

Projet Génie Logiciel

# Compilateur Decac

## Manuel de validation

---

**Équipe GL2**

**Étudiants : Élina Houdé, Julien Pinto Da Fonseca,  
Léa Solo Kwan, Yin Xu**

---

# Sommaire

|  |          |
|--|----------|
| <b>Introduction</b>                                      | <b>3</b> |
| <b>1. Descriptif des tests</b>                           | <b>4</b> |
| 1.1. Les différents types de tests                       | 4        |
| 1.2. Organisation des tests unitaires et de précondition | 4        |
| 1.3. Organisation des tests système                      | 4        |
| 1.2.1. Génération du code (dossier codegen)              | 5        |
| 1.2.2. Analyse contextuelle (dossier context)            | 5        |
| 1.2.3. Analyse syntaxique (dossier syntax)               | 6        |
| <b>2. Résultats de couverture du code (Cobertura)</b>    | <b>6</b> |
| <b>3. Scripts de tests automatisés</b>                   | <b>6</b> |
| <b>4. Gestion des risques et gestion des rendus</b>      | <b>8</b> |
| <b>5. Méthodes de validation hors tests</b>              | <b>8</b> |

## Introduction

Dans ce document, vous trouverez les différents tests et systèmes de validation mis en place.

## 1. Descriptif des tests

### 1.1. Les différents types de tests

Au sein du projet, nous pouvons retrouver plusieurs types de test :

- les **tests de précondition**, afin de lever l'exception *IllegalArgumentException* lorsqu'un argument d'une méthode publique ou d'un constructeur ne vérifie pas une précondition ;
- les **tests unitaires**, afin de valider la non-régression du code. Ceux-ci sont mis en place dans le but de couvrir à minima 80% du code nouveau, c'est-à-dire du code ne provenant pas du commit initial ;
- les **tests système**, afin de valider la non-régression des fonctionnalités. Il s'agit de vérifier que lors de la mise en place d'une fonctionnalité, les programmes .deca qui doivent s'exécuter correctement génèrent toujours les mêmes résultats pour chacune des étapes du compilateur et que les programmes .deca devant provoquer une erreur continuent de provoquer la même erreur.

### 1.2. Organisation des tests unitaires et de précondition

-- Disponible plus tard --

### 1.3. Organisation des tests système

Tous les programmes deca de test sont répartis en trois catégories dans le dossier src/test/deca.

### 1.2.1. Génération du code (dossier codegen)

L'étape de génération de code est traitée de par une base de tests systèmes interactif et une base de tests valides. Les programmes de tests interactifs se situent dans le sous-dossier **interactive** et sont généralement testés manuellement. Les programmes de test valides se situent dans le sous-dossier **valid**.

Chemin général depuis la racine du projet :

***src/test/deca/codegen/***

Chacun des tests possède :

- un fichier **.expected** contenant le résultat attendu en sortie de l'étape de vérification syntaxique. Ce fichier est placé dans le dossier **src/test/deca/syntax/valid**.
- un fichier **.expected** contenant le résultat attendu en sortie de l'étape de vérification contextuelle. Ce fichier est placé dans le dossier **src/test/deca/context/valid**.

et, dans le cas spécifique des programmes deca du sous-dossier **codegen/valid** :

- un fichier **.expected** contenant le résultat attendu lors de l'exécution du code assembleur généré. Ce fichier est placé dans le dossier **src/test/deca/codegen/valid**.

### 1.2.2. Analyse contextuelle (dossier context)

Dans ce dossier se situent tous les fichiers de test invalides pour l'étape de vérification contextuelle du code. Chacun de ces tests possède :

- un fichier **.expected** contenant le résultat attendu en sortie de l'étape de vérification syntaxique. Ce fichier est placé dans le dossier **src/test/deca/syntax/valid**.

- un fichier `.expected` contenant l'erreur attendu en sortie de l'étape de vérification contextuelle. Ce fichier est placé dans le dossier `src/test/deca/context/invalid`.





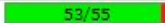
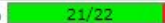



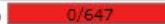
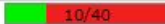
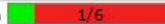




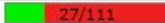
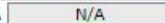
### 1.2.3. Analyse syntaxique (dossier syntax)

Dans ce dossier se situent tous les fichiers de test invalides pour l'étape de vérification syntaxique du code. Chacun de ces tests possède :

- un fichier `.expected` contenant l'erreur attendu en sortie de l'étape de vérification syntaxique. Ce fichier est placé dans le dossier `src/test/deca/syntax/invalid`.

## 2. Résultats de couverture du code (Cobertura)

### Coverage Report - All Packages

| Package  | # Classes | Line Coverage   | Branch Coverage   | Complexity |
|--|-----------|---|---|------------|
| <b>All Packages</b>                                    | 232       | 11 %  422/3593 | 5 %  56/950  | 1,58       |
| <a href="#">fr.ensimag.deca</a>                        | 6         | 10 %  22/216   | 0 %  0/99    | 3,065      |
| <a href="#">fr.ensimag.deca.codegen</a>                | 4         | 96 %  53/55    | 95 %  21/22  | 1,625      |
| <a href="#">fr.ensimag.deca.context</a>                | 21        | 20 %  36/174   | 0 %  0/8     | 1,097      |
| <a href="#">fr.ensimag.deca.syntax</a>                 | 50        | 0 %  0/1932    | 0 %  0/647   | 1,896      |
| <a href="#">fr.ensimag.deca.tools</a>                  | 4         | 25 %  10/40    | 16 %  1/6    | 1,417      |
| <a href="#">fr.ensimag.deca.tree</a>                   | 67        | 24 %  217/890  | 20 %  31/148 | 1,413      |
| <a href="#">fr.ensimag.ima.pseudocode</a>              | 26        | 32 %  57/175   | 15 %  3/20   | 1,176      |
| <a href="#">fr.ensimag.ima.pseudocode.instructions</a> | 54        | 24 %  27/111   | N/A  N/A     | 1          |

Report generated by [Cobertura](#) 2.1.1 on 03/05/20 21:30.

## 3. Scripts de tests automatisés

Afin d'exécuter de manière automatique les tests système et éviter toute régression fonctionnelle, nous avons mis en place plusieurs scripts Shell :

- **basic-synt.sh** : exécute les tests touchant à l'analyse syntaxique ;
- **basic-context.sh** : exécute les tests de résultat de l'analyse contextuelle ;
- **basic-codegen.sh** : exécute les tests de l'étape de génération de code assembleur.

Chemin depuis la racine du projet :

***src/test/script/***

Ces différents scripts sont lancés de façon systématique lors de l'exécution de la commande ***mvn test***, commande permettant également d'exécuter l'ensemble des tests unitaires.

#### **4. Gestion des risques et gestion des rendus**

-- Disponible plus tard --

#### **5. Méthodes de validation hors tests**

-- Disponible plus tard --