# RAPPORT PROJET COMPILATION
# GL

---

# COMPILATEUR LANGAGE (ONE_FOR_ALL)

---


**Réalisé par:**

- SABOUR Zakaria
- TAMRY HAMZA
- TAMTAOUI Abdelwadoud
- TAZI Rida
- FALEH Yasser
- SAID EL Aboudi

**Encadré par :**

- Professeur TABII Youness


Année Universitaire : 2020-2021

# Table des matières

**DESCRIPTION** .....3

**GRAMMAIRE** .....4

**ANALYSEUR LEXICAL** .....11

**ANALYSEUR SYNTAXIQUE** .....18

# DESCRIPTION

Vous avez sans doute eu l'occasion d'apprendre plusieurs langages de programmation, et vous vous êtes trompés de syntaxe. Trop de langages rendent la vie des développeurs moins plaisante c'est pour cela qu'on a décidé de créer un langage universel unique **«ONE_FOR_ALL»** compatible avec la plupart des langages de programmation à savoir c, javaScript, pascal , typeScript mais aussi adapté à de nouvelles règles grammaticales .

# GRAMMAIRE

S -> S'$

S' -> INSTRUCTIONS
| BLOCK
| epsilon

INSTRUCTIONS -> INSTRUCTION [INSTRUCTIONS | e]

INSTRUCTION -> AFFECTATION ;
| APPEL_FONCTION ;
| BOUCLE
| RETURN ;
| VAR_DECLARATION ;
| FONCTION
| CONTROLE
| EXCEPTION
| FILEHANDLING

FILEHANDLING -> print ( PARAMETERS ) ;
| printf ( PARAMETERS ) ;
| scanf ( PARAMETERS ) ;
| print ( PARAMETERS ) ;
| input ( PARAMETERS ) ;
| log ( PARAMETERS ) ;

| fprintf ( PARAMETERS ) ;

| fscanf ( PARAMETERS ) ;

| fread ( PARAMETERS ) ;

| fwrite ( PARAMETERS ) ;

| write ( PARAMETERS ) ;

| read ( PARAMETERS ) ;

| puts ( PARAMETERS ) ;

| gets ( PARAMETERS ) ;

BLOCK -> begin (end | S' end)

AFFECTATION -> const ID AFFECTATION'

| ID AFFECTATION'

AFFECTATION' -> [: [= |e] | = | <-] EXPRESSION

EXPRESSION -> TERM [OPERATEURADD TERM | e]

TERM -> FACTEUR [OPERATEURMULT FACTEUR | OPERATEURSPECIAUX |e]

| + FACTEUR

| - FACTEUR

FACTEUR -> ID

| NUMBER

| ( EXPRESSION )

| BOOLEAN

| APPEL_FONCTION

| STRING

OPERATEURSPECIAUX -> ++

| --

OPERATEURADD -> +

| add

| -

| minus

OPERATEURMULT -> *

| mult

| \

| div

| %

| mod

| modulo

APPEL_FONCTION -> ID ( PARAMETERS ) ;

FONCTION -> TYPE FONCTION'

| function FONCTION'

FONCTION' -> ID(PARAMETERS){ INSTS };

PARAMETERS -> epsilon

| PARAMETER

PARAMETER -> ID TYPE

| PARAMETER , ID TYPE

VAR_DECLARATION -> VARS'

| epsilon

VARS' -> const TYPE IDS_CONST

| let JS_IDS

| VARS''

VARS'' -> ID [: TYPE | is TYPE] OPT [, VARS'' | epsilon]

IDS_CONST -> ID AFFECTATION' [, IDS_CONST| e]

JS_IDS -> ID : TYPE OPT [, JS_IDS | epsilon]

IDS -> ID OPT [, IDS | epsilon]

OPT -> epsilon

| AFFECTATION'

RETURN -> return EXPRESSION

CONDITION -> BOOLEAN

| EXPRESSION COMPAR EXPRESSION

COMPAR -> >

| <

| >=

| <=

| !=

| ==

###### Controls ######

CONTROL -> IF

| CASE

IF -> if ( CONDITION ) [BLOCK_IF [elif BLOCK_IF else BLOCK_IF | e]

| else BLOCK_IF]

BLOCK_IF -> INST

CASE -> switch ( ID ) { BLOCK_CASE }

BLOCK_CASE -> case FACTEUR : INSTRUCTIONS [BLOCK_CASE | e]

###### Boucles ######

BOUCLE -> WHILE

| FOR

| DO_WHILE

WHILE -> while ( CONDITION ) INST

INST -> INSTRUCTION | { INSTRUCTIONS }

FOR -> for ( INSTRUCTION ; INSTRUCTION ; INSTRUCTION) INST

DO_WHILE -> do { INSTRUCTIONS } while ( CONDITION );

###### Terminaux ######

TYPE -> string
| number
| int
| boolean
| bool
| char
| integer
| boolean
| void
| float
| double
| signed
| unsigned
| short

ID -> [a-zA-Z]([a-zA-Z0-9_]*[a-zA-Z0-9])?

STRING -> " STR "

| ' STR '

STR -> [ a-zA-Z0-9_;:!,?\|[](]&~^$µéè@à` ]*

NUMBER -> [0-9]+

BOOLEAN -> true | false

# ANALYSEUR LEXICAL

## Fichier source

```
int id=5, myVariable;
let var1:bool = true, var2, var3;
const constante <- 6;
myString is string = "helloWorld";

/* ceci est un commentaire*/

int function(){
    for(int i=0; i<id; i++)
    {
        var2 = false;
        id++;
    }

    do
    {
        read(myVariable);
    }while(i < 5);

    return id;
}

```

## Sortie standard

```
int -> INT_TOKEN
id -> ID_TOKEN
= -> EG_TOKEN
5 -> NUM_TOKEN
, -> VIR_TOKEN
myVariable -> ID_TOKEN
; -> PV_TOKEN
let -> LET_TOKEN
var1 -> ID_TOKEN
: -> PTS_TOKEN
bool -> BOOL_TOKEN
= -> EG_TOKEN
true -> TRUE_TOKEN
, -> VIR_TOKEN
var2 -> ID_TOKEN
, -> VIR_TOKEN
var3 -> ID_TOKEN
; -> PV_TOKEN
const -> CONST_TOKEN
constante -> ID_TOKEN
<- ->  AFFECARROW_TOKEN
6 -> NUM_TOKEN
; -> PV_TOKEN
myString -> ID_TOKEN
is -> IS_TOKEN
string -> STRING_TOKEN
= -> EG_TOKEN
"helloWorld" -> STRINGVAL_TOKEN
; -> PV_TOKEN
/* -> DC_TOKEN
*/ -> FC_TOKEN
int -> INT_TOKEN
function -> FUNCTION_TOKEN
( -> PO_TOKEN
) -> PF_TOKEN
```

```
{ -> ACO_TOKEN
for -> FOR_TOKEN
( -> PO_TOKEN
int -> INT_TOKEN
i -> ID_TOKEN
= -> EG_TOKEN
0 -> NUM_TOKEN
; -> PV_TOKEN
i -> ID_TOKEN
< -> INF_TOKEN
id -> ID_TOKEN
; -> PV_TOKEN
i -> ID_TOKEN
++ -> INCREM_TOKEN
) -> PF_TOKEN
{ -> ACO_TOKEN
var2 -> ID_TOKEN
= -> EG_TOKEN
false -> FALSE_TOKEN
; -> PV_TOKEN
id -> ID_TOKEN
++ -> INCREM_TOKEN
; -> PV_TOKEN
} -> ACF_TOKEN
do -> DO_TOKEN
{ -> ACO_TOKEN
read -> READ_TOKEN
( -> PO_TOKEN
myVariable -> ID_TOKEN
) -> PF_TOKEN
; -> PV_TOKEN
} -> ACF_TOKEN
while -> WHILE_TOKEN
( -> PO_TOKEN
i -> ID_TOKEN
< -> INF_TOKEN
5 -> NUM_TOKEN
) -> PF_TOKEN
; -> PV_TOKEN
return -> RETURN_TOKEN
id -> ID_TOKEN
; -> PV_TOKEN
} -> ACF_TOKEN
```

## Fichier sortie de L’analyseur lexical et entrée du syntaxique

```
int 0
id 102
= 51
5 103
, 50
myVariable 102
; 35
let 13
var1 102
: 37
bool 12
= 51
true 18
, 50
var2 102
, 50
var3 102
; 35
const 21
constante 102
<- 54
6 103
; 35
myString 102
is 65
string 5
= 51
"helloWorld" 113
; 35
/* 83
*/ 84
int 0
function 34
( 81
) 82
{ 85
for 107
( 81
int 0
i 102
```

```
= 51
0 103
; 35
i 102
< 55
id 102
; 35
i 102
++ 78
) 82
{ 85
var2 102
= 51
false 19
; 35
id 102
++ 78
; 35
} 86
do 28
{ 85
read 29
( 81
myVariable 102
) 82
; 35
} 86
while 27
( 81
i 102
< 55
5 103
) 82
; 35
return 112
id 102
; 35
} 86

```

**Exemple cas d'erreur « string non proprement déclaré »**

```
1  int id=5, myVariable;
2  let var1:bool = true, var2, var3;
3  const constante <- 6;
4  myString is string = "helloWorld;
5
6  int function(){
7      for(int i=0; i<id; i++)
8      {
9          var2 = false;
10         id++;
11     }
12
13     do
14     {
15         read(myVariable);
16     }while(i < 5);
17
18     return id;
19 }
20
```

```
 -> string non proprement declar├® : missing "
"helloWorld;

int function(){
    for(int i=0; i<id; i++)
    {
        var2 = false;
        id++;
    }

    do
    {
        read(myVariable);
    }while(i < 5);

    return id;
}
 -> ERREUR_TOKEN
```

# ANALYSEUR SYNTAXIQUE