//Régles syntaxiques : + nouveau lexical

```
PROGRAM ::= program ID ; BLOCK .
BLOCK ::= CONSTS VARS TABS INSTS
CONSTS ::= const ID = CONSTVAL { ,ID = CONSTVAL}  ; | ε
CONSTVAL ::= STRING | REAL | NUM | BOOLEAN
STRING_TOKEN ,REAL_TOKEN , NUM_TOKEN

VARS ::= var IDLIST
IDLIST ::= IDLIST  | ID : TYPE ;  | ε
TYPE ::= integer | string | real | boolean | char | array
Tinteger_TOKEN | Tstring_TOKEN | Treal_TOKEN | Tboolean_TOKEN |Tchar_TOKEN |
Tarray_TOKEN

ARRAY ::= array [ ELEMENT { , ELEMENT } ] of TYPE ;
ARRAY_TOKEN , CO_TEOKEN , CF_TOKEN , OF_TOKEN ,
ELEMENT ::= NUM | STRING | REAL | NUM : NUM
TODPT_TOKEN ..
INSTS ::= begin INST { ; INST } end
INST ::= INSTS | AFFEC | SI | TANTQUE | ECRIRE | LIRE | FOR | CASE| ε
AFFEC ::= ID := EXPR

SI ::= if COND then INST { else IF_STATEMENT } | if COND then INST
ELSE_TOKEN ,
IF_STATEMENT ::= if COND then INST { else IF_STATEMENT } | INST

FOR ::= for ID := EXPR to EXPR do INST | for ID := EXPR downto EXPR do INST
FOR_TOKEN , TO_TOKEN , DOWNTO_TOKEN
CASE ::= case EXPR of CASE_BRANCH {CASE_BRANCH  ; } ;  end
CASE_TOKEN ,
CASE_BRANCH ::= VALUE : INST
DPT_TOKEN
VALUE ::= NUM  | STRING | BOOLEAN | ID

TANTQUE ::= while COND do INST
ECRIRE ::= write ( EXPR { , EXPR } )
LIRE ::= read ( ID { , ID } )
COND ::= EXPR RELOP EXPR
RELOP ::= = | <> | < | > | <= | >=
EXPR ::= TERM { ADDOP TERM }
ADDOP ::= + | -
TERM ::= FACT   { MULOP FACT }
MULOP ::= * | /
FACT ::= ID | NUM | STRING | REAL | Boolean | Array_access | ( EXPR )
Array_access ::= ID[NUM]
BOOL ::= true | false
TRUE_TOKEN , FALSE_TOKEN
STRING ::= ' char{char} '
APT_TOKEN
ID ::= Lettre { Lettre | Chiffre }
NUM  ::= Chiffre { Chiffre }
REAL ::= Chiffre { Chiffre } . Chiffre { Chiffre }
Chiffre ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
char ::= a|b|...|z|A|B|...|Z|0| 1|...|9| special_character
```

//Régles sémentiques :

Règle 0 :

Lors d'une affectation, il est impératif que les deux types soient compatibles :

```
y := 90 ;
x := y ;
tab[9] := x;
z := tab[3];
```

Règle 1 :

Un entier peut être assigné à une variable de type flottant. De même, une variable réelle peut être comparée à un entier :

```
IF qs = 14
qs := 1
```

Règle 2 :

Dans une instruction **CASE**, le type de l'expression doit être compatible avec les valeurs à comparer :

```
CASE j OF
    'aaaa': z := 97;
    'dddd': z := 94;
```

Règle 3 :

L'indice de boucle **FOR** doit être de type entier :

```
FOR x := 1 TO 5 DO z := z + 5;
```

Règle 4 :

Les types doivent être compatibles dans l'expression de la condition du **WHILE** :

```
WHILE x = 45
```

Règle 5 :

Compatibilité des types dans l'expression de la condition du **IF** :

```
IF qs = 14
```

Règle 6 :

Pour les instructions d'accès à un tableau comme **tab[9] := x;** ou **z := tab[3];**, l'indice ne doit pas dépasser la taille du tableau.

Règle 7 :

Compatibilité des types dans EXPRESSION

```
x:=x-7;
 j:=s+r;
```

//Interpreteur:

ADD :additionne le sous-sommet de la pile et le sommet, laisse le résultat au sommet (idem pour SUB, MUL, DIV)

EQL :laisse 1 au sommet de la pile si sous-sommet = sommet, 0 sinon (idem pour NEQ, GTR, LSS, GEQ, LEQ)

PRN :imprime le sommet, dépile
INN :lit un entier, le stocke à l'adresse trouvée au sommet de la pile, dépile
INT c :incrémente de la constante c le pointeur de pile (la constante c peut être négative)

LDI v :empile la valeur v
LDA a :empile l'adresse a

LDV :remplace le sommet par la valeur trouvée à l'adresse indiquée par le sommet (déréférence)
STO :stocke la valeur au sommet à l'adresse indiquée par le sous-sommet, dépile 2 fois

BRN i :branchement inconditionnel à l'instruction i
BZE i :branchement à l'instruction i si le sommet = 0, dépile

LEN i: Calcule la longueur d'un tableau ou d'une chaîne de caractères et stocke le résultat au sommet de la pile.

FLDA a: Charge l'adresse d'une constante flottante (réelle) sur la pile.
SLDA a: Charge l'adresse d'une chaîne de caractères sur la pile.
BLDA a: Charge l'adresse d'une constante booléenne sur la pile.

FLDI v: Charge une constante flottante (réelle) v sur la pile.
SLDI v: Charge une constante chaîne de caractères v sur la pile.
BLDI v: Charge une constante booléenne v sur la pile.

TLDA a: Charge l'adresse d'un type a sur la pile.

ILDA a: Charge l'adresse d'une constante entière a sur la pile.
ILDI v: Charge une constante entière v sur la pile.

EFOR i: Marque le début d'une boucle for avec l'index i.
BFOR i: Marque la fin d'une boucle for avec l'index i.

TABA i: Déclare un tableau avec l'index i.
TABI i: Déclare un index de tableau i.

CASET i: Marque le début d'une section de cas dans une structure de cas avec l'index i.
CASEE i: Marque la fin d'une section de cas dans une structure de cas avec l'index i.

HLT :halte