# Bilan

Groupe 17
GUILLAUME Laure
LUU Duc Anh
EL RHATRIF Mohammed Amine
CHAKIR Sami
FARID Othmane

24 janvier 2017

## 1 Organisation de l'équipe

Le module SHEME nous a imposé de créer une charte d'équipe, ce qui nous a permis de mettre en place entre nous des règles et des rôles. La répartition des rôles fut naturelle et individuelle. Cette démarche nous a permis d'avoir un début de projet bien organisé et efficace. Nous faisions un point chaque jour sur ce qui avait été fait et ce qui devait être fait. Ainsi nous avions chacun une partie bien définie.

Cependant nous avons au fur et à mesure perdu les bonnes habitudes établies en début de projet. Nous avons moins fait de réunions, ce qui a indubitablement diminué la communication entre les membres de l'équipe. Sur la fin du projet les rôles et les tâches sont devenus moins clairs ce qui a provoqué une mauvaise répartition du travail. Il aurait fallu être plus rigoureux dans notre organisation. Nous aurions dû continuer à suivre les règles établies dans la charte et ne pas perdre de vue le planning établi au début du projet.

## 2 Historique du projet

Pour implémenter ce compilateur, on a tout d'abord analysé la consigne et on a pu diviser le langage deca en quatres sous-langages et construire un compilateur qui puisse compiler chacun de ces sous-langages . Donc c'est une bonne idée de choisir le façon de développement ce compilateur en spirale.

On commence par le plus petit sous-langage : *Hello-World*. Avant le début du premier cycle, on pose un objectif qu'à la fin, notre compilateur doit pouvoir générer des codes d'assembleur qui permet d'imprimer des chaines par l'IMA. Le deuxième cycle consiste à traiter les programmes qui ne contiennent qu'un seul Main block et qui peut réaliser les opérations générales comme les arithmétiques, les booléens et les contrôles.

Le compilateur à ce niveau doit aussi lever des erreurs contextuelles ainsi que les erreurs d'exécutions. La troisième cycle travaille sur le langage orienté objet. Notre compilateur doit fonctionner bien et s'adapter aux conceptions du POO comme l'héritage, la liaison dynamique, l'abtraction, l'encapsulation.

## 2.1 Etape B

L'étape B s'est avérée plus difficile et surtout plus ambigüe que celle d'avant. Pour remédier à cela, on a passé plusieurs jours à chercher l'information, à discuter et à débattre sur comment réaliser notre analyseur contextuel avant la proposition d'une conception utile qui nous guidera vers la solution attendue. Mais avant cela, il était nécessaire de comprendre la conception de l'anlyseur lexical et syntaxique, une chose qui n'était pas évidente, vu que c'était notre première rencontre avec le générateur d'analyseur Antlr, et il n'y avait pas un descriptif efficace pour utiliser cet outil. De ce fait, on a procédé par analogie avec ce qui était fait dans les fichiers CalcParser et CalcLexer.

Après avoir implémenté ce qui a été demandé pour le parser et le lexer dans les fichiers DecaParser et DecaLexer respectivement, et après avoir bien compris ce qui a été attendu pour la partie B, à savoir ; la vérification contextuelle pour les différents noeuds de l'arbre, on a implémenté la grammaire attribuée de la syntaxe contextuelle du langage Deca, décrite dans le cahier de charge et selon le parcours prévu pour cet arbre, et d'une façon qui garantit à la fois l'efficacité et la sûreté de la programmation.

Cette vérification se fait en deux temps :
- Lever un message d'erreur dans le cas où le flux d'entrée ne respecte pas la grammaire attribuée fixée auparavant, c'est dans ce but qu'on a implémenté le domaine des attributs de la grammaire, en commençant par l'environnement type, et en finissant par l'environnement expression. Pour ce faire on a créé une nouvelle classe Environment_type, une classe qui ressemble à la conception de la classe Envornment_exp. Dans cette classe on enregistre les symboles et leurs type_Definition correspondants dans un dictionnaire et avec les getters et setters appropriés. Ce choix se justifie par le fait qu'il va nous aider à manipuler dans la suite d'une façon aisée notre Environment_type.
- Décorer notre arbre syntaxique avec les typages stockés dans l'Environment_type et Environment_exp. Ces deux environnements sont enrichis au fur et à mesure qu'on effectue les trois passes de la vérification contextuelle.

## 2.2 Etape C

Cette partie est moins guidée que les deux premières étapes, d'autre part, l'implémentation pour le sous-langage *Hello-World* est déjà disponible. Donc on a pu passer plus de temps sur les phases d'analyse et de conception.

Dans la phase d'analyse de chaque cycle, on s'est bien documenté pour avoir une vision globale de ce que l'on devait faire pour chaque sous-langage. Au début, on ne savait pas comment générer des instructions assembleur et on a dû revoir l'implémentation du sous-langage *Hello-World* pour savoir comment cela fonctionne. La phase de conception consiste à implémenter des structures de données spécifiques pour que la gestion de mémoire s'adapte à chaque sous-langage. La phase de codage et la validation ont été effectuées en parallèle : on a implémenté le code en suivant les règles de la grammaire attribuée de décompilation, et en ajoutant des mini-tests ceci jusqu'à ce que toutes les règles soient implémentées.

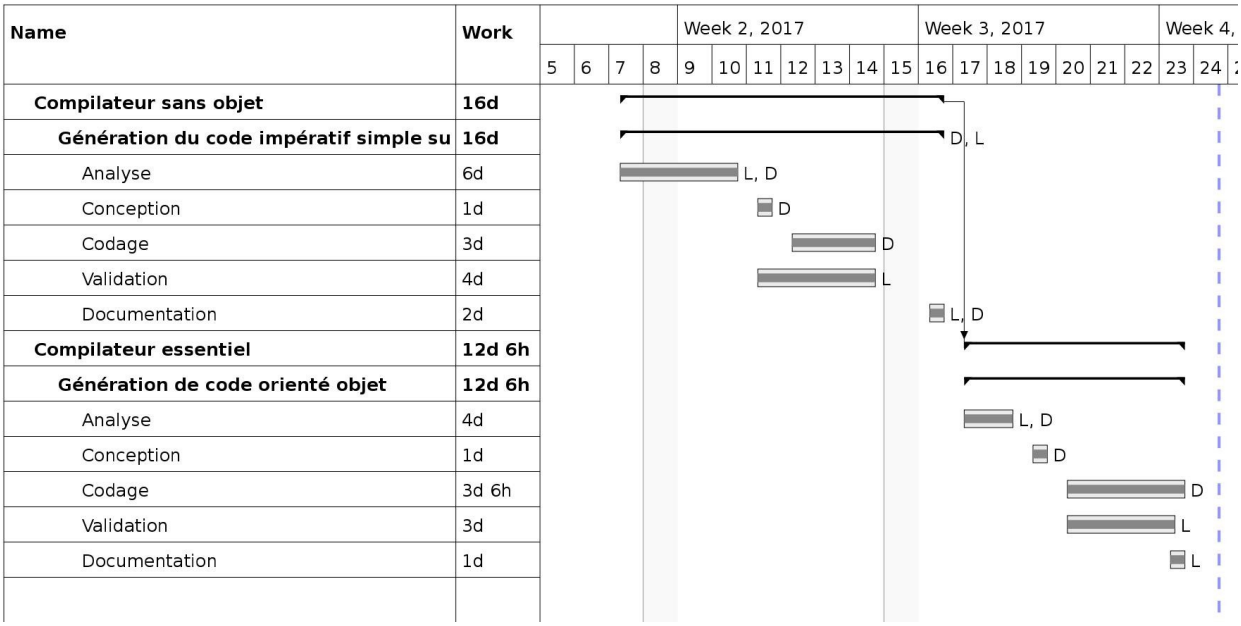


FIGURE 1 – Déroulement de l'étape C.

# 3 Enseignements tirés du projet

## 3.1 Sur la gestion de projet

Ce projet est une expérience nécessaire dans notre formation d'ingénieur. En effet, ce projet nous donne un avant-goût des problèmes que nous rencontrerons en milieu professionel. Il nous a permis d'expérimenter le travail en équipe et d'apprendre plus sur la gestion de projet.

Nous avons tout d'abord dû nous organiser mais aussi créer un planning du projet grâce à l'outil planner. Ainsi nous avons dû prévoir le temps de chaque partie et surtout il a fallu savoir s'adapter aux changements. Effectivement, les taĉhes ont rarement pris le temps prévu et il y a eu de nombreux imprévus (absences, maladies). Nous avons appris qu'il fallait faire avec et s'en accommoder.

Durant le projet nous avons dû communiquer fréquemment entre nous. En effet, nous devions savoir ce que chacun faisait ou devait faire pour ne pas perdre de temps et discuter à propos de l'implémenation des parties en cas de problème. Cependant malgré la mise en place des règles de communication dans la charte d'équipe, nous n'avons quand même pas su communiquer suffisamment, chose qui a ralenti l'avancement du projet.

Nous avons aussi appris qu'il aurait été nécessaire d'avoir un chef de projet avec plus d'autorité. Un chef de projet qui puisse avoir une meilleure vu d'ensemble du projet et qui prenne des décisions de manière plus rapide. Pour cela le chef de projet aurait dû être bien plus a l'écoute des autres membres pour se rendre compte assez tôt des problèmes rencontrés dans la partie C et réagir plus vite. En effet, une raison pour laquelle nous n'avons pas pu terminer le projet est due au fait que nous nous sommes rendus compte trop tard de certains problèmes.

Nous avons aussi appris l'importance de la préparation des suivis afin qu'ils soient plus efficaces.

## 3.2 Sur la programmation

Ce projet nous a appris l'importance des tests dans la réalisation d'un projet et comment utiliser l'outil Cobertura. En effet, jusqu'ici nous ne nous souciions jamais de la couverture de nos tests. Nous avons donc appris que le nombre de tests n'était pas le point le plus important mais bien leur recouvrement. Nous aurions dû faire plus de tests plus tôt dans le projet. Nous avons aussi appris à utiliser l'outil git dans un groupe plus grand qu'auparavant.