****

**LOG3210 – Éléments de langages et compilateurs**

**Hiver 2021**

**Examen Intra**

**Groupe 01**

**1947497 – Yuhan Li**

**Soumis à : Madame Guerrier**

**15 mars 2021**

**Grammaire**

D’abord, pour l’importation j’ai créé une fonction (ImportStmt) que j’ai appelé une seule fois, soit au début de la fonction Program puisqu’on effectue les importations uniquement au début du code. ImportStmt() comprend deux jetons (début et fin d’importation) et une fonction Identifier() qui est une des fonctions de base du langage utilisée pour les expressions quelconques (ex. : noms de fonction, de variable, etc).

Outre Identifier(), ma grammaire comporte 4 autres expressions qui sont à la base de la syntaxe. Ces fonctions sont dédiées aux String, Bool, Real et Int. Chacune de ces fonctions possèdent un jeton que je sauvegarde au niveau du nœud. Pour le String, j’ai ajouté le jeton <STR> qui suit un format inspiré par ce lien : <https://stackoverflow.com/questions/46990699/define-token-to-match-any-string>. Cela permet de reconnaître n’importe quelle chaîne de string, vide ou pas. J’ai regroupé ces fonctions dans PrimitiveExpr() pour simplifier l’appel à ces valeurs de base. De plus, sachant qu’on peut assigner des valeurs directement par l’appel de fonction, j’ai également ajouté CallFunction() dans les options de PrimitiveExpr().

CallFunction() est aussi appelé dans Stmt() puisqu’un appel de fonction peut être une ligne de code à lui seul. En effet, j’ai assumé que chaque fonction appelée dans Stmt() était des lignes ou des blocs d’expressions qui ont une exécution à part entière. Donc, en plus de CallFunction(), j’ai inclus les fonctions liées au cœur, à la déclaration de variable et de fonction, ainsi qu’aux If/While.

Pour la déclaration, j’ai divisé ça en deux fonctions (*DeclareStmt* et *AssignStmt*), simplement pour avoir une meilleure visualisation de ce qui se produit. Pour DeclareStmt(), j’ai ajouté un jeton (“compi”) pour l’annonce d’une déclaration. J’ai également ajouté d’autres terminaux au niveau du jeton TYPE afin de représenter chaque type nécessaire à EstheRust. Pour AssignStmt(), j’ai implémenté Expr() qui est appelée à la droite du terminal “=”. Cette fonction représente des valeurs concrètes qui peuvent provenir d’une opération logique ou arithmétique. C’est pourquoi, Expr() est implémentée de manière récursive afin d’offrir une possibilité d’accès à chaque type d’opération et aux valeurs de base de PrimitiveExpr().

Concernant les diverses opérations, je les ai regroupés en 4 fonctions, où la récursivité suit un ordre d’importance inspiré du tp1 et tp2. La moins prioritaire est CompExpr() (comparaison : and/or ou </>/==/etc), donc elle est appelée directement à partir de Expr(). Pour établir un comportement récursif, on retrouve, de part et d’autre de l’opérateur de comparaison, un appel à l’opération plus prioritaire ainsi qu’à CompExpr(). Ce parsage hiérarchique se poursuit donc jusqu’à l’opération la plus importante, soit les parenthèses (*ParenExpr*). C’est dans ParenExpr() aussi qu’on accède les valeurs de base (*PrimitiveExpr*).

Pour la déclaration de fonction, j’ai implémenté deux fonctions pour mieux visualiser les évènements. La première fonction concerne la signature complète d’une fonction, qui débute en appelant la deuxième fonction. Cette seconde fonction concerne la signature de la première ligne des déclarations. J’ai donc ajouté deux jetons, soit pour l’annonce d’une déclaration et pour la flèche pointant vers le type de retour.

Pour le If et le While, j’ai fait une fonction pour chacun. Chacune des fonctions prend en compte le if/while avec leur condition, ainsi que leur « body ». La fonction IfStmt() contient un crochet en plus, car ce dernier contient le « Else » qui apparaît seule une fois lorsque nécessaire.

Enfin, pour l’opérateur Cœur, ma syntaxe consiste uniquement d’une fonction comprenant le jeton <HEART> que j’ai ajouté au lexique.

**Visiteur**

D’abord, pour le visiteur de Program, Block et Stmt, le principe est le même; c’est-à-dire