Documentation de validation

G27

Descriptif des tests
Pour l'étape A, que des tests systèmes (.deca).
Pour les étapes B et C, tests unitaires (JUnit) en plus des tests systèmes.

Les tests sont organisés de la même manière qu'initialement : trois répertoires "syntax", "context" et "codegen", puis ces répertoires répartis en "valid" et "invalid". Les fichiers résultats (".res") sont stockés dans les répertoires "resultats", à côté de "provided". Pour l'extension, un répertoire "gbacodegen" a été ajouté, contenant des codes valides.

Au début, les tests étaient numérotés pour rendre compte de l'évolution (le test 25 est à faire après le test 24), mais des tests ont été ajoutés (car oubliés) *a posteriori* donc l'ordre des notations n'a plus trop d'importance.
Les tests avaient pour leur totalité pour objectif d'être des étapes (checkpoints) afin de "guider" la programmation du compilateur : passer le test 10, une fois réussi passer le test 11, puis le 12, etc. Cet objectif n'a pas été constamment atteint ; souvent les tests arrivaient après le développement du compilateur, ils servaient donc de vérification du bon fonctionnement du compilateur, et il fallait régler les bugs qui arrivaient. Tous les tests écrits fonctionnent comme attendu (il manque donc sûrement des tests).

Les scripts de tests
Pour le développement du compilateur, les tests sont passés un à un (avec test_lex, test_synt, test_cont, decac et ima) au fur et à mesure. Une fois un test deca fonctionnel (le programme fonctionne comme attendu), on crée un .res (avec les scripts "creer-***res.sh") qui servira lors des étapes de validation : lorsque du code a été *push*, il suffit de lancer les oracles pour lancer l'execution des tests et vérifier leur sortie avec celle du .res.
Afin de faire passer tous les tests, il suffit de lancer oracle.sh, qui lancera lui-même les scripts oracle-lex.sh, oracle-synt.sh, oracle-cont.sh et oracle-ima.sh. Chacun de ces scripts lance les tests respectifs (test_lex pour oracle-lex.sh,..., deca+ima pour oracle-ima.sh), comparant les sorties du programme *prog* avec *prog.res* (on sait que les .res contiennent le resultat attendu).
Si le test passe, un "OK" vert est affiché, sinon un "NOT OK".

Gestion des risques et gestion des rendus
La gestion des risques et des rendus s'est faite assez naturellement : lors des réunions on évoquait les dates des rendus suivants ainsi que les problèmes auxquels chacun est confronté. Ainsi chaque fois qu'un risque était identifié, des

mesures étaient mises en place, et parfois retirées ultérieurement (le risque pouvant être temporaire). Le document rendu sur la gestion des risques et des rendus ne nous a pas été très utile : lorsqu'un risque apparaissait tout le monde était prévenu, et le document sur lequel figurait le risque et sa solution n'était pas consulté, idem pour la gestion des rendus. Tout le monde avait les informations en tête, car nous n'avions pas l'impression d'être confrontés à beaucoup de risques ou de rendus à la fois.

Autre Nous n'avons pas utilisé JaCoCo ni Mockito car notre gestion de tests semblait fonctionner sans. Cependant leur utilisation est une possibilité en guise d'amélioration (une fois le projet et sa gestion maîtrisés, on peut améliorer certains points de l'étape de validation).