# Rapport : Projet Lexer-Parser

Gaspard REGHEM / Thomas VARIN

May 15, 2022

## 1 Niveau 1

Dans un premier temps, nous avons envisagé de ne faire que le niveau 1 donc il existe un *pretty print* du code en plus du niveau 2.

### 1.1 Lexer

Le lexer se trouve dans le fichier *lexer.l* et il est largement inspiré du lexer du TP sur bison. Le lexer reconnait toutes les structures de code possibles d'après le sujet (do/od, if/fi, séquence), la déclaration de processus et de variables, les 4 opérations arithmétiques usuelles et les 5 comparaisons. Les commentaires sont ignorés par le lexer et il distingue deux cas : les commentaires commençant par `//` finissant en fin de ligne et ceux commençant par `/*` finissant par `*/`. Les variables doivent posséder un nom commençant par une minuscule ou un tiret du bas pour être reconnu. On a ajouté l'option `yylineno` pour obtenir la localisation des erreurs lors des échecs.

### 1.2 Parser

Le parser se situe dans le fichier *parser.y* et il est lui-aussi inspiré du parser du TP sur bison. On commence par déclarer toutes les structures qui sont principalement des arbres ou des listes simplement chainées. Ensuite, on peut trouver toutes les fonctions permettant de créer ces structures, en plus, de certaines fonctions utiles. Puis, il y a la grammaire qui permet de construire l'AST. Cet AST est ensuite lu pour effectuer le *pretty print* qui est défini juste en dessous par un ensemble de fonctions qui s'appellent entre elles.

## 2 Niveau 2

Pour ce qui est de la partie 2, nous avons commencé par essayer d'exécuter le code fourni (ou en tout cas de l'interpréter dans notre programme). Tout ceci se situe dans le fichier du lexer (*lexer.l*). Pour chaque commande, nous avons alors rajouté si elle était la prochaine à être exécutée, avec l'attribut `toDoNext`. Alors, la fonction `execute` trouve cette commande et l'exécute, avec une action différente en fonction de cette dernière. On détermine ensuite la prochaine commande à exécuter et on termine la fonction. Il nous suffit de faire suffisamment d'exécutions afin de trouver quelles spécifications sont atteignables.

### 2.1 Détails techniques

Quelques choix peuvent être intéressants à mentionner :

1. Pour sortir d'une boucle avec l'instruction `break`, il nous suffit d'enlever le `toDoNext` du `break` et de le mettre à la boucle directement. Notre fonction qui détermine la prochaine instruction verra alors une boucle terminée et passera à la commande d'après.

2. Afin de faciliter la lecture du code, nous n'avons pas mis beaucoup de commentaires mais des valeurs de fonctions retournés avec des constantes de pré-processeur. Cela nous permet de donner des mots à nos constantes, et d'y voir plus clair.