Compilateur TCP : le langage hepiaL

Voici la grammaire utilisée pour le projet de TCP 2019 dans lequel vous allez créer un compilateur de hepiaL vers la machine virtuelle JAVA (JVM). Ce langage est considérablement réduit mais intègre la base des langages procéduraux (sauf procédures et fonctions). Sa syntaxe est simple. Les règles sont numérotées. Les non-terminaux sont écrits en majuscules. Le caractère * suivant un non-terminal indique que celui-ci peut être répété 0, 1 ou n fois. Le caractère + suivant un non-terminal indique que celui-ci peut être répété 1 ou n fois.
Les unités lexicales correspondant à des mots-clés sont écrites en minuscules, en italique. Les unités lexicales génériques sont écrites en minuscules. Elles sont définies de la façon suivante :

- ***ident*** est une suite non vide de lettres et/ou de chiffres commençant par une lettre : [a-zA-Z] [a-zA-Z0-9]*
- ***constanteEnt*** est une suite non vide de chiffres : [0-9]+
- ***constanteChaine*** est une suite de caractères entourés par des guillemets ;
- pour apparaître dans la chaîne, le guillemet doit être doublé :
  [\"](["^\"]|\"\")*[\"]
- les commentaires commencent par // et se terminent à la fin de la ligne

GRAMMAIRE HEPIAL

~~(1)~~	~~AXIOME~~	~~->~~	~~PROGRAMME~~
(2)	PROGRAMME	->	ENTETE DECLA* ***'debutprg'*** CORPS ***'finprg'***
(3)	ENTETE	->	***'programme'*** ident
(4)	DECLA	->	DECLAVAR
		|	DECLACONST
(5)	DECLAVAR	->	TYPE LIDENT **';'**
(6)	LIDENT	->	**ident**
		|	LIDENT **',' ident**
(7)	DECLACONST	->	***'constante'*** TYPE **ident '='** EXPR ';'
(8)	TYPE	->	***'entier'*** | ***'booleen'***
(9)	CORPS	->	INSTR *
(10)	INSTR	->	AFFECTATION | ECRIRE | LIRE | CONDITION | TANTQUE | POUR

(11)	LIRE	->	***'lire'*** ident **' ;'**
(12)	ECRIRE	->	***'ecrire'*** EXPR **' ;'**
		|	***'ecrire'*** **constanteChaine ' ;'**
(13)	AFFECTATION	->	***ident*** '=' EXPR ';'
(15)	CONDITION	->	**'*si*'** EXPR **'*alors*'** CORPS **'*sinon*'** CORPS **'*finsi*'**
(16)	TANTQUE	->	**'*tantque*'** EXPR **'*faire*'** CORPS **'*fintantque*'**
(17)	POUR	->	**'*pour*' ident '*allantde*'** EXPR **'*a*'** EXPR **'*faire*'** CORPS **'*finpour*'**
(18)	EXPR	->	EXPR OPEBIN EXPR
		|	OPEUN EXPR
		|	'(' EXPR ')'
		|	OPERANDE
(19)	OPERANDE	->	**ident | constanteEnt |** ***'vrai'*** **|** ***'faux'***
(20)	OPEBIN	->	**'+' | '-' | '*' | '/' | '==' | '<>'**
		|	**'<' | '>' | '<=' | '>=' | '*et*' | '*ou*'**
(21)	OPEUN	->	**'~' | '*non*'**

A. Règles relatives à la sémantique des déclarations.

~~A.1 Le corps du programme et une fonction définissent un bloc.~~

~~A.2 La portée d'une déclaration recouvre l'ensemble du bloc qui la contient en dehors des blocs imbriqués contenant la déclaration du même identificateur.~~

A.3 Toute double déclaration de variable est interdite.

A.4 Un type de base est un type primitif comme entier ou booléen.

B. Règles relatives aux types de données.

B.1 Une variable de type primitif (entier ou booléen) est représentée en mémoire par sa valeur;

B.2 Les variables ne sont pas initialisées.

B.3 Les types primitifs ne sont compatibles qu'avec eux-mêmes.

C. Règles relatives à la sémantique des opérateurs.

C.1 Les opérateurs **+,-,*** et **/** acceptent des opérandes entiers et sont à résultat entier; ils ont la sémantique des opérateurs arithmétiques classiques; la division par 0 est interdite.

C.2 Les opérateurs **et, ou** et **non** acceptent des opérandes booléens ; ils ont la sémantique des opérateurs logiques court-circuits.

C.3 Les opérateurs relationnels **<, >, <=, >=** n'acceptent que des booléens entiers; L'expression est de type booléen; ces opérateurs utilisent la relation d'ordre sur les entiers relatifs.

C.4 Les opérateurs **+, -** et **ou** ont la plus faible priorité, inférieure à celle des opérateurs ***, /**, et **et**, inférieure à celle des opérateurs **==, <>, <=, >=, < et >** ; ils sont tous associées de gauche à droite.

C.5 Les deux opérandes des opérateurs de comparaison == et <> sont de même type; l'expression est de type booléen. Deux opérandes de type primitif sont égaux si leurs valeurs le sont.

C.6 L'opérande gauche d'un opérateur binaire est toujours calculée avant son opérande droit.

C.7 L'opérateur conditionnel accepte un premier opérande booléen. Si les deux derniers opérandes sont entiers (resp. booléens), le type de l'expression est entier (resp. booléens).

D. Règles relatives à l'identification des variables et des fonctions.

D.1 L'identification d'une variable se fait à partir du bloc courant. Le type de l'expression est le type de déclaration de la variable. La valeur de l'expression est la valeur de la variable à ce moment de l'exécution.

E. Règles relatives à la sémantique des instructions.

E.1 Dans la première forme d'instruction d'affichage, l'expression est de type primitif (entier ou booléen). L'exécution de cette instruction provoque l'affichage de la valeur de l'expression sur une ligne de la sortie standard. Dans sa seconde forme, l'exécution provoque l'affichage de la chaîne (sans les guillemets) sur une ligne de la sortie standard.

E.2 Dans la lecture, l'identificateur est celui d'une variable de type entier. L'exécution de cette instruction provoque la lecture d'une valeur entière sur l'entrée standard, valeur affectée ensuite à la variable.

E.3 Dans une affectation, la notation d'accès est d'un type compatible avec celui de l'expression. L'exécution de cette instruction provoque l'affectation de la variable avec la valeur de l'expression.

E.4 Dans une itération conditionnelle (boucle tant que), l'expression est de type booléen et les instructions sont correctes. A l'exécution, l'expression booléenne est évaluée; si elle prend la valeur *faux*, l'exécution se poursuit en séquence, après l'itération conditionnelle; si elle prend la valeur *vrai*, les instructions sont exécutées, et l'exécution se poursuit à l'évaluation de l'expression booléenne.

E.5 Dans une itération simple (boucle pour), l'identificateur est celui d'une variable de type entier, appelée variable d'itération; les deux expressions sont de type entier, elles constituent le domaine d'itération; les instructions sont correctes. A l'exécution, les valeurs définissant le domaine ne sont calculées qu'une seule fois au début; la variable d'itération est initialisée avec la valeur de la borne inférieure du domaine; si la variable d'itération dépasse la borne supérieure, l'exécution se poursuit en séquence, après l'itération simple; sinon, les instructions itérées sont exécutées, la variable d'itération est incrémentée de 1 avant de reprendre le test d'arrêt.