Bilan individuel de compétences

Louis Jézéquel-Royer

Janvier 2021

1 Introduction

Lors de ce projet GL, réalisé dans l'équipe 10, j'ai eu l'occasion de développer de nombreuses compétences, que je considère nécessaire pour mon évolution aussi bien personnelle que professionnelle. Ce document a pour but de mettre en avant deux des compétences proposées, qui permettent d'illustrer les enseignements tirés de ce projet.

2 Comprendre finement les implications des calculs faits par une machine

Étant un des deux membres s'occupant de la partie C, c'est-à-dire la partie permettant de générer le code assembleur à partir d'un arbre, ce projet m'a bien entendu permis de comprendre comment un langage de haut niveau (Deca, langage orienté objet qui ressemble à Java) est traduit en un langage de plus bas niveau (assembleur). L'apport principal pour moi est la gestion de la mémoire, notamment dans la pile et le tas.
Après avoir été quelques fois victimes de la fameuse erreur de débordement de pile *"Stack Overflow"* et avoir effectué un bon nombre de recherches sur stack-overflow (y compris durant ce projet), j'étais curieux de découvrir comment fonctionnait la pile. Ainsi, la pile est la zone mémoire dans laquelle se situe toutes les variables d'environnement (les variables globales du main, les variables temporaires se situant dans les méthodes, les méthodes en elle même). La pile contient également toutes les variables temporaires. Par exemple, lors de l'utilisation des registres, il arrive que plus aucun registre ne soit disponible et que nous devons conservé les informations présentes à l'intérieur. Nous allons alors *PUSH* un registre sur le haut de la pile pour le rendre disponible, c'est à dire "poser" le contenu du registre en question afin de pouvoir le retrouver ultérieurement grâce à un *POP*.
Après avoir compris comment fonctionnait la pile pour le sans objet, est venu s'ajouter le fonctionnement du tas. Le tas représente une zone mémoire permettant de stocker les différents objets créés ainsi que leurs attributs.
Couplé avec l'utilisation conséquente du langage assembleur, j'ai pu appréhender

d'une meilleure manière la traduction d'un langage de haut niveau à un langage de bas niveau, et comprendre comment les différentes opérations et instructions fonctionnaient d'un point de vue de la machine.

Par ailleurs, en plus d'utiliser le langage de programmation Java, j'ai pu comprendre plus en profondeur les notions liées à un langage orienté objet, notamment l'héritage d'un point de vue mémoire. Les classes filles sont définies dans un premier temps comme la classe dont elles héritent, et ensuite seulement on vient rajouter et surcharger des attributs et/ou des méthodes.

3 Mettre en œuvre un processus de validation

Durant tout ce projet, avoir une base de test solide permettant de déceler nos erreurs nous a été primordial. Durant l'ensemble du projet, notre vision et notre manière de tester n'a cessé d'évoluer.
Dans un premier temps, nous avons cherché à réaliser le plus de tests unitaires possibles sur les différentes fonctions que nous implantions. Cependant, le nombre de fichiers tests ne cessant d'augmenter, il est vite devenu pénible de procéder aux tests l'un après l'autre. Ainsi, un de nos camarades à décider de créer un fichier bash permettant de tester tous les fichiers dans le dossier de test de codegen, comparant la sortie de l'exécution du fichier assembleur généré avec un fichier réponse associé au fichier *.deca*.
Après cela, il est devenu simpe pour les autres membres du groupe de facilement tester un grand nombre de fichiers *.deca* pour facilement déceler les erreurs de notre compilateur. Après avoir vu à quel point un fichier bash pouvait simplifier la vie, j'ai moi même décidé de m'y pencher à plusieurs reprises afin de simplifier les exécutions de notre programme et nos tests.
Lors du dernier weekend avant le rendu final, nous avons réussi à correctement faire tourner Jacoco, nous montrant ainsi que nous avions effectué une couverture de test de 74%. Le reste du weekend a été consacré à améliorer ce score, ce qui nous a permis de rajouter des tests décelant des erreurs que l'on ne voyait pas précédemment.
Jacoco a été d'une grande utilité et je regrette de ne pas m'y être penché plus tôt dans le projet, cela aurait permis de repérer facilement des erreurs en pointant les endroit non testés dans le code.
L'implémentation de l'extension TRIGO en elle même a été un bon test pour nous et nous a permis de relever d'autres erreurs.
Suite à tout cela, nous avons essayé de mettre en place un système de liste chainée, système simple mais qui nous semblait primordial pour un langage orienté objet. Ce test nous a permis de mettre en avant les limites de *ima* (notamment la taille du tas) plus que les erreurs de notre part.
Notre équipe a particulièrement bien géré son temps, et à part quelques frayeurs, nous n'avons pas apporté de changement majeur proche des différentes deadline.

Ce projet m'a permis de comprendre l'importance d'avoir une base de tests solide, et que malgré l'impression de tester la presque intégralité du programme, des erreurs peuvent toujours survenir.

4 Conclusion

J'attendais principalement de ce projet une meilleure compréhension des calculs effectués par la machine, et je n'ai pas été déçu. De plus, j'ai compris l'importance d'avoir une grande multitude de tests et d'être au courant des potentiels points faibles du code que l'on écrit pour pouvoir corriger des erreurs plus vite. Par ailleurs, ce projet m'a mis sur la voie des scripts bash et m'a donné envie de m'y pencher plus sérieusement.