | | { id call return print printf scanf input log fprintf fscanf fread fwrite write read puts gets def  function const let for do repeat while if switch ( | | ; until } case default { id call return print printf scanf input log fprintf fscanf fread fwrite write read puts gets def function const let for do repeat while if switch ( ) : |
| DOWHILELOOP\_STATEMENT | | do repeat | | ; until } case default { id call return print printf scanf input log fprintf fscanf fread fwrite write read puts gets def function const let for do repeat while if  switch ( ) : |
| FCONDITION | | && || and or | | ) ; |
| CONDITIONS | | ! not ( + - id number boolean string call | | ) ; |
| INPUT\_OUTPUT | | print printf scanf input log fprintf fscanf fread fwrite write read puts gets | | ; until } case default { id call return print printf scanf input log fprintf fscanf fread fwrite write read puts gets def function const let for do repeat while if switch ( ) : |
| ARGUMENT | | id | | ) |
| FONCTION | | def function | | ; until } case default { id call return print printf scanf input log fprintf fscanf fread fwrite write read puts gets def function const let for do repeat while if  switch ( ) : |
| FONCTION2 | | id | | ; until } case default { id call return print printf scanf input log fprintf fscanf fread fwrite write read puts gets def function const let for do repeat while if switch ( ) : |
| PARAMETER | | id | | ) |
| PARAMETER1 | | , | | ) |
| CONTROLE | | if switch ( | | ; until } case default { id call return print printf scanf input log fprintf fscanf fread fwrite write read puts gets def function const let for do repeat while if switch ( ) : |
| IF | | if | | ; until } case default { id call return print printf scanf input log fprintf fscanf fread fwrite write read puts gets def function const let for do repeat while if switch ( ) : |
| Fif | | ( | | ; until } case default { id call return print printf scanf input log fprintf fscanf fread fwrite write read puts gets def function const let for do repeat while if  switch ( ) : |
| Fif2 | | ( + - id number boolean | | ; until } case default |
|  | | string call | | { id call return print printf scanf input log fprintf fscanf fread fwrite write read puts  gets def function const let for do repeat while if switch ( ) : |
| FCONDITION1 | | ) | | ; until } case default { id call return print printf scanf input log fprintf fscanf fread fwrite write read puts gets def function const let for do repeat while if  switch ( ) : |
| FCONDITION2 | | { | | ; until } case default { id call return print printf scanf input log fprintf fscanf fread fwrite write read puts gets def function const let for do repeat while if  switch ( ) : |
| FBLOCK\_IF | | else elif | | ; until } case default { id call return print printf scanf input log fprintf fscanf fread fwrite write read puts gets def function const let for do repeat while if switch ( ) : |
| BLOCK\_IF | | { | | elif else ; until } case default { id call return print printf scanf input log fprintf fscanf fread fwrite write read puts gets def function const let for do repeat while if  switch ( ) : |
| CASE | | switch | | ; until } case default { id call return print printf scanf input log fprintf fscanf fread fwrite write read puts gets def function const let for do repeat while if  switch ( ) : |
| Fcase | | id number boolean string call | | } |
| FACTEUR | | id number boolean string call | | && || and or \* mult / div % mod modulo ++  -- + - comparator ) , : ; |
| FFACTEUR1 | | : | | } |
| FFACTEUR2 | | { id call return print printf scanf input log fprintf fscanf fread fwrite write read puts gets def function const let for do repeat while if switch ( | | } |
| FINSTRUCTIONS | | case default | | } |
| BLOCK\_CASE | | case default | | } |
| SHORTHAND | | ( | | ; until } case default { id call return print printf scanf input log fprintf fscanf fread fwrite write read puts gets def function const let for do repeat while if  switch ( ) : |
| CONDITION | | ( + - id number boolean string call | | && || and or ) ; |
| INSTRUCTION | | id call return print printf scanf input log fprintf fscanf fread fwrite write read puts gets def function const let for do repeat while if switch ( | | ; until } case default { id call return print printf scanf input log fprintf fscanf fread fwrite write read puts gets def function const let for do repeat while if  switch ( ) : |
| VAR\_DECLARATION | | const let | | ; : |
| TYPE | | ∅ | | ∅ |
| Fid1 | | : is | | ; : |
| FVARS\_TYPE | | symbole\_aff , | | ; : |
| Fsymbole\_aff1 | | ( + - id number boolean string call | | ; : |
| FEXPRESSION1 | | , | | ; : |
| VARS'' | | id | | ; : |
| VARS\_TYPE | | : is | | symbole\_aff , ; : |
| Fid2 | | symbole\_aff | | ; : |
| Fsymbole\_aff | | ( + - id number boolean string call | | ; : |
| EXPRESSION | | ( + - id number boolean | | && || and or |
|  | | string call | | comparator ) , : ; |
| FEXPRESSION | | , | | ; : |
| IDS\_CONST | | id | | ; : |

1. **ANALYSEUR LEXICAL**

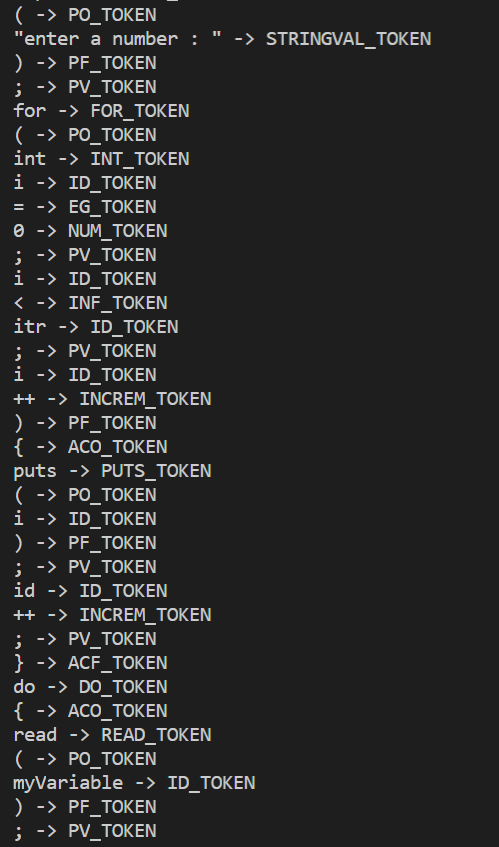
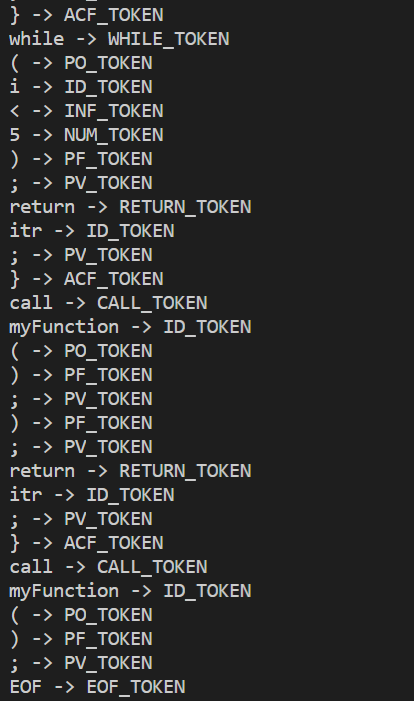
**Code source**



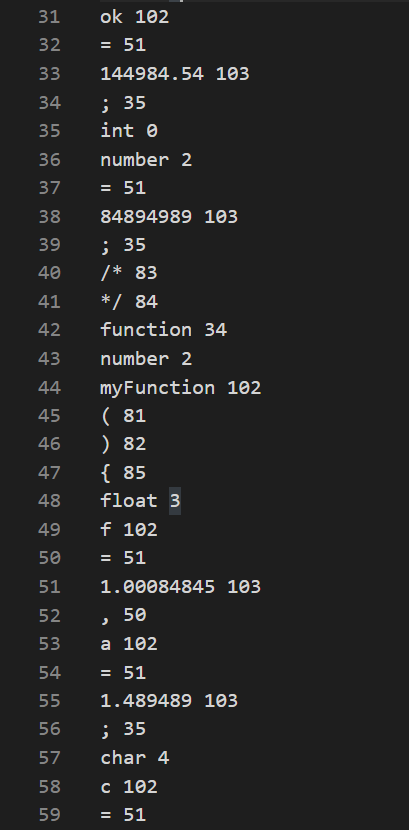
**SORTIE STANDARD (TERMINAL)**

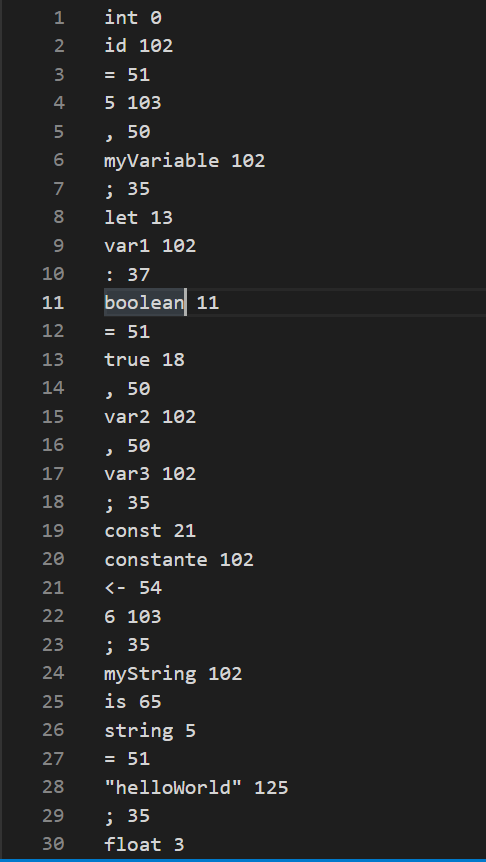
**1**  **2**

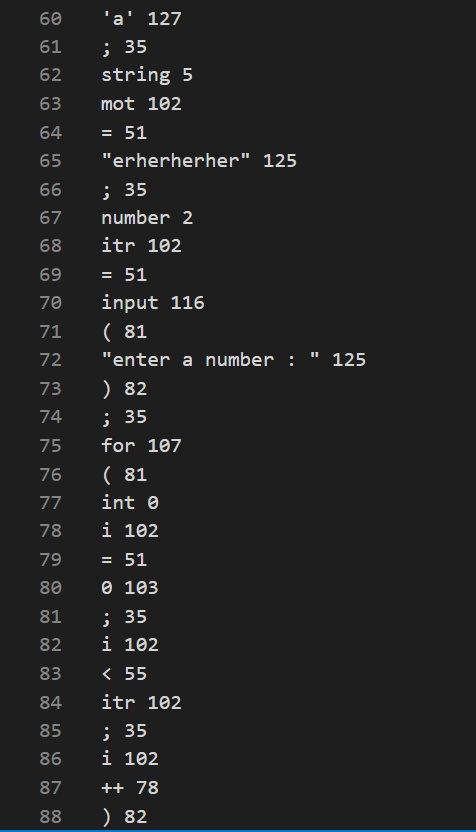
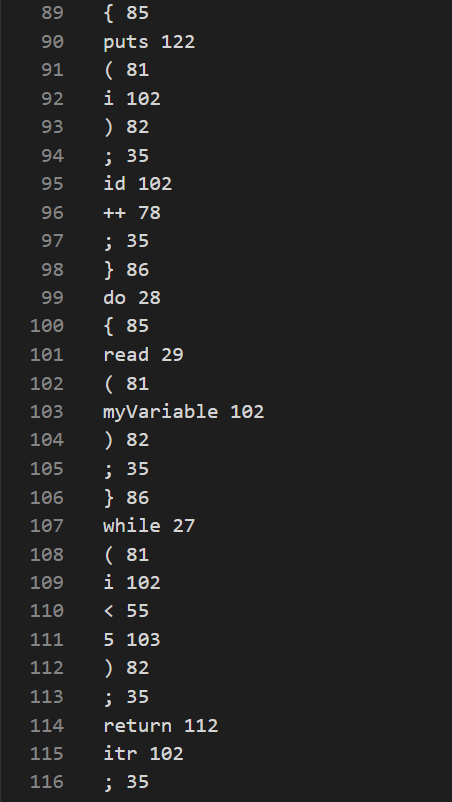
**3**  **4**



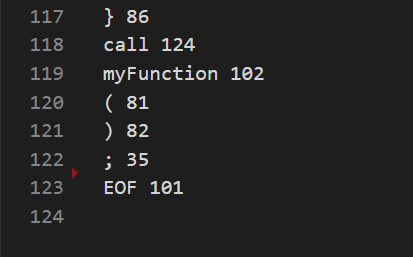
**SORTIE (FICHIER ENTRÉE DE L’ANALYSEUR SYNTAXIQUE)**

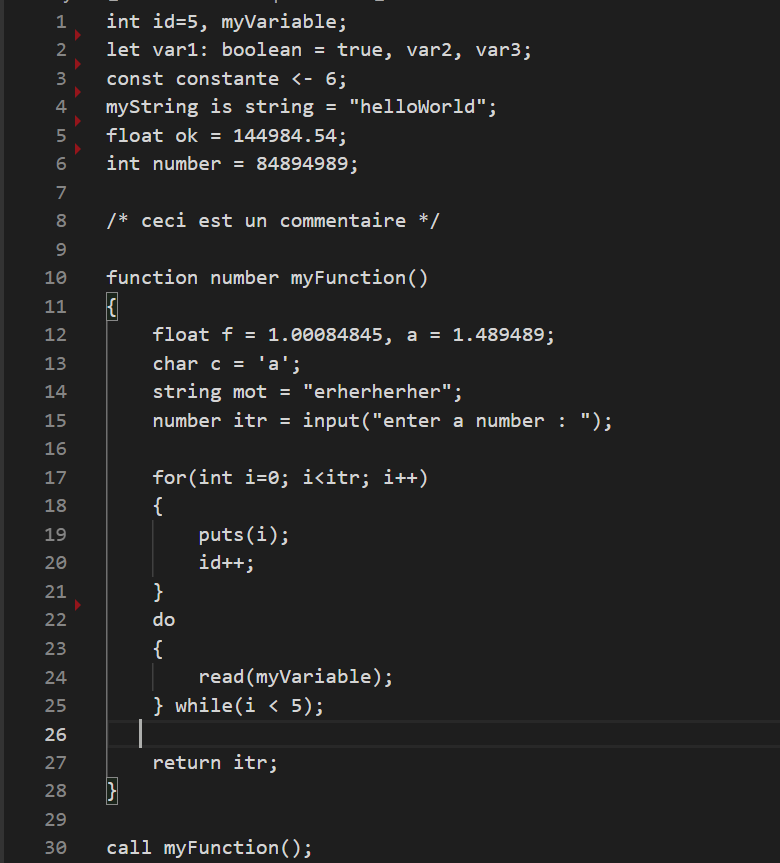
 **1**  **2**

****

 **3** **4**

**5**



1. **ANALYSEUR SYNTAXIQUE**

**Code source**

****

**Cas d’erreur :**

****