Documentation du Compilateur Deca

Roboam Guillaume Raballand Cryprien Talbi El Mehdi
Charles-Mennier Matéo Altieri Aubin

16 janvier 2025
Année académique 2024-2025

Contents

1 Introduction

1.1 Objectif du manuel

Ce manuel a pour objectif de fournir aux utilisateurs du compilateur Deca une documentation concise et pratique. Il s'adresse aux utilisateurs qui disposent déjà des spécifications détaillées du langage Deca et vise à les guider dans l'utilisation du compilateur, tout en précisant les limitations et les particularités de son implémentation.

1.2 Présentation du compilateur Deca réalisé

Le compilateur Deca est conçu pour traduire du code écrit dans le langage Deca (un sous-langage de Java) en code assembleur. Il prend en charge les principales fonctionnalités du langage tout en offrant des extensions et des options de configuration permettant d'adapter son comportement aux besoins spécifiques des utilisateurs.

Le langage Deca complet a été implémenté, ce qui fournit une prise en charge totale des aspects non objets et objets, notamment :
- Création et manipulation de classes, méthodes et champs.
- Utilisation des casts et de `instanceof`.

2 Limitations et particularités de l'implémentation

2.1 Fonctionnalités non implémentées ou partiellement implémentées

L'option `-w` pour afficher des warnings de compilation n'a pas été implémentée. Les seuls détails de compilation affichables sont disponibles via l'option `-d` (debug). Voir la section 4 pour plus d'informations.

2.2 Limitations pour l'utilisateur

Sur l'inclusion de fichiers avec `#include`, il est conseillé à l'utilisateur d'importer uniquement des fichiers déjà syntaxiquement corrects pour éviter des problèmes de reconnaissance de tokens.

Par exemple, si le premier fichier : `print_1.decah`

```
{
    print
```

Deuxième fichier : `print_2.decah`

```
ln("ok");
}
```

Fichier principal :

```
#include "print_1.decah"
#include "print_2.decah"
```

Cela produira une erreur lors de l'analyse syntaxique. En effet même si le programme est correct syntaxiquement, on ne peut pas couper une instruction en 2 fichiers si le 1er morceau est une instruction valide.

3 Messages d'Erreur

Les messages d'erreurs du compilateur sont rédigés afin de respecter ce formatage :

```
<nom de fichier.deca>:<ligne>:<colonne>: <description
   informelle du probleme>
```

3.1 Erreurs de l'analyse lexicale et syntaxique

3.1.1 Erreurs de lexeur

- **Token Recognition** :

  ```
  Token recognition error at '<token>'.
  ```

  Une expression écrite n'existe pas en déca.

3.1.2 Erreurs de parseur

Voici la liste des erreurs qui peuvent être levées par le compilateur lors de la première vérification :

- **Missing Input** :

  ```
  Missing '<input name>' at '<location of expected input>'
  ```

  Un caractère ou une expression n'ont pas été écrits alors qu'on en attendait.

- **Mismatched Input** :

  ```
  Mismatched input '<input name>' expecting '<input1>', '<
     input2>', ...
  ```

  Un caractère ou une expression ont été écrits alors qu'on en attendait d'autres.

- **No Viable Alternative** :

  ```
  No viable alternative at input '<input name>'
  ```

  Un caractère ou une expression ont été écrits alors qu'on ne trouve rien à faire avec.

- **InvalidLValue** :

  ```
  Left-hand side of assignment is not an lvalue.
  ```

  Le type à gauche d'une affectation est incorrect syntaxiquement.

- **IntOverflow** :

  ```
  Error: Integer Overflow.
  ```

  La valeur d'entier entrée n'est pas codable comme un entier signé positif sur 32 bits.

- **FloatOverflow** :

```
Error: Float Overflow.
```

La valeur de `float` entrée est trop grande, telle que celle-ci est arrondie au flottant IEEE-754 simple précision le plus proche, qui est l'infini.

- **FloatUnderflow** :

```
Error: Float Underflow.
```

La valeur de `float` non nulle entrée est trop petite, telle que celle-ci est arrondie au flottant IEEE-754 simple précision le plus proche, qui est zéro.

- **CircularInclude** :

```
Error: Circular Include Detected
```

- **IncludeFileNotFound** :

```
Error: Include File Not Found
```

3.2 Erreurs contextuelles

3.2.1 Passe 1

- **Classe fille sans parent** :

```
"Error: The parent is not defined"
```

Déclaration avec le mot clé `extends` d'une classe fille sans parent.

- **Classe fille de parent non classe**

```
"Error: The parent is not a class"
```

Déclaration avec le mot clé `extends` d'une classe fille dont le parent n'est pas une classe.

- **Déclaration multiple** :

```
"Error: Multiple declaration of '<class name>' , first
   declaration at <localisation>"
```

Déclaration multiple d'une classe.

3.2.2 Passe 2

Champs

- **Déclaration multiple de champ** :

```
"Error: A field with the same name is already declared at
    <localisation>"
```

Déclaration multiple de champ.

- **Champ de type void**

```
"Error: 'void' cannot be used as a type for field
  declaration "
```

Il n'y a pas la possibilité de créer un champ de type `void`.

- **Déclaration étrangère** :

```
"Error: This name is already used for a non field objet
  at <localisation>"
```

Le nom du champ est déjà utilisé dans un autre environnement.

Méthodes

- **Méthode de type void** :

```
"Error: Method parameters cannot have a void type"
```

Il n'y a pas la possibilité de créer de méthode de type `void`.

- **Redéfinition de méthode (signature différente)** :

```
"Error: redefinition of a method with a different
  signature"
```

- **Redéfinition de méthode (type invalide)** :

```
"Error: redefinition of a method with incompatible type"
```

Paramètres

- **Paramètre de type void** :

```
"Error: Method parameters cannot have a void type"
```

Il n'y a pas la possibilité de créer de paramètre de type `void`.

- **Déclaration multiple d'un paramètre** :

```
"Error: Multiple declaration of parameter 'param name',
  first declaration at <localisation>"
```

3.2.3 Passe 3

- **Identifier non initialisé** :

```
"Error: Identifier <name> is undefined"
```

  Si un identifier est appelé sans être initialisé.

- **Type non défini** :

```
"Error: Type <name> is not defined"
```

- **Affectation** :

```
"Error: LValue identifer must be a field, parameter or
   variable"
```

  Mauvais identifier dans le terme gauche d'une affectation.

- **Déclaration variable** `void` :

```
"Error: void cannot be used as a type for variable
   declaration"
```

- **Utilisation de this dans Main** :

```
"Error: Illegal use of 'this' in main block"
```

- **Mauvaise sélection** :

```
"Error: Selection identifer must be a field"
```

  Selection sur un identifier qui n'est pas un champ de classe.

- **Selection sur une classe** :

```
"Error: Left operand of a selection must be a class"
```

  La selection n'a pas été faite sur une classe.

- **Mauvais appel** :

```
"Error: MethodCall identifer must be a method"
```

  Appel sur un identifier qui n'est pas une méthode de classe.

- **Paramètres** :

```
"Error: Too many arguments in method call"
"Error: Too few arguments in method call"
```

  Lors de l'appel d'une méthode, si trop ou trop peu d'arguments.

- **Déclaration variable multiple** :

```
"Error: Multiple declaration of <name>, first declaration
    at <localisation>"
```

- **Création d'objet** :

```
"Error: Cannot create an object from a non class type"
```

Création d'un objet avec `new` d'un type qui n'est pas une classe.

- **Champ protégé** :

```
"Error: Unauthorized access to the protected field <name
   >"
```

Tentative d'accès illégale à un champ protégé.

- **Return** :

```
"Error: Main does not return anything"
"Error: This method does not return anything"
```

Tentative de return quelque chose dans le main, ou de return quelque chose pour une méthode qui ne renvoie rien.

Erreurs de type

- **Expressions de contrôle** :

```
"Error : Expected expression type is boolean, got <type>"
```

Dans un while ou un if, ou une expression attendue doit être de type booléenne.

- **Opération arithmétique** :

```
"Error: Incompatible types for arithmetic operation: <
   type1> <operateur> <type2>"
```

Une opération arithmétique peut s'effectuer seulement entre des entiers et des flottants.

- **Modulo** :

```
"Error: Incompatible types for arithmetic operation: <
   type1> <operateur> <type2>"
```

Il s'agit du cas spécial d'une opération arithmétique, qui peut être effectué seulement entre deux `int`.

- **Opération boooléenne** :

```
"Error: Incompatible types for boolean operation: <type1>
    <operateur> <type2>"
```

- **Not** :

```
"Error: Incompatible types for <operateur> and type <type
  >"
```

  Il faut forcément une condition après un `not`.

- **UnaryMinus** :

```
"Error: Incompatible types for <operateur> and type <type
  >"
```

  Les types sur lesquels UnaryMinus peut être appliqué sont `int` et `float`.

- **Comparaisons** :

```
"Error: Incompatible types for comparison: <type1> <
  operateur> <type2"
```

- **Print** :

```
"Error: Expected expression type is int, float or string,
   got <type>"
```

  Les seuls types qui peuvent être passés dans `print` sont `int`, `float` et `string`.

- **InstanceOf** :

```
"Error: Incompatible types for operation: <type1>
  instanceof <type2>"
```

  L'opération InstanceOf ne peut être effectuée que si `<type1>` est une classe ou null et que `<type2>` est une classe.

- **Type affectation** :

```
"Error: Illegal RValue type, got <type1> expected <type2
  >"
```

3.3 Erreurs lors de l'exécution du code

Liste des erreurs potentielles liées à l'exécution du code assembleur généré.

- **Divison par 0** :

```
"Error: division by 0"
```

  Division entière et reste de la division entière par 0.

- **Débordement arithmétique sur les flottants** :

```
"Error: Overflow during arithmetic operation"
```

- **Absence de `return` lors de l'exécution d'une méthode** :

```
"Error: return instruction missing in <method>"
```

- **Conversion de type impossible** :

- **Déférencement de `null`** :

```
"Error: return instruction missing in <method>"
```

  La valeur entrée n'est pas du type attendu.

- **Erreur de lecture** :

```
"Error: return instruction missing in <method>"
```

  La valeur entrée n'est pas du type attendu.

- **Débordement mémoire** :

```
"Error: Heap Overflow"
"Error: Stack Overflow"
```

  Débordement mémoire dans la pile ou le tas.

- **Accès à des variables non initialisées** :

- **Assembleur** : Utilisation d'une méthode écrite en assembleur non compatible avec l'assembleur généré par le compilateur.

4 Description des Options du Compilateur

La syntaxe d'utilisation de l'exécutable `decac` est :
Usage: decac [[-p — -v] [-n] [-r X] [-d]* [-P] [-a] [-e] <fichier .deca>...] — [-b]
On peut appeler la commande `decac` avec un ou plusieurs fichiers sources Deca.

4.1 Liste des options disponibles

`-b (banner)` Affiche une bannière indiquant le nom de l'équipe.

`-p (parse)` Arrête `decac` après l'étape de construction de l'arbre et affiche la décompilation de ce dernier (i.e. s'il n'y a qu'un fichier source à compiler, la sortie doit être un programme `deca` syntaxiquement correct).

`-v (verification)` Arrête `decac` après l'étape de vérifications (ne produit aucune sortie en l'absence d'erreur).

`-n (no check)` Supprime les tests à l'exécution spécifiés dans les points 11.1 et 11.3 de la sémantique de Deca.

`-r X (registers)` Limite les registres banalisés disponibles à `R0 ... R{X-1}`, avec $4 \leq X \leq 16$.

`-d (debug)` Active les traces de debug. Répéter l'option plusieurs fois pour avoir plus de traces.

```
decac -d LOG INFO
decac -d -d LOG DEBUG
decac -d -d -d LOG WARN
decac -d -d -d -d LOG ALL
```

`-P (parallel)` S'il y a plusieurs fichiers sources, lance la compilation des fichiers en parallèle (pour accélérer la compilation).

`-a X (arithmetic rounding)` Fixe le mode d'arrondi des flottants.

```
SETROUND_TONEAREST : arrondi a la valeur la plus
   proche
(mode d'arrondi initial par defaut)
SETROUND_UPWARD : arrondi a la valeur superieure
SETROUND_DOWNWARD : arrondi a la valeur inferieure
SETROUND_TOWARDZERO : arrondi vers zero.
```

`-e (Java .class generation)` Génère un fichier compilé `.class` en Java à partir du fichier source.

Note : Les options `-p` et `-v` sont incompatibles.

Vous retrouverez le manuel d'utilisation ci-dessus en exécutant l'exécutable `decac` situé dans `src/main/bin/`.

4.2 Présentation de l'extension BYTE

L'objectif de cette extension est de générer le bytecode Java à partir d'un fichier Deca, permettant ainsi de produire des classes .class exécutables directement par la JVM (Java Virtual Machine). Cette extension donne une traduction du code Deca en instructions bytecode, rendant possible l'exécution sans passer par une étape intermédiaire de compilation Java.

4.3 Limitations des extensions

Les `instanceof` et casts ne sont pas encore traduits en bytecode. Dans la version actuelle du code, la partie objet n'est pas encore implémentée, mais sera faite pour la deuxieme version du compilateur qui sortira vendredi 24 janvier. Il y a aussi un problème lors de la génération du bytecode de `println`, qui est confondu avec `print` La génération avec -e sur un fichier plante.