siunitx

On change les règles liés aux opérations pour intégrer dans ces règles l'associativité à gauche et la priorité des multiplication et division.

On change les règles :

$$
\begin{aligned}
< expr > \quad & \rightarrow < entier > \\
& \rightarrow < caractère > \\
& \rightarrow \text{true} \\
& \rightarrow \text{false} \\
& \rightarrow \text{null} \\
& \rightarrow (< expr >) \\
& \rightarrow < acces > \\
& \rightarrow < expr > < operateur > < expr > \\
& \rightarrow \text{not} < expr > \\
& \rightarrow \text{-} < expr > \\
& \rightarrow \text{new} < ident > \\
& \rightarrow < ident > (< expr >_{,}^{+}) \\
& \rightarrow \text{character ' val} (< expr >)
\end{aligned}
$$

$$
\begin{aligned}
< opérateur > \rightarrow & = \\
& \rightarrow \ / = \\
& \rightarrow < \\
& \rightarrow <= \\
& \rightarrow > \\
& \rightarrow >= \\
& \rightarrow \ + \\
& \rightarrow \ - \\
& \rightarrow \ * \\
& \rightarrow \ / \\
& \rightarrow \ rem
\end{aligned}
$$

Par :

$$
\begin{array}{ll}
< expr > & \rightarrow < expr > + \text{T} \\
& \rightarrow < expr > \text{- T} \\
& \rightarrow \text{T} \\
 & \\
\text{T} & \rightarrow \text{T * F} \\
& \rightarrow \text{T / F} \\
& \rightarrow \text{F} \\
 & \\
\text{F} & \rightarrow \text{P} \\
& \rightarrow \text{-P} \\
& \rightarrow \text{not P} \\
 & \\
\text{P} & \rightarrow < entier > \\
& \rightarrow < caractère > \\
& \rightarrow true \\
& \rightarrow false \\
& \rightarrow null \\
& \rightarrow < acces > \\
& \rightarrow new < ident > \\
& \rightarrow < ident > (< expr >^{+}_{,}) \\
& \rightarrow character\,' val\ (< expr >) \\
& \rightarrow < expr > < operateur > < expr > \\
& \rightarrow (< expr >) \\
 & \\
< opérateur > & \rightarrow = \\
& \rightarrow\ / = \\
& \rightarrow < \\
& \rightarrow <= \\
& \rightarrow > \\
& \rightarrow >= \\
& \rightarrow\ rem
\end{array}
$$

On développe la grammaire pour obtenir appliquer la dérécursivation :

$< fichier > \rightarrow$ with Ada.Text_IO; use Ada.Text_IO; procedure $< ident >$ is begin $< instr >^+$ end; EOF
$\rightarrow$ with Ada.Text_IO; use Ada.Text_IO; procedure $< ident >$ is $< decl >^+$ begin $< instr >^+$ end; EOF
$\rightarrow$ with Ada.Text_IO; use Ada.Text_IO; procedure $< ident >$ is begin $< instr >^+$ end $< ident >$; EOF
$\rightarrow$ with Ada.Text_IO; use Ada.Text_IO; procedure $< ident >$ is $< decl >^+$ begin $< instr >^+$ end $< ident >$; EOF

$< decl > \rightarrow$ type $< ident >$;
$\rightarrow$ type $< ident >$ is access $< ident >$;
$\rightarrow$ type $< ident >$ is record $< champs >^+$ end record;
$\rightarrow$ type $< ident >^+_{,} : < type >$;
$\rightarrow$ type $< ident >^+_{,} : < type >$ (:=$< expr >$);
$\rightarrow$ procedure $< ident >$ is begin $< instr >^+$ end;
$\rightarrow$ procedure $< ident >$ is begin $< instr >^+$ end $< ident >$;
$\rightarrow$ procedure $< ident >$ is $< decl >^+$ begin $< instr >^+$ end;
$\rightarrow$ procedure $< ident >$ is $< decl >^+$ begin $< instr >^+$ end $< ident >$;
$\rightarrow$ procedure $< ident > < param >$ is begin $< instr >^+$ end;
$\rightarrow$ procedure $< ident > < param >$ is begin $< instr >^+$ end $< ident >$;
$\rightarrow$ procedure $< ident > < param >$ is $< decl >^+$ begin $< instr >^+$ end;
$\rightarrow$ procedure $< ident > < param >$ is $< decl >^+$ begin $< instr >^+$ end $< ident >$;
$\rightarrow$ function $< ident >$ return $< type >$ is begin $< instr >^+$ end;
$\rightarrow$ function $< ident >$ return $< type >$ is begin $< instr >^+$ end $< ident >$;
$\rightarrow$ function $< ident >$ return $< type >$ is $< decl >^+$ begin $< instr >^+$ end;
$\rightarrow$ function $< ident >$ return $< type >$ is $< decl >^+$ begin $< instr >^+$ end $< ident >$;
$\rightarrow$ function $< ident > < param >$ return $< type >$ is begin $< instr >^+$ end;
$\rightarrow$ function $< ident > < param >$ return $< type >$ is begin $< instr >^+$ end $< ident >$;
$\rightarrow$ function $< ident > < param >$ return $< type >$ is $< decl >^+$ begin $< instr >^+$ end;
$\rightarrow$ function $< ident > < param >$ return $< type >$ is $< decl >^+$ begin $< instr >^+$ end $< ident >$;

$< champs > \rightarrow < ident >^+_{,} : < type >$;

$< type > \rightarrow < ident >$
$\rightarrow$ access $< ident >$

$< params > \rightarrow (< param >^+_{;})$

$< param > \rightarrow < ident >^+_{,} : < type >$
$\rightarrow < ident >^+_{,} : < mode > < type >$

$< mode > \rightarrow$ in
$\rightarrow$ in out

$< expr >$ $\rightarrow < expr > +$ T
$\rightarrow < expr >$ - T
$\rightarrow$ T

T $\rightarrow$ T * F
$\rightarrow$ T / F
$\rightarrow$ F

F $\rightarrow$ P
$\rightarrow$ -P
$\rightarrow$ not P

P $\rightarrow < entier >$
$\rightarrow < caractère >$
$\rightarrow true$
$\rightarrow false$
$\rightarrow null$
$\rightarrow < acces >$
$\rightarrow new < ident >$
$\rightarrow < ident > (< expr >^{+}_{,})$
$\rightarrow character' val (< expr >)$
$\rightarrow < expr > < operateur > < expr >$
$\rightarrow (< expr >)$

$< opérateur > \rightarrow =$
$\rightarrow /=$
$\rightarrow <$
$\rightarrow <=$
$\rightarrow >$
$\rightarrow >=$
$\rightarrow rem$

$< instr >$ $\rightarrow < acces > := < expr >;$

$\rightarrow < ident >;$

$\rightarrow < ident > (< expr >^{+}_{,});$

$\rightarrow$ return;

$\rightarrow$ return $< expr >;$

$\rightarrow$ begin $< instr >^{+}$ end;

$\rightarrow$ if $< expr >$ then $< instr >^{+}$ end if;

$\rightarrow$ if $< expr >$ then $< instr >^{+}$ (else $< instr >^{+}$) end if;

$\rightarrow$ if $< expr >$ then $< instr >^{+} < elsif >^{+}$ end if;

$\rightarrow$ if $< expr >$ then $< instr >^{+} < elsif >^{+}$ (else $< instr >^{+}$) end if;

$\rightarrow$ for $< ident >$ in $< expr >$ .. $< expr >$ loop $< instr >^{+}$ end loop;

$\rightarrow$ for $< ident >$ in reverse $< expr >$ .. $< expr >$ loop $< instr >^{+}$ end loop;

$\rightarrow$ while $< expr >$ loop $< instr >^{+}$ end loop;

$< acces >$ $\rightarrow < ident >$

$\rightarrow < expr > . < ident >$

$< instr >^{+}$ $\rightarrow < instr > < instr >^{+}$

$\rightarrow < instr >$

$< decl >^{+}$ $\rightarrow < decl > < decl >^{+}$

$\rightarrow < decl >$

$< champs >^{+}$ $\rightarrow < champs > < champs >^{+}$

$\rightarrow < champs >$

$< ident >^{+}_{,}$ $\rightarrow < ident > , < ident >^{+}_{,}$

$\rightarrow < ident >;$

$< param >^{+}_{;}$ $\rightarrow < param > ; < param >^{+}_{;}$

$\rightarrow < param >$

$< expr >^{+}_{,}$ $\rightarrow < expr > , < expr >^{+}_{,}$

$\rightarrow < expr >$

$< elsif >^{+}$ $\rightarrow$ elsif $< expr >$ then $< instr >^{+} < elsif >^{+}$

$\rightarrow$ elsif $< expr >$ then $< instr >^{+}$

On enlève les récursivités à gauche qui posent problème. On choisit la numérotation suivante pour l'algorithme :

A_1	$< expr >$
A_2	T
A_3	F
A_4	P
A_5	$< instr >$
A_6	$< acces >$

$$
\begin{aligned}
< expr > \quad & \rightarrow T < expr >_{recur} \\
< expr >_{recur} \quad & \rightarrow +T < expr >_{recur} \\
& \rightarrow -T < expr >_{recur} \\
& \rightarrow \wedge \\
\mathrm{T} \quad & \rightarrow F\ T_{recur} \\
T_{recur} \quad & \rightarrow *F\ T_{recur} \\
& \rightarrow /F\ T_{recur} \\
& \rightarrow \wedge \\
\mathrm{F} \quad & \rightarrow P \\
& \rightarrow -P \\
& \rightarrow not\ P \\
\mathrm{P} \quad & \rightarrow -P\ T_{recur} < expr >_{recur} < opérateur >< expr > P_{recur} \\
& \rightarrow not\ P\ T_{recur} < expr >_{recur} < opérateur >< expr > P_{recur} \\
& \rightarrow < entier > P_{recur} \\
& \rightarrow < caractère > P_{recur} \\
& \rightarrow true\ P_{recur} \\
& \rightarrow false\ P_{recur} \\
& \rightarrow null\ P_{recur} \\
& \rightarrow < acces > P_{recur} \\
& \rightarrow new < ident > P_{recur} \\
& \rightarrow < ident > (< expr >_{,}^{+})\ P_{recur} \\
& \rightarrow character\ '\ val\ (< expr >) P_{recur} \\
& \rightarrow (< expr >)\ P_{recur} \\
P_{recur} \quad & \rightarrow T_{recur} < expr >_{recur} < opérateur >< expr > P_{recur} \\
& \rightarrow \wedge
\end{aligned}
$$

$< instr >$ $\rightarrow < acces > := < expr >;$

$\rightarrow < ident >;$

$\rightarrow < ident > (< expr >^{+}_{,});$

$\rightarrow$ return;

$\rightarrow$ return $< expr >;$

$\rightarrow$ begin $< instr >^{+}$ end;

$\rightarrow$ if $< expr >$ then $< instr >^{+}$ end if;

$\rightarrow$ if $< expr >$ then $< instr >^{+}$ (else $< instr >^{+}$) end if;

$\rightarrow$ if $< expr >$ then $< instr >^{+} < elsif >^{+}$ end if;

$\rightarrow$ if $< expr >$ then $< instr >^{+} < elsif >^{+}$ (else $< instr >^{+}$) end if;

$\rightarrow$ for $< ident >$ in $< expr >$ .. $< expr >$ loop $< instr >^{+}$ end loop;

$\rightarrow$ for $< ident >$ in reverse $< expr >$ .. $< expr >$ loop $< instr >^{+}$ end loop;

$\rightarrow$ while $< expr >$ loop $< instr >^{+}$ end loop;

$< acces >$ $\rightarrow -P\ T_{recur} < expr >_{recur} < opérateur >< expr > P_{recur}\ T_{recur} < expr >_{recur} . < ident >< acces >_{recur}$

$\rightarrow not\ P\ T_{recur} < expr >_{recur} < opérateur >< expr > P_{recur}\ T_{recur} < expr >_{recur} . < ident >< acces >_{recur}$

$\rightarrow < entier > P_{recur}\ T_{recur} < expr >_{recur} . < ident >< acces >_{recur}$

$\rightarrow < caractère > P_{recur}\ T_{recur} < expr >_{recur} . < ident >< acces >_{recur}$

$\rightarrow true\ P_{recur}\ T_{recur} < expr >_{recur} . < ident >< acces >_{recur}$

$\rightarrow false\ P_{recur}\ T_{recur} < expr >_{recur} . < ident >< acces >_{recur}$

$\rightarrow null\ P_{recur}\ T_{recur} < expr >_{recur} . < ident >< acces >_{recur}$

$\rightarrow new < ident > P_{recur}\ T_{recur} < expr >_{recur} . < ident >< acces >_{recur}$

$\rightarrow < ident > (< expr >^{+}_{,})\ P_{recur}\ T_{recur} < expr >_{recur} . < ident >< acces >_{recur}$

$\rightarrow character'\ val\ (< expr >) P_{recur}\ T_{recur} < expr >_{recur} . < ident >< acces >_{recur}$

$\rightarrow (< expr >)\ P_{recur}\ T_{recur} < expr >_{recur} . < ident >< acces >_{recur}$

$\rightarrow -P\ T_{recur} < expr >_{recur} . < ident >< acces >_{recur}$

$\rightarrow not\ P\ T_{recur} < expr >_{recur} . < ident >< acces >_{recur}$

$< acces >_{recur}$ $\rightarrow < acces > T_{recur} < expr >_{recur} . < ident >< acces >_{recur}$

$\rightarrow \wedge$

On injecte les règles de ¡acces¿ dans la règle
On factorise la grammaire et on numérote les règles

$r_1 :< fichier > \rightarrow$ with Ada.Text_IO; use Ada.Text_IO; procedure $< ident >$ is $< fichier >_2$

$r_2 :< fichier >_2 \rightarrow$ begin $< instr >^+$ end $< fichier >_3$
$r_3 : \rightarrow < decl >^+$ begin $< instr >^+$ end $< fichier >_3$

$r_4 :< fichier >_3 \rightarrow$; EOF
$r_5 : \rightarrow < ident >$; EOF

$r_6 :< decl > \rightarrow$ type $< ident > < decl >_{11}$
$r_7 : \rightarrow$ procedure $< ident > < decl >_{21}$
$r_8 : \rightarrow$ function $< ident > < decl >_{31}$
$r_9 : \rightarrow < ident >_{,}^+ : < type > < decl >_{12}$

$r_{10} :< decl >_{11} \rightarrow$;
$r_{11} : \rightarrow$ is $< decl >_{13}$

$r_{12} :< decl >_{12} \rightarrow$;
$r_{13} : \rightarrow :=< expr >$;

$r_{14} :< decl >_{13} \rightarrow$ access $< ident >$;
$r_{15} : \rightarrow$ record $< champs >^+$ end record;

$r_{16} :< decl >_{21} \rightarrow$ is $< decl >_{22}$
$r_{17} : \rightarrow < params >$ is $< decl >_{22}$

$r_{18} :< decl >_{22} \rightarrow$ begin $< instr >^+$ end $< decl >_{23}$
$r_{19} : \rightarrow < decl >^+$ begin $< instr >^+$ end $< decl >_{23}$

$r_{20} :< decl >_{23} \rightarrow$;
$r_{21} : \rightarrow < ident >$;

$r_{22} :< decl >_{31} \rightarrow$ return $< type >$ is $< decl >_{22}$
$r_{23} : \rightarrow < params >$ return $< type >$ is $< decl >_{22}$

$r_{24} :< champs > \rightarrow < ident >_{,}^+ : < type >$;

$r_{25} :< type > \rightarrow < ident >$
$r_{26} : \rightarrow$ access $< ident >$

$r_{27} :< params > \rightarrow (< param >_{;}^+)$

$r_{28} :< param > \quad \rightarrow < ident >_{,}^{+} : < param >_2$

$r_{29} :< param >_2 \quad \rightarrow < type >$

$r_{30} : \quad \rightarrow < mode > < type >$

$r_{31} :< mode > \quad \rightarrow \text{in} < mode >_1$

$r_{32} :< mode >_1 \quad \rightarrow \text{out}$

$r_{33} : \quad \rightarrow \wedge$

$r_{34} :< expr > \quad \rightarrow T < expr >_{recur}$

$r_{35} :< expr >_{recur} \rightarrow +T < expr >_{recur}$

$r_{36} : \quad \rightarrow -T < expr >_{recur}$

$r_{37} : \quad \rightarrow \wedge$

$r_{38} : T \quad \rightarrow F\ T_{recur}$

$r_{39} : T_{recur} \quad \rightarrow *F\ T_{recur}$

$r_{40} : \quad \rightarrow /F\ T_{recur}$

$r_{41} : \quad \rightarrow \wedge$

$r_{42} : F \quad \rightarrow P$

$r_{43} : \quad \rightarrow -P$

$r_{44} : \quad \rightarrow not\ P$

$r_{45} : P \quad \rightarrow -P\ T_{recur} < expr >_{recur} < opérateur >< expr > P_{recur}$

$r_{46} : \quad \rightarrow not\ P\ T_{recur} < expr >_{recur} < opérateur >< expr > P_{recur}$

$r_{47} : \quad \rightarrow < entier > P_{recur}$

$r_{48} : \quad \rightarrow < caractère > P_{recur}$

$r_{49} : \quad \rightarrow true\ P_{recur}$

$r_{50} : \quad \rightarrow false\ P_{recur}$

$r_{51} : \quad \rightarrow null\ P_{recur}$

$r_{52} : \quad \rightarrow < acces > P_{recur}$

$r_{53} : \quad \rightarrow new < ident > P_{recur}$

$r_{54} : \quad \rightarrow < ident > (< expr >_{,}^{+})\ P_{recur}$

$r_{55} : \quad \rightarrow character\ '\ val\ (< expr >) P_{recur}$

$r_{56} : \quad \rightarrow (< expr >)\ P_{recur}$

$r_{57} : P_{recur} \quad \rightarrow T_{recur} < expr >_{recur} < opérateur >< expr > P_{recur}$

$r_{58} : \quad \rightarrow \wedge$

r_{59} :$< instr >$ $\rightarrow < acces > := < expr >;$

r_{60} : $\rightarrow < ident > < instr >_1$

r_{61} : $\rightarrow$ return $< instr >_2$

r_{62} : $\rightarrow$ begin $< instr >^+$ end;

r_{63} : $\rightarrow$ if $< expr >$ then $< instr >^+ < instr >_3$

r_{64} : $\rightarrow$ for $< ident >$ in $< instr >_5$

r_{65} : $\rightarrow$ while $< expr >$ loop $< instr >^+$ end loop;

r_{54bis} : $\rightarrow$ put $(< expr >)$;

r_{66} :$< instr >_1$ $\rightarrow$;

r_{67} : $\rightarrow (< expr >^+_,);$

r_{68} :$< instr >_2$ $\rightarrow$;

r_{69} : $\rightarrow < expr >;$

r_{70} :$< instr >_3$ $\rightarrow$ end if;

r_{71} : $\rightarrow$ else $< instr >^+$ end if;

r_{72} : $\rightarrow < elsif >^+ < instr >_4$

r_{73} :$< instr >_4$ $\rightarrow$ end if;

r_{74} : $\rightarrow$ else $< instr >^+$ end if;

r_{75} :$< instr >_5$ $\rightarrow < expr > .. < expr >$ loop $< instr >^+$ end loop;

r_{76} : $\rightarrow$ reverse $< expr > .. < expr >$ loop $< instr >^+$ end loop;

r_{77} :$< opérateur >$ $\rightarrow =$

r_{78} : $\rightarrow /=$

r_{79} : $\rightarrow <$

r_{80} : $\rightarrow <=$

r_{81} : $\rightarrow >$

r_{82} : $\rightarrow >=$

r_{83} : $\rightarrow$ rem

r_{84} :$< acces >$ $\rightarrow -P\ T_{recur} < expr >_{recur} < acces >_1$

r_{85} : $\rightarrow not\ P\ T_{recur} < expr >_{recur} < acces >_1$

r_{86} : $\rightarrow < entier > P_{recur}\ T_{recur} < expr >_{recur} . < ident >< acces >_{recur}$

r_{87} : $\rightarrow < caractère > P_{recur}\ T_{recur} < expr >_{recur} . < ident >< acces >_{recur}$

r_{88} : $\rightarrow true\ P_{recur}\ T_{recur} < expr >_{recur} . < ident >< acces >_{recur}$

r_{89} : $\rightarrow false\ P_{recur}\ T_{recur} < expr >_{recur} . < ident >< acces >_{recur}$

r_{90} : $\rightarrow null\ P_{recur}\ T_{recur} < expr >_{recur} . < ident >< acces >_{recur}$

r_{91} : $\rightarrow < acces > P_{recur}\ T_{recur} < expr >_{recur} . < ident >< acces >_{recur}$

r_{92} : $\rightarrow new < ident > P_{recur}\ T_{recur} < expr >_{recur} . < ident >< acces >_{recur}$

r_{93} : $\rightarrow < ident > (< expr >^+_,)\ P_{recur}\ T_{recur} < expr >_{recur} . < ident >< acces >_{recur}$

r_{94} : $\rightarrow character'\ val\ (< expr >) P_{recur}\ T_{recur} < expr >_{recur} . < ident >< acces >_{recur}$

r_{95} : $\rightarrow (< expr >)\ P_{recur}\ T_{recur} < expr >_{recur} . < ident >< acces >_{recur}$

$r_{96} :< acces >_1 \quad \rightarrow < opérateur >< expr > P_{recur}\ T_{recur} < expr >_{recur} . < ident >< acces >_{recur}$

$r_{97} : \quad \rightarrow . < ident >< acces >_{recur}$

$r_{98} :< acces >_{recur} \rightarrow < acces > T_{recur} < expr >_{recur} . < ident >< acces >_{recur}$

$r_{99} : \quad \rightarrow \wedge$

$r_{100} :< instr >^+ \quad \rightarrow < instr > < instr >^+_1$

$r_{101} :< instr >^+_1 \quad \rightarrow < instr >^+$

$r_{102} : \quad \rightarrow \wedge$

$r_{103} :< decl >^+ \quad \rightarrow < decl > < decl >^+_1$

$r_{104} :< decl >^+_1 \quad \rightarrow < decl >^+$

$r_{105} : \quad \rightarrow \wedge$

$r_{106} :< champs >^+ \rightarrow < champs > < champs >^+_1$

$r_{107} :< champs >^+_1 \rightarrow < champs >^+$

$r_{108} : \quad \rightarrow \wedge$

$r_{109} :< ident >^+_, \quad \rightarrow < ident > < ident >^+_{,1}$

$r_{110} :< ident >^+_{,1} \quad \rightarrow , < ident >^+_,$

$r_{111} : \quad \rightarrow \wedge$

$r_{112} :< param >^+_; \quad \rightarrow < param > < param >^+_{;1}$

$r_{113} :< param >^+_{;1} \quad \rightarrow ; < param >^+_;$

$r_{114} : \quad \rightarrow \wedge$

$r_{115} :< expr >^+_, \quad \rightarrow < expr > < expr >^+_{,1}$

$r_{116} :< expr >^+_{,1} \quad \rightarrow , < expr >^+_,$

$r_{117} : \quad \rightarrow \wedge$

$r_{118} :< elsif >^+ \quad \rightarrow \text{elsif} < expr > \text{then} < instr >^+ < elsif >^+_1$

$r_{119} :< elsif >^+_1 \quad \rightarrow < elsif >^+$

$r_{120} : \quad \rightarrow \wedge$

$$P_{\wedge}(G) = \{< mode >_1, < expr >_{recur}, < acces >_{recur}, < instr >^+_1, < decl >^+_1, < champs >^+_1, < ident >^+_{,1}, < param >^+_{;1}, < expr >^+_{,1}, < elsif >^+_1\}$$

Non terminal gauche	Règle	Symbole Directeur
$< fichier >$	r_1	with
$< fichier >_2$	r_2	begin
$< fichier >_2$	r_3	type , procedure , function
$< fichier >_3$	r_4	;
$< fichier >_3$	r_5	$< ident >$
$< decl >$	r_6	type
$< decl >$	r_7	procedure
$< decl >$	r_8	function
$< decl >$	r_9	$< ident >$
$< decl >_{11}$	r_{10}	;
$< decl >_{11}$	r_{11}	is
$< decl >_{12}$	r_{12}	;
$< decl >_{12}$	r_{13}	(
$< decl >_{13}$	r_{14}	access
$< decl >_{13}$	r_{15}	record
$< decl >_{21}$	r_{16}	is
$< decl >_{21}$	r_{17}	$< ident >$
$< decl >_{22}$	r_{18}	begin
$< decl >_{22}$	r_{19}	type , procedure , function
$< decl >_{23}$	r_{20}	;
$< decl >_{23}$	r_{21}	$< ident >$
$< decl >_{31}$	r_{22}	return
$< decl >_{31}$	r_{23}	$< ident >$
$< champs >$	r_{24}	$< ident >$
$< type >$	r_{25}	$< ident >$
$< type >$	r_{26}	access
$< params >$	r_{27}	(
$< param >$	r_{28}	$< ident >$
$< param >_2$	r_{29}	$< ident >$, access
$< param >_2$	r_{30}	in
$< mode >$	r_{31}	in
$< mode >_1$	r_{32}	out
$< mode >_1$	r_{33}	$< ident >$, access
$< expr >$	r_{34}	$< entier >$
$< expr >$	r_{35}	$< caractère >$
$< expr >$	r_{36}	true
$< expr >$	r_{37}	false
$< expr >$	r_{38}	null
$< expr >$	r_{39}	(
$< expr >$	r_{40}	$< ident >$, $< entier >$, $< caractère >$, true, false, null, (, not, -, new, character
$< expr >$	r_{41}	not
$< expr >$	r_{42}	-
$< expr >$	r_{43}	new
$< expr >$	r_{44}	$< ident >$
$< expr >$	r_{45}	character
$< expr >_{recur}$	r_{46}	=, /=, <, <=, >, >=, +, -, *, /, rem
$< expr >_{recur}$	r_{47}	), =, /=, <, <=, >, >=, +, -, *, /, rem, ,, ;, then, .., loop, .
$< instr >$	r_{48}	$< ident >$, $< entier >$, $< caractère >$, true, false, null, (, not, -, new, character
$< instr >$	r_{49}	$< ident >$

$< instr >$	r_{50}	return
$< instr >$	r_{51}	begin
$< instr >$	r_{52}	if
$< instr >$	r_{53}	for
$< instr >$	r_{54}	while
$< instr >_1$	r_{55}	;
$< instr >_1$	r_{56}	(
$< instr >_2$	r_{57}	;
$< instr >_2$	r_{58}	$< entier >$, $< caractère >$, true , false , null , (, $< ident >$, not, -, new, character
$< instr >_3$	r_{59}	end
$< instr >_3$	r_{60}	(
$< instr >_3$	r_{61}	(
$< instr >_4$	r_{62}	end
$< instr >_4$	r_{63}	(
$< opérateur >$	r_{64}	=
$< opérateur >$	r_{65}	/=
$< opérateur >$	r_{66}	<
$< opérateur >$	r_{67}	<=
$< opérateur >$	r_{68}	>
$< opérateur >$	r_{69}	>=
$< opérateur >$	r_{70}	+
$< opérateur >$	r_{71}	-
$< opérateur >$	r_{72}	*
$< opérateur >$	r_{73}	/
$< opérateur >$	r_{74}	rem
$< acces >$	r_{75}	$< ident >$
$< acces >$	r_{76}	$< entier >$
$< acces >$	r_{77}	$< caratère >$
$< acces >$	r_{78}	true
$< acces >$	r_{79}	false
$< acces >$	r_{80}	null
$< acces >$	r_{81}	(
$< acces >$	r_{82}	not
$< acces >$	r_{83}	-
$< acces >$	r_{84}	new
$< acces >$	r_{85}	$< ident >$
$< acces >$	r_{86}	character
$< acces >_{recur}$	r_{87}	.
$< acces >_{recur}$	r_{88}	=, /=, <, <=, >, >=, +, -, *, /, rem, :
$< instr >^+$	r_{89}	$< ident >$, $< entier >$, $< caractère >$, true, false, null, (, not, -, new, character, return, begin, if, for, while
$< instr >_1^+$	r_{90}	$< ident >$, $< entier >$, $< caractère >$, true, false, null, (, not, -, new, character, return, begin, if, for, while
$< instr >_1^+$	r_{91}	end, (,)
$< decl >^+$	r_{92}	type, procedure, function
$< decl >_1^+$	r_{93}	type, procedure, function
$< decl >_1^+$	r_{94}	begin
$< champs >^+$	r_{95}	$< ident >$
$< champs >_1^+$	r_{96}	$< ident >$
$< champs >_1^+$	r_{97}	end
$< ident >^+$	r_{98}	$< ident >$
$< ident >_{,1}^+$	r_{99}	,

$< ident >^{+}_{,1}$	r_{100}	:
$< param >^{+}_{;}$	r_{101}	$< ident >$
$< param >^{+}_{;1}$	r_{102}	;
$< param >^{+}_{;1}$	r_{103}	)
$< expr >^{+}_{,}$	r_{104}	$< ident >$, $< entier >$, $< caractère >$, true, false, null, (, not, -, new, character
$< expr >^{+}_{,1}$	r_{105}	,
$< expr >^{+}_{,1}$	r_{106}	)
$< elsif >^{+}$	r_{107}	(
$< elsif >^{+}_{1}$	r_{108}	(
$< elsif >^{+}_{1}$	r_{109}	end, (