



ENSIMAG GRENOBLE-INP

---

# Compilateur DECA

## Manuel utilisateur

---

*Auteurs :*

Oussama LAMZAOURI  
Mohammed hadi CHAMSI  
Ziad RAZANI  
Rong LI  
Sofia ECHARIF

*Tuteur :*

Akram IDANI

24 janvier 2022



# Table des matières

|          |   |          |
|----------|---|----------|
| <b>1</b> | <b>Introduction</b>   | <b>1</b> |
| <b>2</b> | <b>Fonctionnement général</b>                                   | <b>2</b> |
| 2.1      | Comment lancer la compilation d'un fichier <code>.deca</code> ? | 2        |
| 2.2      | Que faire après compilation?                                    | 2        |
| <b>3</b> | <b>Limitations du compilateur</b>                               | <b>4</b> |
| <b>4</b> | <b>Les messages d'erreurs</b>                                   | <b>5</b> |
| 4.1      | Erreurs Lexico-syntaxiques                                      | 5        |
| 4.1.1    | Erreurs lexicales   | 5        |
| 4.1.2    | Erreurs syntaxiques   | 5        |
| 4.2      | Erreurs de contexte   | 5        |
| 4.2.1    | Par rapport aux type  | 5        |
| 4.2.2    | Par rapport aux classes   | 6        |
| 4.2.3    | Par rapport aux champs  | 6        |
| 4.2.4    | Par rapport aux méthodes  | 7        |
| 4.3      | Erreurs de la génération du code assembleur ou son exécution    | 8        |
| <b>5</b> | <b>Limitation en rapport avec l'extension TRIG</b>              | <b>9</b> |



# Chapitre 1

## Introduction

Ce document vous permettra de prendre en main et maîtriser le fonctionnement du compilateur Deca entre vos mains, il vous indiquera de plus les différentes options de compilation proposées ainsi que leurs valeurs de retour attendues. Il est alors essentiel de taper les commandes mentionnées à la lettre et respecter les formats de commandes.

Le compilateur entre vos mains est un compilateur pour le langage informatique DECA. Il est capable de lire votre fichier source tapé en DECA et exécuter vos instructions.

Simplement, commencer par écrire votre fichier source `.deca` dans un répertoire. Puis suivez le guide développé dans la page suivante.

Vous trouverez dans le chapitre dédié une liste des messages d'erreurs qui peuvent être affichés lors d'une compilation, accompagnée des différentes configurations causant leur déclenchement.

De plus, une bibliothèque mathématique `math.decah` vous est proposée pour les calculs incluant les fonctions trigonométriques précisées dans le chapitre dédié.

D'autre part, une liste de limitations techniques de ce compilateur vous est proposée avec quelques justifications.

## Chapitre 2

# Fonctionnement général

### 2.1 Comment lancer la compilation d'un fichier .deca ?

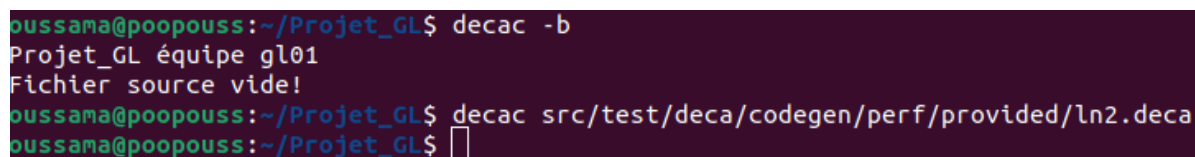
```
decac [[-p | -v] [-n] [-r X] [-d]* [-P] <chemin vers fichier deca>...] | [-b]
```

1. La commande `decac`, sans argument, affichera les options disponibles.
2. On peut appeler la commande `decac` avec un ou plusieurs fichiers sources Deca.

Voici la liste des options :

- `-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 (c.-à-d. s'il n'y a qu'un fichier source à compiler, la sortie doit être un programme Deca syntaxiquement correct)
- `-v` (**vérification**) : 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` (**registres**) : limite les registres banalisés disponibles à `R0 ... RX-1`, avec  $4 \leq X \leq 16$
- `-d` (**debug**) : active les traces de debug. Répéter l'option plusieurs fois pour avoir plus de traces.
- `-P` (**parallel**) : s'il y a plusieurs fichiers sources, lance la compilation des fichiers en parallèle (pour accélérer la compilation)

N.B. Les options '`-p`' et '`-v`' sont incompatibles, ils ne pourront pas alors être lancés dans une même commande `decac`.



```
bussama@poopouss:~/Projet_GL$ decac -b
Projet_GL équipe gl01
Fichier source vide!
bussama@poopouss:~/Projet_GL$ decac src/test/deca/codegen/perf/provided/ln2.deca
bussama@poopouss:~/Projet_GL$
```

FIGURE 2.1 – Exemple d'une commande de compilation pour le fichier source `ln2.deca`.

### 2.2 Que faire après compilation ?

Une fois la compilation terminée, un fichier d'extension `.ass` et du même nom que votre fichier source `.deca` sera généré dans le même répertoire que ce dernier.

L'appel de l'interpréteur s'effectue en exécutant la commande `ima [options] nom_de_fichier_assembleur`

Sans option, le programme est directement assemblé et exécuté.

Options (les trois premières sont exclusives) :

- -d : appel du metteur au point.
- -s : exécution directe sans entrer dans le metteur au point, avec à la fin, si aucune erreur ne s'est produite, affichage du nombre d'instructions du programme et du temps d'exécution.
- -r : exécution directe sans entrer dans le metteur au point, avec passage à la ligne à chaque écriture d'entier, de flottant ou de chaîne.
- -p nnn : (nnn : entier  $\geq 0$ ) exécution avec une pile de nnn mots (par défaut : 10\_000).
- -t nnn : (nnn : entier  $\geq 0$ ) exécution avec un tas de nnn mots (par défaut : 10\_000).

N.B. Il faudra donc penser à installer ima correctement sur votre ordinateur (à compléter par CHAMSI).

Pour notre commande ci-dessus, un fichier ln2.ass est généré dans le répertoire correspondant.

```
boussama@poopouss:~/Projet_GL$ decac src/test/deca/codegen/perf/provided/ln2.deca
boussama@poopouss:~/Projet_GL$ cd src/test/deca/codegen/perf/provided/
boussama@poopouss:~/Projet_GL/src/test/deca/codegen/perf/provided$ ls
ln2.ass ln2.deca ln2_fct.deca syracuse42.deca
boussama@poopouss:~/Projet_GL/src/test/deca/codegen/perf/provided$
```

FIGURE 2.2 – Le fichier .ass pour l'exemple précédent.

Il faudra après d'exécuter le fichier .ass, pour cela il faut lancer la commande suivante en appelant l'interpréteur-metteur au point (débugueur) ima.

```
boussama@poopouss:~/Projet_GL$ ima src/test/deca/codegen/perf/provided/ln2.ass
6.93148e-01 = 0x1.62e448p-1
boussama@poopouss:~/Projet_GL$
```

FIGURE 2.3 – Exécution du fichier ln2.ass par ima.

## Chapitre 3

# Limitations du compilateur

La gestion des casts d'objet et `instanceof` n'est pas possible car nous n'avons pas fini d'implémenter leur génération de code. Le compilateur Deca génère des fichiers assembleur `.ass` qui ont problèmes sur l'adresse mémoire. En revanche, l'analyse syntaxique et contextuelle pour les deux méthodes ont été définies, donc l'utilisateur peut quand même vérifier l'arbre décoré du programme en utilisant la commande `test_context <filename>` et l'arbre de syntaxe abstrait en appliquant la commande `test_synt <filename>`. Aussi, à cause d'une implémentation qui ne correspond pas strictement aux règles communes aux trois passes de vérifications contextuelles, la génération du code va rencontrer quelques problèmes. Par exemple, la comparaison avec `null` n'est pas implémentée dans la vérification du assignement et comparaisons booléens, donc `{Object a = null;}` nous donne le message d'erreur "the following types are not compatible for assignement : Object and null", une comparaison `{a == null;}` nous donne le message d'erreur `invalid type for comparaison operation`.



# Chapitre 4

## Les messages d'erreurs

Les erreurs générées par le compilateur sont de la forme suivante :

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

### 4.1 Erreurs Lexico-syntaxiques

#### 4.1.1 Erreurs lexicales

Ces types des erreurs sont levées par l'analyse lexicale du programme, qui utilise des tokens non reconnus par le compilateur.

- `The token is not recognized`  
un mauvais token est généré.
- `<filename> : include file not found`  
Le nom de fichier en argument `#include` n'existe pas.

#### 4.1.2 Erreurs syntaxiques

Les erreurs syntaxiques sont principalement générés par ANTLR. Elles nous indiquent la position des erreurs de syntaxes et les types de token ou les caractères qui sont attendus. En plus, nous aurons une exception "the number is too large" pour les entiers ou flottants trop grands.

- `mismatched input ' ' expecting`  
Le symbole ne correspond pas à ce qui était attendu.
- `Syntax error: the number is too large or invalid`  
Quand l'utilisateur applique un entier ou un flottant trop grand.
- `left-hand side of assignment is not an lvalue`  
Lors d'une affectation, Si la partie gauche ne correspond pas au token identificateur (IDENT). Par exemple, `0 = x;`
- `missing ' ' at ' '`  
L'oubli d'un symbole, par exemple, d'une accolade fermante à la fin du `main` génère le message *"missing CBRACE at '<EOF>'"*
- `no viable alternative at input ' '`  
Le symbole n'a pas d'alternative viable.
- `extraneous input ' ' expecting ' '`  
Par exemple, lors d'une déclaration, la visibilité est écrite deux fois, `protected protected x;`
- `Circular include for file <filename>`  
L'inclusion circulaire

### 4.2 Erreurs de contexte

Les erreurs contextuelles permettent de rejeter les programmes mal décorés et d'afficher un message d'erreur.

#### 4.2.1 Par rapport aux type

- `the following types are not compatible for assignement : type1 and type2` : erreur soulevée lors d'une affectation quand les deux types sont incompatibles.

- `invalid conversion from type1 to type2` : soulevée lors d'une invalide conversion de type vers un autre type, souvent quand une condition ne retourne pas un type boolean.
  - `invalid type for arithmetique operation type` : soulevée si au moins un des deux opérandes utilisés pour une opération arithmétique n'est ni un *int* ni un *float*.
  - `invalid type for arithmetique operation type1 and type2` : erreur soulevée si les deux opérandes concernés ne sont ni de type *int* ni *float*.
  - `invalid type for comparaison operation type` : soulevée si au moins un des deux opérandes utilisés pour une opération comparaison n'est ni un *int* ni un *float*.
  - `invalid type for comparaison operation type1 and type2` : erreur soulevée si les deux opérandes concernés ne sont ni de type *int* ni *float*.
  - `bad operand types for binary boolean operator operand` : quand l'opérande utilisé dans une opération booléen n'est pas type *boolean*.
  - `can't apply 'minus' to a <type>` : L'utilisation d'un signe "-" sur des opérandes non entiers ou flottants.
  - `can't apply 'not' to a <type>` : L'utilisation d'un signe "!" sur des opérandes non booléens.
- Next level..
- `Declaration does not declare anything.` : Quand on déclare comme `{ int; }`.
  - `<identifiant> was not declared in this scope`. Identificateur (variable, classe ou champs) non déclaré

## 4.2.2 Par rapport aux classes

- Super Class `<class name>` should be declared first : (règle 1.3) La super classe est indéfinie

```
1 class B extends A {
2 }
```

- `extends` can be applied only to classes

```
1 class B extends boolean{
2 }
```

- `<class name>` already declared : Double définition de la classe
- Contextual error : The identifier should be a class : (règle 3.42) Le type spécifié n'est pas un type de class

```
1 {
2     int a = new int();
3 }
```

- Contextual error : The left value should be a class type. : erreur soulevée quand l'opérande droite est de type classe mais l'opérande gauche n'est pas une classe.
- Contextual error : The two classes are not compatible (regle 3.28) : incompatibilité à cause du lien de parenté entre les deux classes.

## 4.2.3 Par rapport aux champs

- `type of attribute cannot be void` (regle 2.5) : la déclaration d'un champ ne doit pas être de type void

```
1 class A {
2     void x;
3 }
```

- `<champ>` must be have a declaration as a field (regle 3.66) : l'attribut n'est pas déclaré comme un champ

```
1 class A {
2 }
3
4 {
5     A a = new A();
6     a.x; // x must be have a declaration as a field
7 }
```

- `<champ>` already declared : Double définition d'un champ
- Contextual error : A protected field can't be accessed outside class (regle 2.5) : l'accès à un champ protégé hors classe

```

1  class A {
2      protected int x;
3  }
4
5  {
6      A a = new A();
7      println(a.x); // access a protected field in Main
8  }

```

- Contextual error : A protected field can only be accessed from within the class it was declared in or from within one of it's subclasses (regle 3.66) : le type de la classe doit être un sous-type de la classe où le champ protégé est déclaré

```

1  class A {
2      protected int x;
3  }
4
5  class B {
6      void add(A a) {
7          a.x = 1; // access a protected field in class A which is not his parent class
8      }
9  }

```

#### 4.2.4 Par rapport aux méthodes

- Contextual error : The object part should be as type ClassType : quand on fait une selection ou methodCall come objet.champ, la partie objet doit être de type class.

```

1  {
2      int y;
3      int x = y.a;
4  }

```

- Contextual error : The number of arguments for the method is not correct (regle 3.71) : Le nombre d'argument donné n'est pas bon.
- Contextual error : the type of parameter dont correspond to the signature of method (regle 3.71) : lors d'un appel de la méthode, on utilise un mauvais paramètre.

```

1  class A {
2      protected int x;
3  }
4  class B extends A {
5      A get(A a) {
6          return a;
7      }
8  }

```

- Contextual error : <method> must have the same signature as in superclass : lors d'une re-définition de la méthode, on utilise une mauvaise signature.

```

1  class A{
2      int test(int x){}
3  }
4  class B extends A{
5      int test(float x){} // Redefinition de la methode avec mauvaise signature
6  }

```

- Contextual error : <method> already declared. : Double définition d'une méthode
- Contextual error : A type to cast can't be void (regle 3.39) : Le type à convertir n'est pas de type void.
- Contextual error : The two types are not compatible to cast.

```

1  {
2      boolean t = (int)(1); // rule 3.39
3  }

```

- Contextual error : 'instanceof' expected class or null for instance but got <type> (regle 3.40) : pour l'opérande gauche de instanceof on doit avoir class ou null.
- Contextual error : 'instanceof' expected class but got <type> (regle 3.40) : pour l'opérande droite de instanceof on doit avoir class.

```

1  class A{}
2  {A a;
3  boolean x = a instanceof int;}

```

- Contextual error: 'this' can only be applied in class (règle 3.43) : Utilisation de 'this' en dehors d'une classe
- Contextual error: Can't return in a function void (regle 3.24) : On ne peut pas retourner dans une fonction void

```

1  class A {
2      void test() {
3          return 1; // regle 3.24
4      }
5  }

```

- Contextual error: The type of 'return' in current class doesn't match the type demanded

```

1  class A {
2      boolean isReturn() {
3          return "wrong return";
4      }
5  }

```

- Contextual error : Return value is expected in this method (regle 3.24) : On quitte une fonction non void sans return

## 4.3 Erreurs de la génération du code assembleur ou son exécution

Le compilateur Deca est possible de générer des fichiers assembleur .ass incorrects, car le comportement est indéfini à l'exécution. Si on active pas l'option -n (no check), on peut avoir ces erreurs.

- Error: Use zero as division  
L'utilisateur n'a pas le droit de faire une division par 0.
- Error : Stack Overflow  
Si le calcul demandé à la machine est tellement chargé qu'il provoque un débordement de pile, ce type d'erreurs pourrait
- Error : Heap Overflow  
Si l'allocation d'un nouvel objet échoue, cette erreur sera renvoyée à l'utilisateur.
- Error : Object is null  
Si l'on veut accéder aux attributs ou aux méthodes d'un objet n'ayant pas été instancié correctement, (=null), cette erreur apparaîtra à l'écran.
- WINT/WFLOAT avec R1 indéfini  
Lors d'utilisation de variables non initialisés. Par exemple, `int x; print(x);`

## Chapitre 5

# Limitation en rapport avec l'extension TRIG

Au vu du manque de tests effectués sur le fichier "Math.decah" (Contrainte de temps causée par une phase de debuggage du compilateur, ce qui ne permettait pas d'exécuter les tests de la librairie "Math.decah"), il est difficile de déterminer si l'implémentation obéit aux normes de précision et performance, mais il est important de noter que les fonctionnalités suivantes n'ont pas été implémentées ce qui garantirait à priori une meilleure précision du côté informatique et la performance :

- Instruction FMA : Cette instruction peut être utilisée dans les calculs des séries puisqu'elle n'effectue qu'un arrondi à la place de deux, ceci réduira drastiquement l'écart entre la valeur théorique et la valeur obtenue.
- Cast float en int : Il est à noter que la fonction floor implémentée dans la librairie utilise une boucle (Complexité :  $\lfloor f \rfloor$  opérations contre 3 opérations pour l'implémentation triviale) car le cast de float en int ne marche pas très bien.
- La méthode de Horner : Non implémentée, donc la réduction devrait être moins précise.

D'autre part, l'algorithme utilisé a été étudié et l'erreur mathématique pour la plupart des fonctions est prouvée inférieure à  $10^{-8}$ . Une correction des bugs et implémentation des méthodes précisées plus haut devrait donner en conséquence une librairie rapide et fiable.