Document de Validation

Groupe 41

Corentin HEUZÉ

Dorian VERNEY

Adrien AUBERT

Yuxuan LU

Yung Pheng THOR

Table de matière

1. Introduction

Une composante cruciale du projet consiste à garantir que le code atteint ses objectifs. Pour cela, diverses techniques sont mises en œuvre.

Dans notre projet, la validation du compilateur repose sur l'utilisation de tests. C'est pourquoi nous avons suivi une architecture de tests particulière, car cela a permis d'évaluer la qualité du système et de s'assurer qu'il répond aux attentes fixées.

2. Description de tests

Des tests ont été élaborés pour diverses phases du projet. En ce qui concerne la première phase (partie A), les tests se concentrent sur la construction de l'arbre. Dans la partie B, les tests couvrent le parcours de l'arbre abstrait, la gestion des environnements, les enrichissements de l'arbre et la décoration. Quant à la phase C, les tests portent sur les différentes constructions de code .Deca, en mettant particulièrement l'accent sur les séquences d'instructions assembleur.

L'objectif des tests était d'évaluer le bon fonctionnement de notre compilateur. Ainsi, la stratégie adoptée consistait à classer les tests en fonction des phases du projet. C'est pourquoi l'on retrouve plusieurs répertoires de tests, chacun contenant deux autres répertoires nommés respectivement "valide" et "invalide". L'objectif est non seulement de vérifier la bonne exécution du programme, mais également de s'assurer que les messages d'erreurs, voire même l'identification de ces dernières, correspondent à ce que nous souhaitons afficher.

Nous avons cherché à évaluer les aspects lexicographiques, syntaxiques, contextuels et la partie de génération de code.

2.1. Tests Lexicographiques

Les tests lexicaux sont situés dans les répertoires *./src/test/deca/lexer/valid/* et *./src/test/deca/lexer/invalid/*. Ces tests peuvent être exécutés manuellement à l'aide de la commande *./test_lex <nom du fichier .deca>*, ou bien automatiquement à l'aide du script *test-lexer.sh*.

L'ensemble des tests s'afficheront dans la console avec un récapitulatif à la fin, permettant de comptabiliser les réussites et les échecs. Chaque ligne correspondant à un test sera également colorée pour indiquer son succès ou son échec. Les lignes en vert représentent un succès, tandis que les lignes en rouge signalent un échec.

2.2. Tests Syntaxiques

Les tests syntaxiques se trouvent dans les répertoires ./src/test/deca/syntax/valid/ et ./src/test/deca/syntax/invalid/. Vous pouvez les exécuter manuellement en utilisant la commande *./test_synt <nom du fichier .deca>*, ou de manière automatisée à l'aide du script test-syntax.sh.

L'intégralité des tests s'affiche dans la console accompagnée d'un récapitulatif à la fin, permettant de répertorier les succès et les échecs. Chaque ligne correspondant à un test sera colorée pour indiquer son résultat, le vert symbolisant un succès et le rouge signalant un échec.

2.3. Tests contextuels

Les tests contextuels sont localisés dans les répertoires *ProjetGL/src/test/deca/context/valid* et *ProjetGL/src/test/deca/syntax/context/invalid*. Ces tests peuvent être exécutés manuellement en utilisant la commande *test_context <nom du fichier test.deca>*, ou bien automatiquement à l'aide d'un script automatisé appelé *test-context.sh*.

Tous les tests seront affichés dans la console avec un bilan à la fin pour comptabiliser les succès et les échecs. Chaque ligne correspondant aux tests aura également une couleur pour indiquer son succès ou son échec. Les lignes en vert indiquent un succès, tandis que les lignes en rouge signalent un échec.

2.4. Tests Codegen

Les tests de génération de code sont localisés dans les répertoires *./ProjetGL/src/test/deca/codegen/valid*, *./ProjetGL/src/test/deca/codegen/perf/provided*, *./ProjetGL/src/test/deca/codegen/provided*. Ces tests peuvent être exécutés manuellement en utilisant la commande *decac <nom du fichier.deca>*, suivie de la commande *ima <nom du fichier.ass>*, ou automatiquement à l'aide des scripts *auto-codegen.sh* et *decompile.sh*. Le script *auto-codegen.sh* lance tous les tests dans les trois répertoires et compare avec les résultats écrits dans le fichier lui-même.

3. Les scripts de tests

Pour automatiser les tests, deux scripts ont été créés dans le répertoire *./src/test/script*. Tout d'abord, il y a *all.sh* pour exécuter tous les tests de la Partie A et de la Partie B. Ensuite, il y a *auto-codegen.sh* pour lancer tous les tests de la Partie C. On peut également exécuter la commande *mvn verify* pour lancer tous les tests.

4. Gestion des risques

Au sein d'une équipe, il est crucial d'anticiper les risques potentiels afin de prévenir d'éventuels problèmes susceptibles de perturber le bon fonctionnement du groupe. L'objectif est de trouver des solutions.

On peut distinguer deux catégories de risques :

- Risques liés à l'organisation
- Risques techniques

Risque	Solution	Importance (_/3)
Oubli de la date d'un suivi	Mise en place d'un planning, avec des suivis intermédiaires permettant de garantir l'achèvement des tâches nécessaires pour assurer un suivi optimal	3
Oubli d'un document dans un rendu de suivi	Check-list des éléments à rendre pour chaque suivi	3
Branching très lourd et des problèmes de branches et version divergents	Produire des commits clairs et précis pour savoir se repérer plus facilement en cas de problème. Toujours être à deux pour minimiser les erreurs	3
Mésentente entre des membres du groupe	Organisation d'une petite réunion dans le but de trouver la source du problème et de le résoudre rapidement afin de préserver le bon déroulement du projet.	3
Erreur de compilation	Validation de la compilation réussie du code avant tout push sur Git et évitement d'effectuer des changements significatifs du code juste avant les dates limites	3
Avoir une couverture de tests trop faible	Utilisation de l'outil Jacoco pour vérifier les parties non couvertes par les tests	3
Problèmes de régression	Lancement de l'ensemble de nos tests après l'ajout de nouvelles fonctionnalités dans le but d'éviter d'éventuels problèmes de régression	3

5. Gestion des rendus

Suite au rendu du code source du projet le lundi 22 janvier 2024, tous les membres de l'équipe se sont réunis pour répartir les tâches de rédaction. Chaque membre a pris en charge une partie spécifique du compilateur, s'engageant à la compléter avant les dates limites de remise. Yung et Yuxuan sont responsables du document de validation, du bilan du projet et du manuel utilisateur. Dorian et Corentin rédigent la documentation de conception, l'impact énergétique et les slides pour la soutenance. Adrien poursuit le travail sur la partie extension et prend en charge la documentation relative aux extensions.

6. Résultats de Jacoco

Deca Compiler

Deca Compiler

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
fr.ensimag.deca.syntax		74%		56%	481	686	506	2,091	246	368	2	48
fr.ensimag.deca.tree		85%		80%	182	783	336	2,120	110	542	3	86
fr.ensimag.deca		69%		67%	25	89	79	281	7	53	0	5
fr.ensimag.arm.pseudocode		0%		0%	42	42	94	94	33	33	6	6
fr.ensimag.deca.context		79%		66%	44	166	48	270	25	130	0	21
fr.ensimag.ima.pseudocode		80%		83%	25	89	37	191	21	77	2	27
fr.ensimag.ima.pseudocode.instructions		79%		n/a	14	62	24	111	14	62	10	54
fr.ensimag.deca.codegen		92%		75%	10	37	13	95	6	29	2	7
fr.ensimag.deca.tools		98%		87%	1	19	1	43	0	15	0	3
Total	5,151 of 24,203	78%	417 of 1,262	66%	824	1,973	1,138	5,296	462	1,309	25	257

Une couverture de code de 78% est considérée comme assez bonne. Parfois, atteindre une couverture de 100% n'est pas nécessaire ou réaliste, en particulier si certaines parties du code sont difficiles à tester ou ne présentent pas de risques significatifs.

Il est recommandé de considérer également la qualité des tests, en s'assurant qu'ils couvrent différents scénarios et situations, et qu'ils sont suffisamment robustes pour détecter les éventuelles erreurs. Dans l'ensemble, une couverture de code de 75% est un indicateur positif de la maturité des tests, mais il est toujours bon de continuer à améliorer la couverture et la qualité des tests au fur et à mesure que le développement progresse.

7. Méthodes de validation utilisées autres que le test

Pour valider notre code, chaque membre de l'équipe, lors de chaque pull, a pris en charge l'intégration des nouvelles parties du code écrites. La hiérarchie du projet nous a également permis de déboguer plusieurs problèmes ; en effet, la partie C permet de corriger diverses vulnérabilités dans la partie B, nécessitant ainsi un fonctionnement correct du code en amont.