Documentation utilisateur du langage Deca

Grégoire Rabusson Lucas Block Nathan Gicquel
Nicolas Ferlut Gabriel Levy

24 janvier 2021

Table des matières

1 Introduction

Ce document est la documentation utilisateur du langage Deca avec l'utilisation de notre compilateur. Ce compilateur est écrit en Java et permet de générer du code assembleur pour la machine abstraite ima. Certaines portions de code en Java sont générés avec ANTLR4.

Pour compiler et exécuter, on utilise dans le terminal les commandes :

1) **decac code.deca** où code.deca est un fichier écrit dans le langage Deca. Cette commande va générer un fichier code.ass écrit en assembleur.

2) **ima code.ass** qui va éxécuter le fichier.

Les différentes options de compilation :

-b : bannière. Permet d'afficher le nom de membres de l'équipe.

-p : parse. Permet de ne faire que la construction de l'arbre et affiche la décompilation de celui-ci.

-v : verification. Arrête decac après l'étape de vérifications. Aucune sortie s'il n'y pas d'erreur.

-n : no check. Arrête decac après l'étape de vérifications. Aucune sortie s'il n'y pas d'erreur.

-r X : registers. Limite les registres banalisés disponibles à R0 ... RX-1, avec 4 <= X <= 16.

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

-P : s'il y a plusieurs fichiers sources, lance la compilation des fichiers en parallèle (pour accélérer la compilation)

Exemple : **decac -n fichier.deca**

2 Limitations du compilateur

Notre compilateur permet donc de programmer en langage Deca. Cependant, certaines fonctionnalités n'ont pas pu être totalement implementées ou ne fonctionnent pas à la perfection quant à la gestion des classes. En revanche, l'ensemble des opérations arithmétiques, des comparateurs et des déclarations dans un "main" sont fonctionnels et permettent d'ores et déjà de réaliser des algorithmes pertinents.

Pour ce qui est de l'utilisation de classes, il est bien evidemment possible de les instancier, et de gérer les champs et les méthodes déclarer dans celles-

ci. Cependant, il se peux que certaines ereurs de compilation surviennent du à une gestion maladroite des registres, notamement lors de appels des méthodes et la gestion des valeurs retournées. De plus, le terme "null" ne sera pas utilisable pour ce langage, tout comme la fonction "instance of" qui n'ont pas pu être implémentés.

Une spécificité requise par le compilateur est la gestion des nombres qui nécessite un trop grand nombre de bits pour être codé. Ainsi, le langage Deca admet des valeurs maximales pour les différents types de variables. Un int a pour valeur minimal -2,147,483,648 et pour valeur maximal +2,147,483,647. Un float a pour valeur maximal $(2-2^{-23})\times 2^{127} \approx 3.4028235\times 10^{38}$. De même, un flottant inférieur à 2^{-149} sera arrondi à 0. Ces limitations sont dû à la manière dont sont sauvegardés ces types en mémoire (nombre de bits,norme...)
.

Notre compilateur contient tout le langage essentiel. Il contient également les conversions entre int et float, et également entre classes (à condition qu'il y ait un lien d'héritage entre elles). Il ne contient cependant aucun test d'appartenance et donc pas de commande instanceof().

3 Messages d'erreur

Les erreurs dû à la compilation sont de la forme :
fichier.deca :num1 :num2 : description de l'erreur. **fichier.deca** correspond au fichier, **num1** au numéro de ligne et **num2** au numéro de colonne, **description de l'erreur** est ce que nous expliquons dans cette documentation.

3.1 Erreur lexicographique

fichier.deca :ligne :colonne : token recognition error at : "caractère" : "caractère" induit une erreur lexicale qui empêche le compilateur de trouver un token.

nom de fichier + " : include file not found" : erreur lors de l'utilisation d'un include : le fichier n'a pas été trouvé.

"Circular include for file " + name : erreur lors de l'utilisation d'un include. Arrive si une chaîne d'include est circulaire, c'est à dire qu'un fichier inclut un fichier qui a déjà été include.

3.2 Erreur de Syntaxe

nom.deca :ligne :colonne : no viable alternative at input 'texte' : erreur lorsqu'un enchaînement de caractères qui de devrait pas avoir lieu à lieu. 'text' représente le premier token qui est un problème. Par exemple : un oubli de '{' au début du main, un entier assigné avec une valeur autre que 0 et commençant par 0, ou encore une déclaration ayant lieu après une assignation.

nom.deca :ligne :colonne : mismatched input 'commande' expecting {*liste*} : erreur due à une mauvaise utilisation de la fonction commande représentant print, println, printx, printlnx. **liste** représente ce qui doit être entré comme argument de commande.

nom.deca :ligne :colonne : missing 'caractère_manquant' at 'token' : erreur dû à l'oublie du caractère caractère_manquant avant token.

nom.deca :ligne :colonne : missing '}' at '<EOF>' : même erreur qu'au dessus. C'est pour la cas particulier où on oublie un '{' avant la fin du fichier (représenté par EOF).

nom.deca :ligne :colonne : extraneous input 'caractère_extra' expecting {*liste*} : erreur dû à l'utilisation du caractère caractère_extra trop de fois. **liste** représente ce qui peut remplacer le caractère_extra.

nom.deca :ligne :colonne : left-hand side of assignment is not an lvalue : erreur ayant lieu lorsque la partie gauche d'un assigne n'est pas une valeur légale.

3.3 Erreur contextuelle

nom.deca :ligne :colonne : Le type de la valeur à assigner ne se correspond pas au type de la variable (3.28) : erreur ayant lieu si le type de la valeur à assigner ne correspond pas au type de la variable.

nom.deca :ligne :colonne : La condition doit être un booleen (3.29) : erreur ayant lieu si on utilise un opérateur de condition sur une valeur qui n'est pas une condition.

nom.deca :ligne :colonne : Les opérateurs arithmétiques nécessitent deux opérandes de type int ou float ! (3.33) : erreur ayant lieu si on utilise un opérateur arithmétique sur une valeur qui n'est pas un float

ou un int.

nom.deca :ligne :colonne : Pour un opérateur booleen les deux operandes doivent être des booleens (3.33) : erreur ayant lieu si on utilise un opérateur booleen sur une valeur qui n'est pas un booleen.

nom.deca :ligne :colonne : Les opérateurs de comparaison nécessitent deux opérandes de type int ou float ! (3.33) : erreur ayant lieu si on utilise un opérateur de comparaison sur une valeur autre qu'un float ou qu'un int.

nom.deca :ligne :colonne : Ce type ne peut pas être print, il doit être int, float ou string (3.31) : erreur ayant lieu si on utilise la fonction print sur une valeur qui n'est pas int float ou string.

nom.deca :ligne :colonne : La methode n'est pas une methode dans la superclass (2.7) : erreur ayant lieu si le nom d'un attribut ou d'une méthode de classe fille est déjà utilisé (par une méthode ou un attribut) de la classe mère.

nom.deca :ligne :colonne : La methode n'as pas la même signature dans la classe mère (2.7) : erreur ayant lieu lors d'une tentative de réécriture de méthode. On ne peut réécrire une méthode qu'en gardant la même signature que dans sa classe mère.

nom.deca :ligne :colonne : La methode *nom_de_la_methode* est deja définie : erreur ayant lieu si on nomme deux méthodes de la même manière. Cette erreur apparaît sur la deuxième méthode déclarée.

nom.deca :ligne :colonne : Les deux types ne sont pas des classes, on peut pas Cast 3.39 : erreur ayant lieu tente de cast autre chose que des classes.

nom.deca :ligne :colonne : Les deux classes ne sont pas compatibles, on peut pas Cast 3.39 : erreur ayant lieu tente de cast deux variables de classes non compatibles (pas d'héritage entre elles).

nom.deca :ligne :colonne : La classe mère n'est pas définie 1.3 : erreur ayant lieu si on fait hériter une classe d'une classe qui n'est pas définie.

nom.deca :ligne :colonne : La classe *nom_de_la_classe* est deja définie ! : erreur ayant lieu si on déclare deux classes du même nom. Cette erreur apparaît sur la deuxième classe déclarée.

nom.deca :ligne :colonne : La Variable *nom_de_la_variable* est deja définie (3.17) : erreur ayant lieu si on déclare deux variables du même nom. Cette erreur apparaît sur la deuxième variables déclarée.

nom.deca :ligne :colonne : On ne peut pas déclarer un void (2.5) : erreur ayant lieu si on essaye de déclarer un void.

nom.deca :ligne :colonne : Le champ n'est pas un champ dans la superclass (2.5) : erreur ayant lieu si on essaye d'utiliser un champ (méthode ou attribut) qui n'est pas dans la superclass.

nom.deca :ligne :colonne : On ne peut pas déclarer un string (3.1) : erreur ayant lieu si on essaye de déclarer un String.

nom.deca :ligne :colonne : La variable *nom_de_la_variable* n'appartient pas à l'environment (0.1) : erreur ayant lieu si on essaye d'utiliser une variable non définie.

nom.deca :ligne :colonne : Le type n'appartient pas a l'environment (0.2) : erreur ayant lieu si on essaye d'utiliser un type non défini.

nom.deca :ligne :colonne : Le type de retour n'est pas le même que dans la superClass (ou un sous-type)(2.7) : erreur ayant lieu lors d'une tentative de réécriture de méthode. On ne peut réécrire une méthode qu'en gardant le même type de retour que dans sa classe mère.

nom.deca :ligne :colonne : La methode ne retourne rien alors que le type de retour n'est pas Void : erreur ayant lieu s'il n'y a pas de type de retour alors qu'il devrait y en avoir un.

nom.deca :ligne :colonne : On ne peut pas lancer un appel a methode sur un attribut sans classe : erreur ayant lieu si on essaie d'appeler une méthode sur une variable alors que cette variable n'est pas du type class.

nom.deca :ligne :colonne : Vous avez appelé un élément qui n'est pass une methode avec la syntaxe d'une methode : erreur ayant lieu si on essaie d'appeler une méthode alors que l'élément appelé n'est pas une méthode (attribut par exemple).

nom.deca :ligne :colonne : Vous n'avez pas donné le bon nombre d'arguments à la méthode : erreur ayant lieu si on essaie d'appeler une méthode et qu'on ne met pas le bon nombre d'argument dans cet appel.

nom.deca :ligne :colonne : L'identifiant n'est pas un champ : erreur ayant lieu si on essaie d'appeler un attribut alors qu'il n'y en a pas de ce nom.

nom.deca :ligne :colonne : Un champ protégé ne peut pas être appelé depuis le main : erreur ayant lieu si on essaie d'appeler un attribut alors que celui-ci est *protected*.

nom.deca :ligne :colonne : La classe actuelle n'as pas accès au paramètre : erreur ayant lieu si on essaie d'appeler un attribut protected depuis une classe qui n'y a pas accès.

nom.deca :ligne :colonne : Pour un modulo il faut deux int (3.33) : erreur ayant lieu si on essaie de faire un modulo sur autre chose que des int.

nom.deca :ligne :colonne : Le Not ne peut être appliqué qu'à un boolean (3.37) : erreur ayant lieu si on essaie de faire un not sur autre chose qu'un booleen.

nom.deca :ligne :colonne : On ne peut pas avoir un void en parametre (2.9) : erreur ayant lieu si on essaie de mettre un void en paramètre d'une méthode.

nom.deca :ligne :colonne : On définit deux fois le parametre : erreur ayant lieu si on définit deux fois un paramètre.

nom.deca :ligne :colonne : On ne peut pas retourner un void : erreur ayant lieu si on essaie de renvoyer un void.

nom.deca :ligne :colonne : This ne peut pas etre appelé dans le main (3.43) : erreur ayant lieu si on essaie d'utiliser this dans le main.

nom.deca :ligne :colonne : L'UnaryMinus nécessite un opérande de type int ou float (3.37) ! : erreur ayant lieu si on essaie d'utiliser un '-' devant autre chose qu'un float ou un int.

3.4 Erreur d'exécution de l'assembleur

Les erreurs dans cette catégorie ont lieu lorsque l'utilisateur utilise la commande **ima fichier.ass**. Les erreurs n'ont pas de numéro de ligne ou de colonne.

Division par 0 interdite : erreur ayant lieu si l'utilisateur tente d'effectuer une division par 0.

Error : Input/Output error : erreur pouvant avoir lieu si lorsqu'une demande est effectué sur le terminal de l'utilisateur. L' erreur arrive si l'utilisateur ne rentre pas une variable demandé. Exemple un float à la place d'un int, ou un String à la place d'un float ou d'un int.

Error : Debordement flottant nombre trop grand : erreur pouvant avoir lieu si le nombre assigne à une variable est trop grand et ne peut pas être représenté.

4 Extension : TAB

Nous avons étendu le langage Deca avec un nouveau type : les tableaux. Notre compilateur permet aux utilisateurs de créer des tableaux et des matrices. (cf Documentation sur l'extension)

La Bibliotheque standard contient des méthodes de calcul. Ces méthodes sont contenues dans la librairie **MathArray.decah** qu'il faut inclure en début de fichier si vous voulez l'utiliser.

Exemple :

```
#include "MathArray.decah"

{
    MathArray Calc = new MathArray();
    float[] A = {1.0, 2.3, 5.3};
    float[] B = {1.0, 2.8, 5.0};
    float[] C = Calc.addFloatTable(A, B);
    println(C[0], " ", C[1], " ", C[2]);
}
```

4.1 Limitations

Les méthodes de la bibliothèque sont basiques : opposé (tableau et matrice), transposé d'une matrice, addition (tableau et matrice) et multiplication (matrice). Ces méthodes peuvent être utilisées avec des éléments de type int ou float.

Un des objectis étaient aussi de pouvoir créer des tableaux d'éléments de classe créé. Les étapes A et B donnent un arbre contextuelle cohérent. La gestion de la mémoire ne permet pas de créer ces tableaux à l'heure actuelle.

4.2 Messages d'erreur

Comme cette extension consiste à étendre le langage, des erreurs contextuelles ont été ajouté à la liste précédente (paragraphe 3). Certaines erreurs sont réutilisés pour les tableaux. La liste exhaustive des erreurs sera détaillé dans la documentation de l'extension. Mais par exemple, on ne peut pas déclarer un tableau d'entiers et mettre des flottants à l'interieur :

```
{
    int[] tableau = {1.5, 5.02, 2.5};
}
```