

# Projet Génie Logiciel

## Document de Validation

Réalisé par :  
Mouez JAAFOURA, Ilyas MIDOU, Meriem KAZDAGHLI,  
Salim KACIMI, Mehdi DIMASSI

**Date :** 23janvier 2025



Grenoble INP - Ensimag & Phelma  
Université Grenoble Alpes

## Table des matières

<b>1</b>	<b>Organisation des tests</b>	<b>2</b>
1.1	Catégories de Tests . . . . .	2
1.2	Importance des Tests Valides et Invalides . . . . .	2
<b>2</b>	<b>Répertoire des Scripts</b>	<b>2</b>
2.1	Nécessité de l'Automatisation des Tests . . . . .	3
2.2	Scripts Disponibles . . . . .	3
2.3	Méthode pour la validation du compilateur . . . . .	3
<b>3</b>	<b>Résultats de Jacoco</b>	<b>4</b>
<b>4</b>	<b>Conclusion</b>	<b>4</b>

## 1 Organisation des tests

Dans le répertoire **java/fr/ensimag/deca** se trouvent les tests unitaires, classés selon les classes Java correspondantes, avec une division similaire à celle du répertoire de développement principal. Tous les fichiers liés à la validation du compilateur sont regroupés dans ces répertoires. Et donc le répertoire courant **java/fr/ensimag/deca** est défini comme étant celui-ci :

```
|->deca
  |->syntax
    |->valid
    |->invalid
  |->context
    |->valid
    |->invalid
  |->codegen
    |->valid
    |->invalid
  |->extension
    |->syntax
      |->valid
      |->invalid
    |->context
      |->valid
      |->invalid
    |->codegen
      |->valid
      |->invalid
```

### 1.1 Catégories de Tests

Cette configuration distingue plusieurs catégories de tests :

- **Syntax** : Validation de la structure syntaxique du code
- **Context** : Vérification des règles contextuelles et sémantiques
- **Codegen** : Tests de génération de code
- **Extension** : Tests pour les fonctionnalités supplémentaires

### 1.2 Importance des Tests Valides et Invalides

La subdivision entre tests valides et invalides est cruciale :

- **Tests Valides** : Évaluent la conformité du compilateur aux spécifications
- **Tests Invalides** : Identifient et vérifient la détection des erreurs

Cette approche exhaustive garantit une validation complète du compilateur, en s'assurant non seulement du bon fonctionnement du code correct, mais aussi de la capacité à détecter et signaler correctement les erreurs.

## 2 Répertoire des Scripts

Le répertoire **script** contient l'ensemble des scripts shell utilisés pour automatiser la validation des tests.

## 2.1 Nécessité de l'Automatisation des Tests

L'exécution manuelle des tests présente plusieurs inconvénients :

- Consommation importante de temps
- Risque élevé d'erreurs humaines
- Difficulté à vérifier l'absence de régressions lors de l'ajout de nouvelles fonctionnalités

L'automatisation devient ainsi cruciale pour :

- Économiser du temps
- Identifier rapidement les erreurs de développement
- Assurer une couverture complète des tests

## 2.2 Scripts Disponibles

Les scripts suivants ont été développés :

**basic-context.sh** Exécute les scripts de tests contextuels fournis par les professeurs sur les répertoires `context/valid` et `context/invalid`

**basic-synt.sh** Exécute les scripts de tests syntaxiques fournis par les professeurs sur les répertoires `syntax/valid` et `syntax/invalid`

**basic-gencode.sh** Teste le compilateur sur les fichiers de génération de code et exécute la commande `ima` sur les tests générés

**basic-extension.sh** Effectue des tests séquentiels :

- Tests syntaxiques
- Tests contextuels
- Génération de code assembleur

**test\_options.sh** Vérifie le bon fonctionnement de toutes les options du compilateur

## 2.3 Méthode pour la validation du compilateur

Les testeurs écrivent le plus de tests de conformité possible afin de couvrir tous les détails et de détecter un maximum d'erreurs dans le développement. Les parties liées aux tests et à la programmation sont indépendantes, ce qui signifie que le testeur n'est pas informé du code en cours d'écriture par le développeur.

Après la création de cette base de tests, le code est testé. Si des modifications sont nécessaires, elles sont effectuées sur le code afin de corriger les erreurs identifiées.

### 3 Résultats de Jacoco

Jacoco est l'outil privilégié pour analyser la couverture de nos tests. Il permet d'identifier précisément les parties du code qui sont effectivement prises en compte lors de l'exécution des tests intégrés à Maven. Actuellement, la couverture de nos tests est de 73%. Cela signifie que 27% du code n'est pas encore couvert par nos tests. Le principal point faible se situe dans la partie "extension" du compilateur, qui n'est pas encore suffisamment testée. Il faudra donc développer davantage de tests pour cette fonctionnalité afin d'atteindre notre objectif de 90% de couverture. Une fois cette amélioration réalisée, nous disposerons d'une suite de tests plus complète, nous garantissant une meilleure fiabilité et un meilleur contrôle de la qualité du compilateur Deca.

#### Deca Compiler
















Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
fr.ensimag.deca.syntax		63%		47%	653	851	920	2,497	334	450	10	55
fr.ensimag.deca.tree		83%		70%	259	1,040	378	2,839	80	650	1	96
fr.ensimag.deca.context		68%		61%	111	252	151	461	75	182	6	31
fr.ensimag.deca		65%		55%	45	102	104	289	14	59	0	5
fr.ensimag.ima.pseudocode		79%		75%	23	82	35	176	18	72	1	25
fr.ensimag.ima.pseudocode.instructions		77%	n/a	n/a	16	62	28	111	16	62	13	54
fr.ensimag.deca.tools		89%		75%	5	22	5	47	2	16	0	3
fr.ensimag.deca.codegen		97%		90%	4	32	3	87	2	21	0	2
Total	8,054 of 30,077	73%	711 of 1,779	60%	1,116	2,443	1,624	6,507	541	1,512	31	271

FIGURE 1 – Couverture globale du compilateur

### 4 Conclusion

Ce document met en lumière les différentes étapes de validation du compilateur Deca, allant de l'organisation des tests jusqu'à l'analyse de leur couverture avec Jacoco. Grâce à une structure bien définie des répertoires et à l'automatisation des tests via des scripts dédiés, nous avons pu optimiser nos efforts de validation tout en minimisant les erreurs humaines.

Cependant, bien que la couverture actuelle de nos tests atteigne 73%, elle reste en deçà de notre objectif de 90%. Ce résultat reflète le besoin d'ajouter davantage de tests, en particulier pour la partie "extension" du compilateur. Ces efforts supplémentaires permettront non seulement d'améliorer la fiabilité du compilateur, mais également de renforcer sa robustesse face aux cas d'utilisation plus complexes.

Enfin, la méthodologie adoptée dans ce projet pose des bases solides pour les futures étapes de développement et de maintenance. La combinaison d'une organisation claire, d'outils performants comme Jacoco, et d'une stratégie d'automatisation garantit une validation complète et efficace du compilateur Deca. Ces avancées témoignent de notre engagement à l