Bilan de gestion d’équipe et de projet

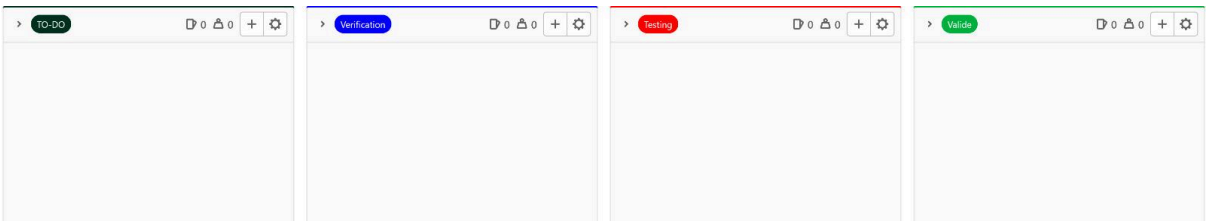
BOUTALEB Youssef
BOUHAR Mounsef
MALYAH Lina
EL KAZDAM Zakaria
EL ASLI Adnan

I. Organisation de l'équipe

L'équipe a adopté la méthodologie Scrum, une approche agile axée sur la flexibilité et l'adaptabilité face aux changements. Le projet a été segmenté en sprints, des cycles de développement courts caractérisés par des objectifs clairs et une durée généralement fixe, par exemple, une semaine.

GitLab a été sélectionné comme plateforme de gestion de code source et de collaboration, permettant ainsi une traçabilité complète des modifications apportées au code et une gestion efficace des branches de développement.

Pour structurer le travail dans chaque sprint, des listes d'étapes ont été établies (TODO, vérification, test, validation), en plus de la création d'issues sur GitLab. À chaque itération, un membre de l'équipe s'assigne une issue en la plaçant dans la liste TODO. Une autre personne est désignée pour effectuer la vérification, et une troisième personne pour effectuer les tests. Enfin, la validation est réalisée par l'ensemble du groupe, offrant une perspective critique sur chaque spécification du compilateur.



Au démarrage du projet, nous avons délibérément confronté l'ampleur du sujet à cinq personnes pour obtenir une compréhension initiale. Nous avons commencé par la réalisation de la partie "Hello World", puis à la fin de cette étape, la répartition naturelle et individuelle des rôles s'est mise en place. Cette approche a permis d'instaurer une organisation efficace dès le début du projet, avec des points quotidiens pour suivre les réalisations et les prochaines étapes. Chacun avait une responsabilité bien définie.

Cependant, au fil des vacances, nous avons progressivement perdu certaines bonnes pratiques. Les réunions ont diminué, entraînant une communication moindre entre les membres de l'équipe. À la fin des vacances et au début des trois dernières semaines du projet, l'équipe a retrouvé sa motivation. Les rôles se sont rétablis, les

tâches sont devenues plus claires, et une répartition du travail plus équilibrée s'est opérée. Des binômes se sont formés pour aborder les parties A, B et C, tandis qu'une personne se concentrait sur l'extension. En cas de blocage sur une partie, l'équipe s'entraidait pour résoudre rapidement le problème. La cohésion de l'équipe et la volonté d'apprendre chaque jour ont été des valeurs clés qui ont contribué à la réalisation d'un produit satisfaisant à la fin du projet.

II. Historique du projet

Pour mettre en œuvre ce compilateur, nous avons procédé à une analyse approfondie de l'énoncé, ce qui nous a permis de subdiviser le langage Deca en sous-langages. Nous avons ensuite élaboré un compilateur capable de traiter chacun de ces sous-langages. Ainsi, l'idée de suivre une approche graduelle pour le développement de ce compilateur s'est révélée judicieuse. Nous avons amorcé le processus par le plus petit sous-langage : Hello-World.

Avant d'entamer la première étape, nous avons établi comme objectif que notre compilateur devait, à la fin, être capable de générer des codes d'assembleur permettant d'imprimer des chaînes par l'intermédiaire de l'IMA.

La deuxième étape s'est concentrée sur le traitement des programmes contenant un seul bloc Main, capable d'exécuter des opérations générales telles que les opérations arithmétiques, les opérations booléennes et les structures de contrôle. À ce stade, le compilateur devait également être capable de détecter des erreurs contextuelles ainsi que des erreurs d'exécution.

La troisième étape s'est penchée sur le langage orienté objet. Notre objectif était d'assurer le bon fonctionnement du compilateur en l' adaptant aux concepts de la programmation orientée objet, tels que l'héritage.

2.1 Etape B

L'étape B s'est révélée plus complexe que la précédente, nécessitant plusieurs jours de discussions pour élaborer une approche efficace de notre analyseur contextuel. Pour mieux comprendre cette étape, nous avons d'abord dû assimiler la conception de l'analyseur lexical et syntaxique, ce qui n'était pas évident en tant que première expérience avec le générateur d'analyseur Antlr.

Après avoir correctement implémenté les parties nécessaires pour le parser et le lexer dans les fichiers DecaParser et DecaLexer respectivement, et une fois que nous avons bien saisi les attentes pour la partie B, à savoir la vérification contextuelle pour

les différents nœuds de l'arbre, nous avons mis en place la grammaire attribuée de la syntaxe contextuelle du langage Deca.

Cette vérification contextuelle s'effectue en deux temps. Tout d'abord, elle consiste à lever un message d'erreur si l'entrée ne respecte pas la grammaire attribuée établie au préalable. À cette fin, nous avons introduit une nouvelle classe, EnvironmentExp, structurée de manière similaire à la classe EnvironmentType, établissant une hiérarchie des environnements avec des getters et setters appropriés. Ce choix facilite la manipulation ultérieure de nos EnvironmentExp.

Ces deux environnements sont enrichis au fur et à mesure des trois passes de la vérification contextuelle. Notre méthode de progression a impliqué la compréhension de l'écriture de la grammaire avec ses conditions et ses affectations. Nous avons écrit le code en suivant les règles de la grammaire attribuée, en ajoutant des mini-tests à chaque étape jusqu'à ce que toutes les règles soient pleinement implémentées.

À mesure que le processus avance, nos tests gagnent en complexité, et notre compilateur étend sa couverture.

2.2 Etape C

Cette étape présente moins de directives que les deux précédentes. Par ailleurs, l'implémentation pour le sous-langage Hello-World est déjà presque entièrement disponible, ce qui nous a permis de consacrer davantage de temps aux phases d'analyse et de conception. Durant la phase d'analyse de chaque cycle, nous nous sommes documentés de manière approfondie pour obtenir une vision globale des tâches à accomplir pour chaque partie.

Au départ, la génération d'instructions assembleur nous semblait complexe, nécessitant une révision du langage assembleur et de l'implémentation du sous-langage Hello-World pour comprendre son fonctionnement. La phase de conception a impliqué l'implémentation de structures de données spécifiques afin d'adapter la gestion de la mémoire à chaque partie.

III. Répartition temporelle des différentes phases d'activité

Le temps consacré aux diverses activités a été réparti de manière stratégique tout au long du processus de développement, couvrant les étapes clés du projet, à savoir l'analyse, la conception, le codage, la validation et la documentation.

- ***Analyse*** : Une part significative du temps a été allouée à l'analyse, où nous nous sommes documentés de manière approfondie pour comprendre les exigences de chaque partie (Hello-World, Sans-Objet, Essentiel, Complet). Cette phase a été cruciale pour établir une vision claire de nos objectifs et des spécifications à respecter.
- ***Conception*** : Après une analyse approfondie, nous avons consacré du temps à la phase de conception. Cela a impliqué la création de structures de données spécifiques pour chaque partie, garantissant une gestion de la mémoire adaptée aux besoins uniques de chaque partie.
- ***Codage*** : La phase de codage a été menée en parallèle avec la conception. Nous avons suivi les règles de la grammaire attribuée de décompilation, en mettant en œuvre le code progressivement tout en ajoutant des mini-tests pour garantir la conformité avec les spécifications.
- ***Validation*** : La validation a été une étape continue, réalisée tout au long du processus de codage. Des tests ont été mis en place à chaque étape, permettant de détecter et de corriger les erreurs au fur et à mesure.
- ***Documentation*** : La documentation a été soigneusement élaborée, couvrant les étapes du processus, les choix de conception, les règles de décompilation, et les décisions relatives à l'extension. Cela assure une référence claire pour le projet, facilitant la compréhension et la maintenabilité à l'avenir.