Equipe GL 33

Documentation sur la validation

Tests unitaires :

Lorsque nous avions remarqué une irrégularité au niveau du code, ou lorsque un professeur nous faisait remonter une erreur remarquée (notamment lors du rendu intermédiaire), nous avons mis en place des tests unitaires, destinés à vérifier qu'un point spécifique d'une fonctionnalité était correct. On peut citer en exemple le fichier deca qui avait pour unique instruction de faire une division par 0, afin de vérifier que l'erreur remontait correctement à l'exécution. Ces tests ont ensuite été ajoutés dans la base de tests de non-régression une fois que les fonctionnalités étaient implantées correctement.

Tests automatisés :

Ce projet contient des scripts qui permettent de faire des tests de non-régression (appliqué à la vérification syntaxique et contextuelle du compilateur). Il va exécuter l'un des scripts fournis (test_lex, test_synt, ou test_context, selon les cas), vérifier que l'on obtient une erreur sur les fichiers deca incorrects, et que l'on en obtient pas pour les fichiers deca corrects. Il va ensuite comparer la sortie de ces tests avec une sortie enregistrée au préalable, qui permet de vérifier que les erreurs affichées sont bien correctes, et que les vérifications syntaxiques / contextuelles donnent des fichiers corrects en entrée de l'étape suivante.

- Syntaxe

Ici, nous avons séparé les tests en deux parties : les tests valides et les tests invalides. Les tests invalides sont là vérifier soit le bon fonctionnement du lexer, soit du parser, mais jamais les deux. Ainsi, le script compare avec l'attendu soit avec le lexer soit avec le parser, et fait sa décision en fonction du nom d'extension du fichier "expected" lié au fichier deca testé (.lis pour le parser, .lxr pour le lexer). Comme les fichiers valides le sont pour le lexer et le parser, le script compare avec l'attendu pour le lexer et le parser.

- Contexte

Comme avant, nous avons séparé les tests en deux parties, entre les valides et les invalides. Ici, pas besoin de faire plusieurs vérifications, le script ne fait donc qu'une passe sur chaque fichier deca, qu'il soit correct ou non

- Génération de code

Malheureusement, le temps nous a manqué pour automatiser cette partie, mais nous voulions mettre en place la même mécanique que pour l'étape de vérification syntaxique, à savoir deux catégories valides et invalides, et de ne vérifier qu'un aspect par fichier invalide et tous les aspects pour les fichiers valides (compilation, décompilation, limitation à un nombre aléatoire de registre au lieu des 16 usuels, no-check…).