# Le manuel utilisateur

GUILLAUME Laure
LUU Duc Anh
EL RHATRIF Mohammed Amine
CHAKIR Sami
FARID Othmane

23 janvier 2017

## 1 Limitations et particularités de notre implémentation

### 1.1 Présentation du compilateur

Ce compilateur est conçu pour compiler des fichiers écrits en langage Deca qui est un sous-langage Java. Pour compiler un programme, il vous suffit d'écrire la commande :
*decac -option*
Vous trouverez aussi une base de tests associée avec des scripts permettant de compiler tous les tests et d'en vérifier le résultat. En effet, lors de l'exécution de ces scripts le code assembleur et le resultat sont sauvegardés respectivement dans des fichier .res et .dec .
Le compilateur est aussi capable de décompiler des fichiers, produisant alors des fichiers en langage deca correctes.

### 1.2 Particularité de notre compilateur

Tout d'abord notre compilateur permet l'analyse lexicale de fichiers transmis par l'utilisateur. Il les transforme en des entités lexicales selon les conventions fixées dans le cahier des charges. Ces entités lexicales sont par la suite mises en entrée de l'analyseur syntaxique. Si le fichier contient des lexèmes invalides, le compilateur rapportera une erreur (cf 2. Messages d'erreur). Le compilateur permet ainsi l'analyse syntaxique selon la syntaxe concrète de Deca définie dans le cahier des charges et en fait aussi une verfication contextuelle. Il effectue par la suite des décorations enrichies mentionnées dans le cahier des charges. Apres avoir fait les les étapes précédentes ,le compilateur génére le code assembleur correspondant au fichier utilisateur. Ce code assembleur sera transmis en entrée à l'assembleur ima , qui l'exécutera.
Nous avons pu mettre en place comme convenu les otions suivantes :

- -b qui permet de connaître ne nom de l'équipe ;
- -p arrête decac après l'étape de construction de l'arbre ;
- -d arrête decac après l'étape de vérifications.

Nous avons aussi choisi de faire une extension mathématique, ainsi notre compilateur est capable de compiler des fichiers contenant des fonctions mathématiques.(cf 3. Extension math).
Notre assembleur compile de manière correcte, en fournissant des résultats satisfaisants les programmes deca ne comportant de programmation orientée objet. Cependant l'implémentation

de la partie objet possède quelques lacunes et ne peu par conséquent qu'exécuter des programmes simples. Ainsi notre compilateur est très limité en programmation orientée objets. Nous parlerons de ces limitations dans la partie suivante.

### 1.3 Limitations et conséquences pour l'utilisateur

L'implémentation de la partie objet n'étant pas complète il reste quelque points ou notre compilateur est incapable de compiler le ficher utilisateur. Ainsi les classes ne doivent pas avoir des champs avec des noms qui ont déjà été utilisées par ses classes mères comme champs. De plus dans les méthodes, pour invoquer un champs il est nécessaire de passer par la selection. En effet, pour le bon fonctionnement du compilateur pour faire $x = x + 1$ dans une méthode, l'utilisateur devra faire : $this.x = this.x + 1$ si x n'est pas un paramètre de la méthode.
Nous n'avons pas non plus eu le temps d'implémenter les opérateurs instanceof et le casting que cela soit au niveau de l'analyse contextuelle ou dans la génération du code.
Nous n'avons par ailleurs pas eu le temps d'implémenter toutes les options. Les options suivantes ne sont donc pas supportée par le compilateur :

- -n la suppression des tests de débordement à l'exécution ;
- -r X limitation des registres banalisés disponibles ;
- -d Activation des traces de debug ;
- -P la compilation en parallèle.

## 2 Messages d'erreur

Lorsque notre compilateur n'est pas capable de compiler le fichier utilisateur celui-ci rend une erreur. Cette erreur permettera à l'utilisateur de comprendre les causes de cet échec. Il y a 2 grands types d'échec : ceux du a la syntaxes et ceux dus à l'exécution du code assembleur. L'erreur peut aussi être lexicale et elle sera alors relevée par ANTRL4 par un message contenant *extraneous input*.
Vous trouverez ci-dessous des tableaux contenant les erreurs avec leurs causes et le messages associé. Elles sont classées par règles et par type d'erreur.

## 2.1 Erreur de syntaxe contextuelles

| Règles | Description | Messages d'erreurs |
|---|---|---|
| Passe 1 | ... | ... |
| 1.3 (decl_class->...) | class_super de la classe reconnue n'est pas dans Env_type Super n'est pas associée à une définition de classe. | erreur Contextuelle :la classe mère est non encore déclarée. |
| 1.3 (decl_class-> ...) | deux classes ont le même nom. | erreur Contextuelle :cette class a déjà été déclarée. |
| 2.5 (decl_field-> ...) | un champs de la classe qui existe dans les super classes. | la variable est déjà un champ ou une méthode dans une super classe. |
| 2.5 (decl_field-> ...) | le champs est de type void. | le type d'un champ ne devrait pas être un void. |
| Passe 2 | ... | ... |
| 2.3 (decl_class-> ...) Empilement | env_exp_Field et env_exp_Methode ne sont pas distincts. | il existe un champ/methode du même nom. |
| 2.7 (decl_method-> ...) | override d'une méthode mais les champs différents. | this method does not have the same signature as the method in the superclass |
| 2.7 (decl_method-> ...) | override d'une méthode mais le type de retour diffère | this method is already existing in a superClass with a different type |
| 2.7 (decl_method-> ...) | le nom de la méthode est utilisé comme champ dans une classe mère. | name of the method is used as a field in a superClass |
| 2.7 (decl_method -> ...) | le nom de la méthode est utilisée comme champ ou une méthode dans la classe courante. | this name is already used for a field or method in the current class |
| 2.7 (decl_method -> ...) | le type de retour de la méthode n'est pas un sous type de la méthode qui est override. | this method type is not a sub type of the method in the super Class |
| 2.9 (decl_parm -> ...) | type du paramètre est void | type shouldn't be void in declaration. |
| 2.9 (decl_parm -> ...) | deux ou plus paramètres ont le même nom. | this parametres is already declared |

| Passe 3 | ... | ... |
|---|---|---|
| 3.7 (decl_field-> ...) | type de l'initialisation est incompatible avec le type du champ (boolean x = 1). | type de l'initialisation est incompatible avec le type du champ. |
| 3.17 (decl_var-> ...) | type de l'initialisation est incompatible avec le type de la variable déclarée | type shouldn't be void in declaration |
| 3.17 (decl_var-> ...) | une variable ne peut être déclarée qu'une seule fois. | this variable is already declared |
| 3.18 (bloc -> ...) | toute déclaration se fait avant la liste des instructions | Erreur syntaxique |
| 3.24 (inst -> return ) | le type de retour doit être différent de void. | le valeur de retour est de type null |
| 3.28 (rvalue -> ...) | La valeur non terminale ne correspond pas aux sous-ensembles des expressions compatibles avec le type d'entrée | type not expected |
| 3.29 (condition -> ...) | la condition n'est pas une expression de type booléen | Contextual error : condition is not boolean |
| 3.31 (exp_print->...) | l'expression à afficher doit être de type int, float ou string. | invalid argument for print |
| 3.32 (exp -> assign. . . ) | le type de la variable n'est pas compatible avec le type du retour de la fonction read | variable not compatible to instruction read |
| 3.33 (esp -> op_bin . . . ) | la comparaison doit être entre deux expressions de même type, sauf les int et les float. | cannot use operator "+ this.getOperatorName() +" on two different type |
| 3.33 (esp -> op_bin . . . ) | les opérations arithmétiques se font sur les expressions de même type, sauf les int et les float. | this operation can't be applied to this giving types |
| 3.33 (esp -> op_bin . . . ) | les opérations faites sur des expressions du type booléen doivent être booléennes | this operation can be done only on boolean types |
| 3.33 (esp -> op_bin . . . ) | le UnaryMinus n'est applicable que sur les int et les float | operation can't be applied on this giving types |
| 0.1 (Type -> ...) | les types reconnue doit être définie dans l'Environnement_Type | type not defined |

| | | |
|---|---|---|
| 0.2 (identifier ->...) | on doit trouver une définition associée au nom reconnue dans l'environnement env_exp | variable not declared |
| 3.51 | l'opération Modulo n'est applicable que sur les entiers. | you can apply % only on integers |
| 3.66 Selection (a.x, this.x,a.getx(), ...) | a.method() l'opérande de gauche 'a' n'est pas une classe | can't call a method : left operand is not a class |
| 3.66 Selection (a.x, this.x,a.getx(), ...) | a.method() l'opérande de droite 'method' n'est pas une méthode | this ident is not a method |
| 3.66 Selection (a.x, this.x,a.getx(), ...) | la méthode a des champs différent de la méthode qui est override | this method has a different signature |

## 2.2 Erreur d'exécution du code assembleur

| Type | Description | Messages d'erreurs |
|---|---|---|
| Erreur de mémoire | Débordement du a une valeur trop grande | Overflow during arithmetic operation |
| Erreur de mémoire | Débordment de la pile : il n'y a plus de place dans la pile | Stack Overflow |
| Erreur de mémoire | Debordement du tas : il ne reste plus d espace pour la création d'un nouvel objet | Heap Overflow Error |
| Erreur de type | type en entrée non compatible | Input/Output error |
| Erreur de division | on ne peut pas diviser par zero | division by zero |
| Erreur assignement | assignement a une variable nulle | Dereferencement null |

# 3 Extension math

## 3.1 Présentation de l'extension :

Les applications de la trigonométrie sont très variées : l'architecture, l'acoustique, l'optique, la météorologie, l'astronomie ... Il en découle donc l'importance des calculs effectués, mais surtout le degré de gravité qu'une simple erreur peut engendrer.

Il s'agit dans cette extension d'implémenter les fonctions trigonométriques *sin*, *cos*, *arctan* ainsi que *arcsin*, qui prennent un flottant comme argument, tout en respectant le cahier de charges qui allie précision et rapidité.

A ces quatre fonctions s'ajoutent les fonctions *racine*, qui prend comme paramètre un flottant et retourne sa racine, et la fonction *puiss*, qui prend comme paramètre un flottant et un entier $n$, et qui retourne la puissance $n$-ème du flottant. On a décidé de rajouter ces deux fonctions pour optimiser la bibliothèque demandée.

## 3.2 Limitations de l'extension :

### 3.2.1 Précision :

Pour évaluer la précision des fonctions *sin* et *cos*, on s'est proposé de comparer les valeurs données par la bibliothèque implémentée, avec celle du langage Java, qui est infiniment précise. En effet, les valeurs retournées par cette dernière sont d'une imprécision maximale d'1 *ulp*. D'où la légitimité de la référence choisie.

L'échantillon qu'on a choisi est l'intervalle $[0, 1000000]$, ce même intervalle a été divisé en trois, et ce pour une meilleure analyse des résultats obtenus : $[0, 100]$, qui contient 51 valeurs distancées de 2, $[100, 10000]$, avec des valeurs qui sont distancées de 50 dans un premier temps et de 20000 ensuite, et finalement $[100000, 1000000]$ avec des valeurs distancées de 20000 au début et 30000 après.

Voici un tableau qui montre dix valeurs pour chacun des trois intervalles pour la fonction *cos*.
On remarque que pour les valeurs du premier intervalle, les résultats obtenus sont quasiment exacts, et que pour le second, l'erreur varie entre $10^{-5}$ à $10^{-4}$, et entre $10^{-4}$ à $10^{-3}$ pour le troisième.
S'il est bien-sûr évident qu'une valeur exacte puisse être trouvée partout, on peut néanmoins se permettre une telle marge d'erreur, puisque d'une part elle est très faible, et d'autre part, les calculs trigonométriques se font rarement sur de tels arguments. Le plus crucial est donc d'avoir des résultats exacts pour les petits arguments, ce qui est le cas. Ce constat est le même pour la fonction *sin*, ce qui est attendu vu qu'on peut directement obtenir la valeur du *sin* à partir du *cos*.

On a procédé de manière analogue pour évaluer la fonction *arctan*, avec le même intervalle $[0, 1000000]$, mais avec des sous-intervalles différents. Cette fois-ci, on l'a découpé en 4 : le premier intervalle est $[0, 10]$, avec un pas de 1, le second est $[10, 100]$ avec un pas de 5, le troisième est $[100, 1000]$ avec un pas de 100, et finalement $[10000, 1000000]$ avec un pas de 30000.

| | cos du Java | cos de l'extension |
|---|---|---|
| 0 | 1 | 1 |
| 2 | -0.41614683 | -0.4161468 |
| 10 | -0.83907152 | -0.8390714 |
| 16 | -0.9576594 | -0.95765954 |
| 20 | 0.40808206 | 0.40808165 |
| 40 | -0.66693806 | -0.6669387 |
| 54 | -0.82930983 | -0.82930976 |
| 60 | -0.9524128 | -0.95241296 |
| 72 | -0.96725058 | -0.9672503 |
| 84 | -0.68002349 | -0.6800214 |
| 100 | 0.86231887 | 0.86233556 |
| 350 | -0.28363327 | -0.28362942 |
| 500 | -0.88384927 | -0.883853 |
| 700 | -0.83910432 | -0.8391087 |
| 900 | 0.06624670 | 0.066239715 |
| 10000 | -0.95215536 | -0.9521695 |
| 20000 | 0.81319969 | 0.8132534 |
| 60000 | -0.28854362 | -0.2892933 |
| 80000 | -0.79187464 | -0.79164904 |
| 100000 | -0.99936080 | -0.9994483 |
| 180000 | 0.76953727 | 0.7581017 |
| 240000 | 0.38939500 | 0.386508 |
| 320000 | -0.87083496 | -0.8715602 |
| 400000 | 0.98978925 | 0.99118495 |
| 500000 | -0.98406100 | -0.9819012 |
| 590000 | -0.86526272 | -0.8813694 |
| 740000 | -0.54698358 | -0.5588615 |
| 800000 | 0.95936553 | 0.96489525 |
| 980000 | 0.55807565 | 0.5441009 |
| 1000000 | 0.93675212 | 0.9282601 |

FIGURE 1 – Tableau de comparaison des valeurs estimées de la fonction cos par les bibliothèques Math de Java et de l'extension.

Ici, on remarque une erreur uniforme de $10^{-8}$ à $10^{-7}$, ce qui est excellent. En effet, les calculs qu'on fait sont en simple précision, c'est-à-dire qu'un nombre peut être représenté à au plus 8 chiffres après la virgule, une erreur de l'ordre de $10^{-8}$ est ce qu'on peut obtenir de mieux alors, si ce n'est le résultat exact. Cette précision est identique pour la fonction *arcsin*, vu qu'on la déduit de la fonction *arctan*.

| | arctan du Java | arctan de l'extension |
|---|---|---|
| 0 | 0 | 0 |
| 2 | 1.10714871 | 1.1071484 |
| 4 | 1.32581766 | 1.3258177 |
| 7 | 1.42889927 | 1.4288993 |
| 10 | 1.47112767 | 1.4711277 |
| 25 | 1.53081763 | 1.5308176 |
| 45 | 1.54857776 | 1.5485778 |
| 70 | 1.55651158 | 1.5565116 |
| 85 | 1.55903216 | 1.5590322 |
| 100 | 1.56079666 | 1.5607967 |
| 200 | 1.56579636 | 1.5657964 |
| 300 | 1.56746300 | 1.567463 |
| 500 | 1.56879632 | 1.5687964 |
| 700 | 1.56936775 | 1.5693678 |
| 1000 | 1.56979632 | 1.5697963 |
| 10000 | 1.57069632 | 1.5706964 |
| 100000 | 1.57078632 | 1.5707864 |
| 400000 | 1.57079382 | 1.5707939 |
| 700000 | 1.57079489 | 1.5707949 |
| 1000000 | 1.57079532 | 1.5707954 |

FIGURE 2 – Tableau de comparaison des valeurs estimées de la fonction arctan par les bibliothèques Math de Java et de l'extension.

### 3.2.2 Rapidité :

Pour estimer la rapidité de notre bibliothèque, on a testé sur la fonction *sin*, pour laquelle on a fait trois tests, portant sur trois boucles. Les résultats illustrés ci-dessus indiquent que la croissance du temps d'exécution est exponentielle, néanmoins les valeurs obtenues sont très satisfaisantes, et ce même pour des échantillons relativement importants ( 100000 éléments ).

| | Taille de la boucle | Temps en secondes |
|---|---|---|
| Test 1 | 10000 | 1.5 |
| Test 2 | 100000 | 90 |
| Test 3 | 200000 | 385 |

FIGURE 3 – Tableau représentant le temps d'exécution de la bibliothèque Math.