TD02 - CONSTRUCTION D'UNE GRAMMAIRE DE L

1. Objectif

L'objet de ce TD est l'écriture d'une grammaire hors-contexte non ambigüe du langage L. Cette grammaire est d'une importance capitale pour la suite du projet. En effet c'est à partir de celle-ci que l'on écrira l'analyseur syntaxique du compilateur.

2. Spécification du langage L

On prendra pour modèles les langages comme C, dont on pourra remplacer les mots-clés par d'autres plus expressifs ou plus sympathiques, avec les contraintes suivantes :

Types simples. Le langage L connaît un seul type simple, le type `entier`.

Types dérivés. L possède un unique type dérivé, le type « tableau d'entiers à un indice ». Une variable de type tableau est nécessairement globale. Une variable de ce type est déclarée à l'aide de crochets après le nom de la variable. La taille d'un tableau doit être un nombre entier positif.

Déclarations de variables. Une variable doit être déclarée avant de pouvoir être utilisée. Une déclaration de variable est constituée d'un nom de type (`entier`) suivi d'un nom de variable suivi, optionnellement, d'une taille constante de tableau entre crochets. On rappelle que tout nom de variable doit commencer par un symbole dollar ($). Dans la grammaire, on considère que cette question est déjà traitée au niveau de l'analyseur lexical et on peut représenter un identificateur de variable par son code `ID_VAR`. On peut déclarer plusieurs variables en séparant les déclarations les unes des autres par une virgule.

Chaînes de caractères. On ne s'attardera pas sur les chaînes de caractères, ni constantes ni variables. Il n'est pas possible de manipuler des chaînes de caractères en L.

Opérateurs. On s'en tiendra aux opérateurs arithmétiques (au moins : `+`, `-`, `*` et `/`) de comparaison (au moins : `<` et `=`) et logiques (au moins : `&` (et), `|` (ou) et `!` - (non)) usuels.

Instructions. Le langage L connaît les instructions suivantes :

- Affectation `$a = $b + 1;` (contrairement à C, une affectation n'est pas une expression, elle ne correspond pas à une valeur).

- Instruction `si` *expression* `alors { ... }` et `si` *expression* `alors { ... } sinon { ... }`
- Instruction `tantque` *expression* `faire { ... }`
- Instruction `retour` *expression* `;`
- Instruction d'appel de fonction simple `fonction(` *liste d'expressions* `);` et à la fonction prédéfinie `ecrire(` *expression* `);`
- Bloc d'instructions, délimité par des accolades `{ ... }`
- Instruction vide `;`

Les instructions de type vide, affectation, retour et appel de fonction sont obligatoirement suivies d'un point-virgule. Les accolades autour des blocs d'instruction du `si` et du `tantque` sont obligatoires.

Fonctions. Le langage L permet de définir des fonctions à résultat entier. Leurs arguments sont simples (i.e. des entiers) et le passage se fait par valeur. Les fonctions peuvent des variables locales.

Les parties d'une fonction doivent être déclarées dans l'ordre suivant : en-tête, déclaration des variables locales, bloc d'instructions du corps de la fonction. Remarquez que, à la différence du C, les variables locales sont déclarées entre l'en-tête et le bloc d'instructions. L'en-tête d'une fonction est un identificateur pour nommer la fonction, suivi d'une liste de déclarations d'arguments entre parenthèses.

Une fonction ne peut pas être déclarée à l'intérieur d'une autre : les fonctions sont toutes globales. Un programme en L est une suite de déclarations de fonctions. On ne peut pas déclarer deux fonctions avec le même nom (pas de surcharge) et on doit déclarer toutes les fonctions appelées (sauf les fonctions spéciales `lire` et `ecrire`). La fonction appelée en premier dans un programme en L est celle qui s'appelle `main`.

L'appel d'une fonction comme si c'était une procédure – c'est-à-dire en ignorant le résultat rendu – est permis, comme en C.

Procédures prédéfinies. Les entrées-sorties de base se présentent comme des appels à des fonctions prédéfinies `lire` et `ecrire`. Par exemple :

- lecture : *$variable* `= lire();`
- écriture : `ecrire(` *expression* `);`

À part le fait qu'elles sont prédéfinies, ces deux fonctions sont traitées comme les fonctions ordinaires, écrites par les programmeurs L. En

particulier, le passage de l'argument de la fonction `ecrire` et le retour du résultat de la fonction `lire` se fait comme pour les fonctions ordinaires.

3. Exemple

Voici un exemple d'un programme en L.

```
entier $t[10], entier $a; # déclaration de variables globales

f(entier $a, entier $b)   # déclaration d'une fonction à deux args
entier $c, entier $k;     # déclaration de deux variables locales
{                         # début d'un bloc d'instruction
  $k = $a + $b;           # affectation et expression arithmétique
  retour $k;              # valeur de retour de la fonction
}                         # fin du bloc d'instruction

main()                    # point d'entrée dans le programme
entier $d;
{
  $a = lire();            # appel à la fonction lire
  tantque 0 < $a faire {  # instruction tantque avec expression
    $d = f($d, 2 * $a);   # affectation et appel de fonction
    ecrire($d + 1);       # appel de la fonction prédéfinie ecrire
    $a = $a - 1;
  }
}
```

4. Grammaire des expressions arithmétiques

Les expressions arithmétiques en L sont composées de variables simples, d'accès à une case d'un tableau, de nombres entiers, d'appels de fonctions (dont la fonction spéciale `lire`), d'opérateurs arithmétiques et logiques et de parenthèses. Il n'y a pas de distinction entre les expressions arithmétiques et logiques : comme en C, les opérateurs logiques renvoient zéro pour représenter FAUX, et un autre entier différent de zéro pour représenter VRAI. Dans un même niveau de précédence, tous les opérateurs sont associatifs à gauche (c'est-à-dire, $5 - 3 + 2$ équivaut à $(5 - 3) + 2$). Voici dans l'ordre décroissant la priorité des opérateurs :

1	(*expression*)	
2	! (non)	
3	*	/
4	+	-
5	=	<
6	& (et)	
7	| (ou)	

Écrire la partie de la grammaire du langage L qui permet de générer/reconnaître les expressions. Cette grammaire doit être non ambigüe et respecter la priorité des opérateurs décrite ci-dessus. Pour écrire une grammaire non ambiguë, vous pouvez partir d'une grammaire ambiguë et rajouter des non terminaux pour chaque niveau de prédédence.
N'oubliez pas de représenter les appels de fonction, qui peuvent prendre en paramètre une liste d'expressions séparées par des virgules. Ces expressions sont évaluées dans l'ordre de gauche à droite.

5. Grammaire des instructions

Écrire la partie de la grammaire du langage L qui permet de générer des instructions de différents types. N'oubliez pas les règles pour représenter une suite d'instructions dans un bloc, et rappelez vous que seuls certains types d'instructions sont suivis d'un point-virgule.

6. Grammaire des déclarations de variables

Écrire la partie de la grammaire du langage L qui permet de générer une séquence de déclarations de variables.

7. Grammaire des définitions de fonctions

Écrire la partie de la grammaire du langage L qui permet de générer une séquence de définitions de fonctions.

8. Grammaire complète

Assembler toutes le sous-grammaires précédentes pour créer la grammaire du langage L.