Rapport Projet Compilation GL

Compilateur du langage *ONE FOR ALL*

Encadré par :

Professeur Youness **TABII**

Elaboré par :

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

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

Table des matières

A 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 .

B Grammaire

PROGRAM	→	*INSTRUCTIONS $*
INSTRUCTIONS	→	{ *INSTRUCTION INSTRUCTIONS* }
	|	*INSTRUCTION FINSTRUCTION*
FINSTRUCTION	→	*INSTRUCTIONS*
	|	*epsilon*
INSTRUCTION	→	*AFFECTATION ;*
	|	*APPEL_FONCTION ;*
	|	*RETURN ;*
	|	*BOUCLE*
	|	*INPUT_OUTPUT*
	|	*FONCTION*
	|	*CONTROLE*
	|	*VAR_DECLARATION ;*
AFFECTATION	→	*id Fid*
Fid	→	*:= EXPRESSION ;*
	|	*= EXPRESSION ;*
	|	*<- EXPRESSION ;*
EXPRESSION	→	*TERM FTERM*
	|	*(EXPRESSION)*

FTERM	$\rightarrow$	$epsilon$
	$\mid$	$+$ *EXPRESSION*
	$\mid$	$-$ *EXPRESSION*
TERM	$\rightarrow$	*FACTEUR FFACTEUR*
	$\mid$	$+$ *FACTEUR*
	$\mid$	$-$ *FACTEUR*
FFACTEUR	$\rightarrow$	*OPERATEURMULT FACTEUR*
	$\mid$	$epsilon$
	$\mid$	*OPERATEURSPECIAUX*
OPERATEURMULT	$\rightarrow$	$*$
	$\mid$	$mult$
	$\mid$	$/$
	$\mid$	div
	$\mid$	$\%$
	$\mid$	mod
	$\mid$	$modulo$
OPERATEURSPECIAUX	$\rightarrow$	$++$
	$\mid$	$--$

FACTEUR	$\rightarrow$	id
	$\mid$	$number$
	$\mid$	$boolean$
	$\mid$	*APPEL_FONCTION*
	$\mid$	$string$
APPEL_FONCTION	$\rightarrow$	$call\ id\ ($ *APPEL_FONCTION_ARG*
APPEL_FONCTION_ARG	$\rightarrow$	*ARGUMENT* $)$
	$\mid$	$)$
ARGUMENT	$\rightarrow$	id *ARGUMENT1*
ARGUMENT1	$\rightarrow$	$,\ id$ *ARGUMENT1*
	$\mid$	$epsilon$
RETURN	$\rightarrow$	$return$ *EXPRESSION*
BOUCLE	$\rightarrow$	*FORLOOP_STATEMENT*
	$\mid$	*DOWHILELOOP_STATEMENT*
	$\mid$	*WHILELOOP_STATEMENT*
FORLOOP_STATEMENT	$\rightarrow$	$for\ Ffor$

$Ffor$	$\rightarrow$	$($ *FOR1*
	$\mid$	id $Fid3$
FOR1	$\rightarrow$	*VAR_DECLARATION* *FVAR_DECLARATION*
FVAR_DECLARATION	$\rightarrow$	$;$ *FVAR_DECLARATION2*
	$\mid$	$:$ *FVAR_DECLARATION3*
FVAR_DECLARATION2	$\rightarrow$	*CONDITIONS* *FCONDITIONS1*
FCONDITIONS1	$\rightarrow$	$;$ *FCONDITIONS2*
FCONDITIONS2	$\rightarrow$	*INSTRUCTION* *FINSTRUCTION1*
FINSTRUCTION1	$\rightarrow$	$)$ *FINSTRUCTION2*
FINSTRUCTION2	$\rightarrow$	$\{$ *INSTRUCTIONS* $\}$
	$\mid$	*INSTRUCTION*
FVAR_DECLARATION3	$\rightarrow$	id $Fid4$
$Fid4$	$\rightarrow$	$)$ $Fid5$

Fid5 → { *INSTRUCTIONS* }

| *INSTRUCTION*

Fid3 → *in id* { *INSTRUCTIONS* }

| = *number to number do INSTRUCTIONS* ;

WHILELOOP_STATEMENT → *while Fwhile*

Fwhile → (*Fwhile2*

Fwhile2 → *CONDITIONS FCONDITIONS*

FCONDITIONS →) *FCONDITIONS2*

FCONDITIONS2 → *INSTRUCTION*

| { *INSTRUCTIONS* }

DOWHILELOOP_STATEMENT → *do* { *INSTRUCTIONS* } *while* (*CONDITIONS*) ;

| *repeat INSTRUCTIONS until CONDITIONS* ;

CONDITIONS → *CONDITION FCONDITION*

| *!* (*CONDITION*)

| *not* (*CONDITION*)

FCONDITION	$\rightarrow$	*&& CONDITIONS*
	|	*|| CONDITIONS*
	|	*and CONDITIONS*
	|	*or CONDITIONS*
	|	*epsilon*
CONDITION	$\rightarrow$	*EXPRESSION comparator EXPRESSION*
INPUT_OUTPUT	$\rightarrow$	*print (ARGUMENT) ;*
	|	*printf (ARGUMENT) ;*
	|	*scanf (ARGUMENT) ;*
	|	*input (ARGUMENT) ;*
	|	*log (ARGUMENT) ;*
	|	*fprintf (ARGUMENT) ;*
	|	*fscanf (ARGUMENT) ;*
	|	*fread (ARGUMENT) ;*
	|	*fwrite (ARGUMENT) ;*
	|	*write (ARGUMENT) ;*
	|	*read (ARGUMENT) ;*
	|	*puts (ARGUMENT) ;*
	|	*gets (ARGUMENT) ;*

FONCTION	→	*def type FONCTION2*
	|	*function type FONCTION2*
FONCTION2	→	*id (PARAMETER) { INSTRUCTIONS }*
PARAMETER	→	*id type PARAMETER1*
PARAMETER1	→	*, id type PARAMETER1*
	|	
CONTROLE	→	*IF*
	|	*CASE*
	|	*SHORTHAND*
IF	→	*if Fif*
Fif	→	*(Fif2*
Fif2	→	*CONDITION FCONDITION1*
FCONDITION1	→	*) FCONDITION2*
FCONDITION2	→	*BLOCK_IF FBLOCK_IF*
FBLOCK_IF	→	

	|	*else BLOCK_IF*
	|	*elif BLOCK_IF else BLOCK_IF*
BLOCK_IF	→	{ *INSTRUCTIONS* }
CASE	→	*switch (EXPRESSION)* { *BLOCK_CASE* }
BLOCK_CASE	→	*case Fcase*
	|	*default : INSTRUCTIONS*
Fcase	→	*FACTEUR FFACTEUR1*
FFACTEUR1	→	*: FFACTEUR2*
FFACTEUR2	→	*INSTRUCTIONS FINSTRUCTIONS*
FINSTRUCTIONS	→	*epsilon*
	|	*BLOCK_CASE*
SHORTHAND	→	*(CONDITION) ? INSTRUCTION :INSTRUCTION*
VAR_DECLARATION	→	*const TYPE IDS_CONST*
	|	*let VARS2*

VARS2	$\rightarrow$	$id\ \ Fid1$
$Fid1$	$\rightarrow$	*VARS_TYPE FVARS_TYPE*
FVARS_TYPE	$\rightarrow$	$symbole\_aff\ \ Fsymbole\_aff1$
	$\mid$	, *VARS2*
	$\mid$	$epsilon$
$Fsymbole\_aff1$	$\rightarrow$	*EXPRESSION FEXPRESSION1*
FEXPRESSION1	$\rightarrow$	, *VARS2*
	$\mid$	$epsilon$
VARS_TYPE	$\rightarrow$	$:\ type$
	$\mid$	$is\ \ type$
IDS_CONST	$\rightarrow$	$id\ \ Fid2$
$Fid2$	$\rightarrow$	$symbole\_aff\ \ Fsymbole\_aff$
$Fsymbole\_aff$	$\rightarrow$	*EXPRESSION FEXPRESSION*
FEXPRESSION	$\rightarrow$	, *IDS_CONST*
	$\mid$	$epsilon$