Manuel Utilisateur - Compilateur Deca

GL03 :Benoît GIRARD, Mattéo ROSSILLOL–LARUELLE,
Rania ZOUGARI, Ambroise CHABANOL, Alexandre GRONDIN

Janvier 2025

Table des matières

1. Introduction

1.1. Présentation du compilateur Deca

Ce manuel a pour objectif de guider les utilisateurs dans l'utilisation du compilateur Deca, développé dans le cadre du Projet Génie Logiciel. Il fournit une vue d'ensemble du fonctionnement, des commandes, des limitations connues, ainsi qu'une explication des messages d'erreur.

Le compilateur permet de transformer un programme écrit en langage Deca en un programme assembleur exécutable par la machine virtuelle IMA.

1.2. Objectifs et fonctionnalités principales

Les principaux objectifs et fonctionnalités du compilateur Deca sont :
- Analyser lexicalement et syntaxiquement des programmes Deca
- Effectuer des vérifications contextuelles sur les programmes Deca
- Générer du code assembleur IMA à partir de programmes Deca valides
- Gérer les inclusions de fichiers avec la directive include
- Implémenter les types de base : entiers, flottants, booléens, chaînes de caractères
- Supporter les structures de contrôle : if/else, while
- Gérer les opérations arithmétiques et logiques
- Implémenter la programmation orientée objet : classes, héritage, méthodes
- Fournir des fonctions d'entrée/sortie de base : print, println, readInt, readFloat
- Permettre la surcharge de méthodes
- Gérer les conversions de types (cast)
- Implémenter l'opérateur instanceof
- Fournir une bibliothèque standard minimale
- Générer des messages d'erreur précis en cas de programme invalide
- Offrir différentes options de compilation : -p (parse only), -v (verification only), -n (no check), etc.
- Permettre la compilation parallèle de plusieurs fichiers
- Implémenter des extensions optionnelles comme les fonctions mathématiques avancées

L'objectif pédagogique est de mettre en pratique les concepts de génie logiciel, de compilation et de travail en équipe sur un projet conséquent.

1.3. Outils disponibles

Pour mener à bien ce projet de développement d'un compilateur Deca, notre équipe a mis en place plusieurs outils essentiels :
- **Github** pour la gestion de versions et le travail collaboratif
- **Maven** pour la gestion des dépendances et l'automatisation des builds
- **ANTLR** pour la génération de l'analyseur lexical et syntaxique
- **JUnit** pour les tests unitaires
- **Jacoco** pour l'analyse statique du code

2. Installation et configuration

2.1. Prérequis système

Pour installer et utiliser le compilateur Deca, votre système doit répondre aux prérequis suivants :
- Java Development Kit (JDK)
- Maven
- Git pour la gestion de version
- Un environnement de développement intégré (IDE) compatible avec Java
- Au moins 2 Go de RAM disponible

— Espace disque suffisant pour le projet et ses dépendances (environ 500 Mo)

2.2. Procédure d'installation

Nous avons automatiser l'installation la récupération et le peuplement de la variable d'environnement PATH pour que tout fonctionne avec IMA avec le Makefile Pour installer le compilateur Deca, suivez ces étapes :

1. Clonez le dépôt Git du projet :

   ```
   git clone adresse_projet
   ```

2. Naviguez dans le répertoire du projet :

   ```
   cd deca
   ```

3. Installer IMA :

   ```
   $make USERNAME=<username> ENSIPC=<ensipc> get
   #Pour télécharger l'archive et la décompresser
   ```

4. Compilez le projet avec Maven :

   ```
   mvn clean install
   ```

5. Vérifiez que l'installation s'est bien déroulée en exécutant les tests (Cela lancera tout les tests deca) :

   ```
   mvn test
   ```

2.3. Instructions de construction

— Avant de compiler
`mvn clean`
— Compiler avec Maven
`mvn compile`
— Lancer tous les tests
`mvn test`
— Vérifier la sortie des tests
`mvn verify`

3. Utilisation de base

3.1. Structure d'un programme Deca

Un programme Deca de base a la structure suivante :

```
{
    // Déclarations de variables
    type1 var1;
    type2 var2 = valeur_initiale;

    // Instructions
    instruction1;
    instruction2;
    // ...
}
```

Les éléments principaux d'un programme Deca sont :
— **Accolades** : Le programme principal est encadré par des accolades .
— **Déclarations de variables** : Elles se font au début du bloc, avant les instructions.
— **Instructions** : Elles constituent le corps du programme.

— **Commentaires** : Ils commencent par // pour une ligne ou /* ... */ pour plusieurs lignes.

Les types de bases sont les mêmes qu'en JAVA, Exemple de programme Deca simple :

```
{
    int x = 5;
    float y = 3.14;
    println("La valeur de x est : ", x);
    println("La valeur de y est : ", y);
}
```

3.2. Compilation d'un fichier Deca

Pour compiler un programme Deca, on utilise la commande `decac` suivie du nom du fichier source :

```
decac [options] mon_programme.deca
```

Cette commande va générer un fichier assembleur IMA nommé `monprogramme.ass` dans le même répertoire.
Si plusieurs fichiers sont spécifiés, ils seront compilés séquentiellement :

```
decac fichier1.deca fichier2.deca fichier3.deca
```

3.3. Exécution d'un programme compilé

Une fois le programme compilé, on peut l'exécuter en utilisant l'interpréteur IMA :

```
ima mon_programme.ass
```

L'interpréteur IMA va exécuter les instructions assembleur générées par le compilateur Deca.

3.4. Options de ligne de commande

Le compilateur Deca accepte plusieurs options de ligne de commande pour modifier son comportement :

— `-p` : Arrête la compilation après l'étape de construction de l'arbre syntaxique abstrait et affiche l'arbre obtenu.
— `-v` : Arrête la compilation après l'étape de vérification (ne produit pas de code assembleur).
— `-n` : Ne vérifie pas les débordements lors des opérations arithmétiques.
— `-r` X : Limite le nombre de registres disponibles pour la génération de code à X ($4 \leq X \leq 16$).
— `-d` : Active les traces de debug. Répéter l'option pour augmenter le niveau de détail.
— `-P` : S'il y a plusieurs fichiers sources, lance la compilation en parallèle (par défaut, elle est séquentielle).
— `-b` : Affiche la bannière

4. Fonctionnalités du langage Deca

4.1. Types de données

Deca propose les types de données de base suivants :
— **int** : entiers signés sur 32 bits
— **float** : nombres à virgule flottante simple précision (IEEE 754)
— **boolean** : valeurs booléennes `true` ou `false`
— **String** : chaînes de caractères existe uniquement en tant que RValue

Le langage supporte également :
— Les classes définies par l'utilisateur

— Le type `void` pour les méthodes sans valeur de retour
— Le type `null` pour les références nulles
Deca utilise un typage statique faible, avec vérification des types à la compilation.

4.2. Structures de contrôle

Deca offre les structures de contrôle classiques des langages impératifs :
— **Conditionnelles** :
  — `if (condition) { ... } else { ... }`
  — `if (condition) { ... } else if (condition) { ... } else { ... }`
— **Boucles** :
  — `while (condition) { ... }`
— **Sauts** :
  — `return [expression];`
Contrairement à Java, Deca ne dispose pas des instructions `for`, `do-while`, `switch`, `break` ou `continue`.

4.3. Fonctions et méthodes

Les fonctions et méthodes en Deca sont définies avec la syntaxe suivante :

```
type nomMethode(type1 param1, type2 param2, ...) {
    // Corps de la méthode
    return expression;
}
```

Caractéristiques principales :
— Les paramètres sont passés par valeur pour les types primitifs, par référence pour les autres
— Le type de retour peut être un type de base, une classe ou `void`
— Les méthodes ne peuvent pas être surchargées
— Pas de support pour les méthodes variadiques ou les paramètres par défaut

4.4. Classes et objets

Deca est un langage orienté objet avec les caractéristiques suivantes :
— **Définition de classes** :

```
class NomClasse [extends ClasseParente] {
    // Attributs
    // Méthodes
}
```

— **Héritage simple** : une classe peut hériter d'une seule classe parente
— **Encapsulation** : mots-clés `protected` et implicitement `public`
— **Polymorphisme** : liaison dynamique des méthodes
— **Mot-clé `this`** : pour référencer l'objet courant
— **Instanciation** : utilisation du mot-clé `new`
Deca ne supporte pas les interfaces, les classes abstraites ou les génériques.

4.5. Bibliothèque standard

La bibliothèque standard de Deca est minimaliste et inclut :
— **Entrées/Sorties** :
  — `print(expression)` : affiche une expression
  — `println(expression)` : affiche une expression suivie d'un retour à la ligne

  - `printx(expression)` et `printlnx(expression)` : versions hexadécimales
  - `readInt()` : lit un entier depuis l'entrée standard
  - `readFloat()` : lit un flottant depuis l'entrée standard
- **Classe Object** : classe mère implicite de toutes les classes
- **Classe Math** (Pas notre extention) : fonctions mathématiques de base comme `sin`, `cos`, `sqrt`

La bibliothèque standard est volontairement limitée pour se concentrer sur les aspects fondamentaux de la programmation orientée objet et systèmes.

5. Gestion des erreurs

Descritpion des erreurs qui peuvent être levées.

5.1. Erreurs de ligne de commandes

Ce type d'erreur est levé lors de la lecture de la ligne de commande entrée par l'utilisateur :

- **Invalid argument : <argument>**
  Description : L'argument donné n'est pas autorisé
- **Invalid file supplied, expected : .deca**
  Description : Le fichier n'est pas un fichier deca
- **Bad argument -[b | p | v | r]**
  Description : Mauvais argument dû à un mauvais ordre, une incompatibilité de certaines options ou un doublon par exemple.

5.2. Erreurs lexicographiques

Ce type d'erreur est levé lors de l'analyse lexicale du programme :

- **Integer is too big**
  Description : Le nombre entier fourni dépasse la taille autorisée qui est entiers signés codables sur 32 bits.
- **Float is out of range**
  Description : Le nombre décimal fourni dépasse la taille autorisée.
- **Incorrect string : unfinished or forbidden symbol**
  Description : La chaîne de caractères fournie comporte un symbole interdit ou n'est pas correctement terminée.
- **Unfinished multiline comment**
  Description : Un commentaire multiligne dans le code source n'est pas correctement fermé.
- **Unauthorised type**
  Description : Le type de donnée utilisé n'est pas autorisé ou n'est pas pris en charge dans le contexte donné.

5.3. Erreurs syntaxiques

Ce type d'erreur est levé lors de l'analyse lexicale du programme, les différents types d'erreurs sont :

- **InvalidLValue**
  Description : Le type de donnée utilisé n'est pas autorisé ou n'est pas pris en charge dans le contexte donné (assignation).
- **mismatched input '...' expecting '...'**
  Description : Le mot réservé n'est pas le bon, il est attendu un autre.
  Exemple : Un else sans if ; Un while sans condition ; Un mauvais usage de New...
- **mismatched input '{' expecting '('**
  Description : La déclaration d'un if ou d'un while est faite sans condition.

— **no viable alternative at input '...'**

Description : Un token est présent à un endroit où il ne devrait pas l'être, inversement, n'est pas présent alors il le devrait.

Exemple : Un else avec une condition ; Un nom de méthode commençant par un chiffre ; Un problème lié à un identificateur de confidentialité ; Une méthode sans type de retour...

— **Missing '...' at '...'**

Description : Manque d'un mot réservé ou token dans un endroit où il aurait dû être présent.

Exemple : Pas de parenthèse ouvrante dans une condition if ; Trop de parenthèses/accolades ouvrantes/fermantes...

— **extraneous input '...' expecting '...'**

Description : Le mot ne devrait pas être la

5.4. Erreurs contextuelles

Levées lors de la partie B du compilateur, les erreurs contextuelles s'assurent du bon typage du programme deca. Eles vérifient les erreurs de grammaire.

— **Contextual Error : Identifier not defined**

Description : L'identificateur utilisé n'a pas été préalablement déclaré dans le contexte actuel.

— **Contextual Error : Type or class not defined**

Description : Le type ou la classe utilisé n'a pas été déclaré ou n'est pas accessible dans le contexte actuel.

— **Contextual Error : Class already defined**

Description : La classe que vous essayez de déclarer a déjà été définie précédemment dans le programme.

— **Contextual Error : Super class not defined**

Description : La classe mère spécifiée pour l'héritage n'a pas été déclarée dans le programme.

— **Contextual Error : Field already defined**

Description : Le champ que vous essayez de déclarer a déjà été défini précédemment dans la classe actuelle.

— **Contextual Error : Field couldn't be void**

Description : Un champ ne peut pas être déclaré avec le type void.

— **Contextual Error : Identification name already exists in superclass and is not a field**

Description : Le nom du champ que vous utilisez existe déjà dans la super classe, mais il n'est pas déclaré comme champ.

— **Contextual Error : Method signature differs from the superclass**

Description : La signature de la méthode que vous déclarez est différente de celle définie dans la super classe.

— **Contextual Error : The defined class does not return what was defined in the superclass**

Description : La classe que vous définissez ne renvoie pas le même type que celui défini dans la super classe.

— **Contextual Error : Method already defined**

Description : La méthode que vous essayez de déclarer a déjà été définie précédemment dans la classe actuelle.

— **Contextual Error : Param couldn't be void type**

Description : Le paramètre ne peut pas être déclaré avec le type void.

— **Contextual Error : Param already defined**

Description : Le paramètre que vous essayez de déclarer a déjà été défini précédemment.

— **Contextual Error : Var couldn't be void type**

Description : La variable ne peut pas être déclarée avec le type void.

— **Contextual Error : Var already defined**
Description : La variable que vous essayez de déclarer a déjà été définie précédemment.

— **Contextual Error : Return type differs from the correct type**
Description : Le type de retour de la méthode diffère du type correct.

— **Contextual Error : Only types Int, String, and Float can be printed**
Description : Assurez-vous que le type que vous essayez d'imprimer est parmi les types Int, String et Float.

— **Contextual Error : Types are not consistent**
Description : Les types ne sont pas cohérents pour l'opération ou l'affectation que vous essayez de réaliser.

— **Contextual Error : Type has to be boolean for the condition**
Description : Le type doit être boolean pour la condition spécifiée.

— **Contextual Error : Assign types are not consistent**
Description : Les types des deux côtés de l'opération d'affectation ne sont pas cohérents.

— **Contextual Error : The two operands must be boolean/int or float**
Description : Les deux opérandes doivent être de type boolean, int ou float.

— **Contextual Error : Not an Int**
Description : L'expression spécifiée pour readint n'est pas de type Int.

— **Contextual Error : Not a Float**
Description : L'expression spécifiée pour readfloat n'est pas de type Float.

— **Contextual Error : Only on Float or Int**
Description : Cette opération est uniquement autorisée sur des types Float ou Int.

— **Contextual Error : Only on booleans**
Description : Cette opération est uniquement autorisée sur des valeurs de type boolean.

— **Contextual Error : Protected field**
Description : Le champ spécifié est protégé et ne peut être accédé que dans la sous-classe.

— **Contextual Error : Instanceof requires class types or basic types(boolean, int, float)**
Description : instanceof nécessite un type class ou basique.

— **Contextual Error : Identifier not a class**
Description : L'identificateur spécifié n'est pas une classe.

— **Contextual Error : You can't call 'this' in the main**
Description : Il est impossible d'appeler 'this' dans la fonction main.

— **Contextual Error : Modulo operation only exists for integers**
Description : L'opération modulo n'est possible que sur des nombres entiers.

— **Contextual Error : Object is not a class**
Description : L'objet spécifié n'est pas une classe.

— **Contextual Error : The number of parameters does not match what was defined in the superclass**
Description : Le nombre de paramètres ne correspond pas à ce qui a été défini dans la super classe.

— **Contextual Error : Parameter type(s) is/are incorrect**
Description : Le(s) type(s) du(des) paramètre(s) est/sont incorrect(s).

— **Contextual Error : Not a method**
Description : L'identificateur spécifié n'est pas une méthode.

5.5. Erreurs d'exécution

Enfin, le dernier type d'erreur rencontré dans ce compilateur est le type relatif aux erreurs d'exécution :

— **float Overflow**

Description : division par 0 incluant les flottants

— **zero division**
Description : division par 0 uniquement pour les entiers

— **Can not dereference null pointer**
Description : déréférencement de null.

— **Stack is full**
Description : débordement mémoire de la pile.

— **Method ends without return**
Description : fin d'une méthode sans return.

— **Input/Output error**
Description : erreur de lecture pas bon type d'entrée

— **Heap overflow**
Description : débordement mémoire du tas.