Compte Rendu des Réunions

Semaine du 05 – 09 Janvier 2026

Équipe : 43G8 | Rédacteur : TOUATI Ayoub

Récapitulatif

N°	Date	Horaires	Durée	Objet
2	Jour. 06/01	16h30–18h45	2h 15	Point d'avancement global
Total (Volume Horaire de toutes les réunions) :				**7 h 28**

Réunion 2 – Jour 06/01 (16h30–18h45)

Lieu : Visioconférence (Teams) | **Présents :** 5/5

Objectifs

Présentation de l'état d'avancement de chaque étape du projet (Étapes A, B et C).

Ordre du jour

1. Tour de table et présentation de l'avancement individuel
2. État d'avancement de l'Étape A (Analyse lexicale et syntaxique)
3. État d'avancement de l'Étape B (Analyse contextuelle)
4. État d'avancement de l'Étape C (Génération de code)
5. Discussion des points bloquants et prochaines étapes

État d'avancement – Étape A : Analyse lexicale et syntaxique

Responsable : MOUNTASSIR Hamza

Travaux réalisés :

- Implémentation complète du `DecaLexer` pour l'analyse lexicale
- Implémentation complète du `DecaParser` pour l'analyse syntaxique
- Construction de l'arbre syntaxique abstrait (AST) conforme à la grammaire du langage Deca

Éléments non traités :

- Instruction `case`
- Opérateur `instanceof`

Statut : **Complété** (hors extensions)

État d'avancement – Étape B : Analyse contextuelle

Responsables : ARDAN Fatima-Azzahra - EL GOUIJ Faiçal

Travaux réalisés :

- Implémentation de la vérification contextuelle pour Deca **sans objet** et **avec objet**
- Implémentation des décorateurs pour l'ensemble des règles de la grammaire contextuelle, conformément au polycopié
- Gestion complète des erreurs contextuelles avec rapport détaillé :
  - Référencement de chaque règle par son numéro dans le polycopié
  - Indication du fichier Java correspondant et de la partie de code concernée
  - Exemples de cas d'erreurs pour chaque règle
- Création des fichiers nécessaires pour Deca sans objets, chacun équipé des méthodes :
  - `decompile()`
  - `prettyPrintChildren()`
  - `iterChildren()`
- Implémentation de plusieurs tests d'isolation (.deca) pour les parties A et B
- Correction des erreurs détectées dans le `DecaParser` et les fichiers Java associés
- Implémentation des tests unitaires avec JUnit
- Prise en main de l'outil Jacoco pour la couverture de code

Éléments non traités :

- Instruction `case`
- Opérateur `instanceof`

Statut : **Complété** (hors extensions)

État d'avancement – Étape C : Génération de code

Responsable : EL ARABI Mohammed - TOUATI Ayoub

Travaux réalisés :

- Implémentation de `codeGenExpr` pour la génération du code d'évaluation des expressions arithmétiques et des expressions de conversion (`int` → `float`)
- Implémentation de `codeGenBool` pour la génération du code des expressions booléennes élémentaires, conformément à la section 7.2 du document Gencode (p. 221–222)
- Implémentation de `codeGenExprBool` pour factoriser la génération de code des expressions booléennes :
  - Appel de `codeGenBool` pour chaque opérateur booléen
  - Invocation depuis `codeGenExpr` pour respecter l'architecture globale
- Implémentation de `codeGenBool` pour les opérateurs de comparaison : `==`, `!=`, `<`, `>`, `<=`, `>=`
- Implémentation de `codeGenExpr` pour :
  - Les déclarations de variables globales (adressage via `GB`)
  - Les identificateurs, avec une méthode `getAddress` pour récupérer l'adresse mémoire de l'opérande

— Implémentation de `codeGenInst` pour la génération de code des structures de contrôle conditionnelles (`if`/`then`/`else`, `while`), conformément à la section 8 du document Gencode (p. 225)

— Mise en place d'une classe `Helper` pour automatiser la gestion des registres :

  — Allocation et libération des registres

  — Respect de la convention R2...RMAX

— Mise en place d'une classe `Helper` dédiée à la gestion de la pile :

  — Sauvegarde et restauration via `PUSH`/`POP` en cas de saturation des registres

  — Anticipation de la prise en compte de `TSTO`

Travaux en cours :

— Implémentation d'une classe factorisant le pattern de génération de code pour les opérations arithmétiques binaires (`ADD`, `SUB`, `MUL`...) afin d'éviter la duplication de code et de centraliser la gestion registres/pile

Statut : En cours

Synthèse et décisions

Étape	Avancement	Responsable(s)	Statut
Étape A	Analyse lexicale & syntaxique complète	MOUNTASSIR	Complété
Étape B	Vérification contextuelle complète	ARDAN , EL GOUIJ	Complété
Étape C	Génération de code en cours	EL ARABI, TOUATI	En cours

Points d'attention :

— Les extensions `case` et `instanceof` n'ont pas encore été traitées

— La gestion des registres et de la pile pour la génération de code est en cours de finalisation

Prochaines étapes :

— Finaliser l'implémentation de la génération de code (Étape C)

— Poursuivre l'écriture des tests unitaires et d'intégration

— Augmenter la couverture de code avec Jacoco

— Préparer la documentation pour le prochain livrable SHEME

Fait le 07 janvier 2026

Équipe 43G8