# Compilateur Deca : Manuel Utilisateur

Équipe 58

January 23, 2017

## 1 Introduction au compilateur

Le compilateur Deca prend en argument un fichier source en Deca et le compile pour donner un fichier en code assembleur.
Ce compilateur gre toutes les tapes de la compilation, c'est-a-dire le parsing (analyse lexical et syntaxique) du fichier source, l'analyse contextuelle du code et la gnration du code assembleur. Plusieurs options sont disponibles pour le compilateur

## 2 Exécution de decac

Pour lancer le compilateur, il faut utiliser la commande `decac`. Cette commande possde diffrentes options d'excution:

- `decac -b`
  Affiche la bannière de l'équipe qui a programmé ce compilateur.
- `decac -p <ficher deca>`
  Parse le fichier passé en argument et affiche sa dcompilation.
- `decac -v <fichier deca>`
  Vérifie contextuellement le fichier deca. Affiche une erreur si il y en a.
- `decac -n <fichier deca>`
  Excute le fichier deca en ignorant les erreurs de dbordement à l'excution.
- `decac -r X <fichier deca>`
  Execute le fichier deca en limitant les registres disponibles à `R{0}...R{X}`. `X doit tre compris entre 4 et 16.`
- `decac -d <fichier deca>`
  Augmente le niveau de debug lors de l'excution. Rpter l'option jusqu' 4 fois pour augmenter les traces de debug.
- `decac -P <fichier(s) deca>`
  Si il y a plusieurs fichiers sources, les compile en parallle.

Plusieurs options peuvent tre appeles simultanment, `-p` et `-v` sont cependant incompatibles. Le fichier `.ass` obtenu en sortie du compilateur xecut avec la commande `ima <fichier assembleur>`.

# 3 Erreurs leves par le compilateur

## 3.1 Erreurs lexicales

Voici les erreurs qui peuvent tre leves par l'analyse lexicale:

- `token recognition error`
  Est leve si le compilateur ne reconnat pas le lexme.

## 3.2 Erreurs syntaxiques

- `RecognitionException / FailedPredicateException`
  Echec de l'analyse syntaxique. Est leve si le parser a rduit les possibilits une seule, et que le code fourni ne convient pas. Le prochain symbole attendu sera fourni.

- `NoViableAltException`
  Echec de l'analyse syntaxique. Est leve s'il y avait plusieurs possibilits l'endroit de l'chec, mais qu'aucune ne convenait au code. Les prochaines symboles possibles seront fournies.

- `InvalidLValue`
  Echec de l'analyse syntaxique. Est leve si ce dernier repre une tentative de dfinir une forme qui n'a pas l'tre. Exemple : (1+1) = 3; causera cette erreur, car (1+1) n'est pas une variable ou autre forme qu'il est sens d'assigner une valeur.

## 3.3 Erreurs contextuelles

Voici les erreurs qui peuvent tre leves par l'analyse contextuelle :

- `type or class undefined`
  Est leve si un identifier de type est attendu, mais l'identifier trouv n'est pas un type prdfini ou une classe qui a t dfinie.

- `class extends type`
  Est leve si, lors de la dclaration d'une classe, l'identifier suivant `extends` est un type (`int,float,boolean,void`);

- `class already defined`
  Est leve si une mme classe est dclare plusieurs fois.

- `field already defined`
  Est leve si un champ est dclar plusieurs fois ou dclar avec le mme nom qu'une mthode existante.

- `field cannot be void`
  Est leve si un champ a pour type `void`.

- `method already defined`
  Est leve si une mthode est dclare plusieurs fois dans la mme classe ou dclare avec le mme nom qu'un champ existant.

- `method overrides method with different signature`
  Est leve si une mthode essaie d'craser une mthode avec une diffrente signature (diffrent type, diffrent nombre de paramtres ou différent types de paramtres).

- `parameter cannot be void`
  Est leve si un paramtre d'une mthode a pour type `void`.

- `parameter already defined`
  Est leve si plusieurs paramtres d'une mme mthode ont le mme nom.

- `variable already defined`
  Est leve si une variable est dclare plusieurs fois.

- `variable cannot be void`
  Est leve si une variable a pour type `void`.

- `return called in void method`
  Est leve si `return` est appele dans une mthode qui a pour type `void`.

- `type of expression must match method type`
  Est leve si l'expression renvoye n'est pas du mme type que la mthode.

- `expected type found class`
  Est leve si l'expression est une instance de classe alors qu'une expression de type `int`,`float` ou `boolean` est attendue.

- `incompatible class type`
  Est leve si la classe attendue n'est pas une super-classe de la classe renvoy. Ce cas est le seul qui permet d'initier l'instance d'une classe avec une diffrente classe.

- `expected different type`
  Est leve si l'expression trouve n'est pas du type attendu, sauf si l'expression est de type `int` et le type attendu est `float`. Dans ce dernier cas, on converti l'`int` en `float`.

- `only string, int and float expressions can be printed`
  Est leve si un `void`,`null` ou une instance de classe est mis en paramtre de `print` ou `println`.

- `condition must be boolean`
  Est leve si la condition dans un `if` ou `while` n'est pas boolen.

- `operands must be int or float`
  Est leve si les oprandes d'un opration arithmtique (ou d'une comparaison) ne sont pas numriques.

- `operand must be int or float`
  Est leve si l'oprande du moins unaire n'est pas numrique.

- `operands must be boolean`
  Est leve si les oprandes d'une opration boolenne n'est pas `boolean`.

- `operand must be boolean`
  Est leve si l'oprande de `Not(!)` n'est pas boolen.

- `operands must have same type`
  Est leve si les oprandes d'une comparaison n'ont pas le mme type. Notamment, on ne peut pas comparer un `int` et un `float`.

- `incompatible types for cast`
  Est leve si expression est `Cast` dans un type incompatible a son propre type. Le seul `Cast` de type accept est le passage de `int` à `float`.

- `type cast to class`
  Est leve si une expression de type `int,float` ou `boolean` est `Cast` dans une `class`.

- `class cast to type`
  Est leve si l'instance d'une classe est `Cast` en `int,float` ou `boolean`.

- `classes incompatible for cast`
  Est leve si l'instance d'une classe est `Cast` dans une classe incompatible. Cela se produit quand les deux classes ne sont pas dans la mme hirarchie de classes.

- `identifier is not a class`
  Est leve si l'identifier suivant `new` n'est pas un nom de classe.

- `cannot call this in main`
  Est leve si `this` est appel en dehors d'une dclaration de classe.

- `left operand not instance of a class`
  Est leve si l'identifier a gauche d'un appel de champ n'est pas l'instance d'une classe.

- `no such field in class`
  Est leve si l'identifier à droite d'un appel de champ ne correspond pas à un champ de la classe à gauche.

- `identifier is not a field`
  Est leve si l'identifier à droite d'un appel de champ correspond à une mthode.

- `field is protected`
  Est leve si on essaie d'appeler un champ `protected` en dehors de la classe à laquelle il appartient.

- `expression not instance of a class`
  Est leve si l'identifier  gauche d'un appel de mthode n'est pas l'instance d'une classe.

- `direct method call in main`
  Est leve si une mthode est appele sans prciser la classe ou l'instance de classe, en dehors d'une dclaration de classe.

- `identifier is not a method`
  Est leve si l'identifier a droite d'un appel de mthode correspond a un champ.

- `number of parameters does not match signature`
  Est leve si une mthode est appele avec un nombre de paramtres diffrent du nombre de paramtre dans sa signature.

- `parameter type does not match signature`
  Est leve si le type d'un paramtre dans un appel de mthode ne correspond pas au type de paramtre dans la signature de la mthode.

- `class type in call not subclass of class type in signature`
  Est leve si, lorsqu'une mthode demande une instance de classe en paramtre, la classe appele n'est pas une sous-classe de la classe donne en signature.

## 3.4 Erreurs à l'éxecution

- `Erreur : pile pleine"`
  Est leve si il est impossible pour le programme de reserver la pile dont il a besoin avec l'operation assembleu `TSTO`

- `Error: Input/Output error`
  Est leve si :

  - la valeur reue pour un entier sur l'entre utilisateur standard est suprieur $2^{31} - 1$ ou infrieur $-2^{31}$

  - la valeur reue pour un flottant sur l'entre utilisateur standard est suprieur `1.175494351 E 38` ou infrieur `3.402823466 E + 38` en valeur absolue

  - la valeur reue n'est pas conforme au type attendu lors de la lecture

  - affichage d'une variable du mauvais type. N'est pas cens tre soulev.

- `Error: Arithmetic Overflow`
  L'erreur est soulev lors de l'overflow de variables flottantes. Il est possible que cela provienne de:

  - Multiplication de deux flottant dpassant la taille limite.

  - Division d'un flottant par un flottant trs petit provocant le dpassement de taille du rsulat.

- `Error: Heap OverFlow`
  Est leve lorsque il y a dpassement de tas. C'est dire que le programme a essay de d'allouer plus de mmoire qu'il ne lui est possible. Normalement avec la machine ima cette valeur pour une allocation de 10002 plage 4 octets

# 4 Choix d'implmentation et erreur la compilation

## 4.1 Choix d'implmentation

L'implmentation est assez proche du squelette fourni. La diffrene majoritaire est que la mthode `codeGen` retourne une variable `DVal` qui nous permet d'optimiser l'utilisation des registres. Nous avons aussi pris le partie de gnraliser l'appelle certaine classe pour la gnration du code. Ainsi le package `codeGen` offre des classes permetants une gnration de code gnral. Tel que `codeGenBinaryInstructionDValtoReg`.

## 4.2 Comportement inattendu

Lors d'appel de plusieurs intialisations ( c'est dire qu'une initialisation apelle l'initialisation d'une autres classe.) nous avons remarqu que le code gnr sauvegarde plus de registre que ceux utilis lors de l'apelle de la mthode.
Il existe aussi un problme si l'apelle d'une mthode besoin d'accder la pile. En effet la gestion de l'overflow n'est pas adapt cet environement. Pour le corrig il nous aurait suffit de modifier rapidement le code de `DecacCompiler` pour ajouter un compteur de sauvgarde.

## 4.3 Oublies

Nous avons oubli de cod les dclaration de variables dans les mthodes

# 5 Extensions

Cette partie a pour but de prsenter succintement le fonctionnement des extensions implmentes dans le compilateur Deca. Elles sont au nombre de deux : DeadStore et ConstantFolding

### 5.0.1 Deadstore

Pour activer le deadstore sur un programme, il suffit de rajouter l'option -o1 pendant la compilation du fichier Decac, en tapant en ligne de commande decac -o1 fichier.deca. Le fichier assembleur obtenu sera ainsi optimis avec les spcificits du deadstore. Le Deadstore permet de supprimer les variables initialises mais non utilises par la suite dans votre programme. Par exemple, si vous crivez :

```
int a=1;
int b=2;
  int c;
c=5*b+1;
```

Le compilateur aura dtruit de faon interne la variable "a" du programme, car elle n'est pas utilise ni dans la liste des instructions ni aprs son initialisation.

### 5.0.2 ConstantFolding

Pour activer le ConstantFolding sur un programme, il suffit de rajouter l'option -o2 pendant la compilation du fichier decac, en faisant par exemple decac -o2

fichier.deca. Le fichier assembleur obtenu sera ainsi optimis avec les spcificits de ConstantFolding. Le ConstantFolding permet de remplacer le calcul de constantes par le rsultat directement. Par exemple si vous crivez :

```
int a=5;
int b=10;
int c=15;
int d=5+5+5+5+5;
```

Le compilateur effectuera en interne le calcul de 5+5+5+5+5, et le remplacera par son rsultat, ici 25. A la fin, le fichier assembleur sera adapt pour ce programme deca :

```
int a=5;
int b=10;
int c=15;
int d=25;
```

Le calcul est rendu plus rapide pour le compilateur, et pourra acclrer grandement les longs calculs de constantes (attention, le calcul avec des variables n'est pas pris en compte)

### 5.0.3 limitations

Pour le Deadstore, l'optimisation ne prend pas en charge le cas o il y a des calculs lors des initialisations. pour que le deadstore entre en action, la variable ne doit pas tre utilise au-del des initialisations, au niveau des instructions
Pour le ConstantFolding, le code n'est pas optimis, seul le calcul avec des constantes l'est.