Projet Génie Logiciel

Manuel Utilisateur

Aurélien VILMINOT
Damien CLAUZON
Guilherme KLEIN
Léon ROUSSEL
Pierre ARVY

24 janvier 2022

Résumé

La documentation utilisateur s'adresse à un futur utilisateur du compilateur Deca développé durant le projet de Génie Logiciel. Il sera laissé au lecteur le soin de connaître la syntaxe et les spécifications détaillées du langage Deca. Outre l'implémentation du compilateur classique, il sera présenté les détails de l'extension TAB. Ces détails concerneront le mode opératoire pour utiliser cette dernière ainsi que les limitations des fonctionnalités de l'extension.

Table des matières

1 Compilateur

1.1 Utilisation

Le compilateur implémenté dispose de différentes options lors de la saisie de la commande `decac`. Ces dernières sont listées ci-dessous et les détails associés y sont spécifiés :

`-b`	Affiche une bannière indiquant le nom de l'équipe
`-p`	Arrête decac après l'étape de construction de l'arbre, et affiche la décompilation de ce dernier
`-v`	Arrête decac après l'étape de vérifications
`-n`	Supprime des tests à l'exécution [1] (*correspondant aux points 11.1 à 11.3 de la section [Semantique]*)
`-r X`	Limite les registres banalisés disponibles à $R_0 \dots R_{X-1}$, avec $4 \leq X \leq 16$
`-d`	Active les traces de "debug". Répéter l'option plusieurs fois pour avoir plus de traces
`-P`	S'il y a plusieurs fichiers sources, la compilation des fichiers est effectuée en parallèle

Sans option et noms de fichiers, la commande `decac` affiche l'aide utilisateur regroupant les différentes options ci-dessus. La syntaxe d'utilisation de l'exécutable `decac` est :

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

Les fichiers Deca doivent avoir l'extension `.deca` pour être compilés et peuvent être placés à n'importe quelle place dans la commande par rapport aux options.

Le compilateur est prévu pour fournir du code assembleur exécutable sur une machine IMA. Les détails techniques quant à son fonctionnement ne seront pas détaillés dans ce document. Brièvement, la compilation se déroule en trois étapes :

1. Analyse syntaxique
2. Analyse contextuelle
3. Génération de code

Une étape ne peut s'exécuter que si les précédentes se sont correctement déroulées.

1.2 Limitations

Concernant le compilateur Deca, sauf deux limitations, toutes les spécifications fournies par le polycopié ont été implémentées.

L'instruction **`instanceof`** n'est pas fonctionnelle et ne peut donc être utilisée dans un programme Deca. En cas d'utilisation de cette dernière dans une condition booléenne, cette instruction sera ignorée par le programme Deca. Soit le programme suivant :

```
1 class A {}
2 class B {}
3 {
4    A a = new A();
5    if (a instanceof B) {
6       println("foo");
7    }
8 }
```

Du fait de la non-implémentation de l'instruction `instanceof`, le programme ci-dessus affiche la chaîne de caractères "*foo*" sur la sortie standard.

L'instruction **`cast`** fonctionne seulement avec des types primitifs. En d'autres termes, il n'est actuellement pas possible de réaliser l'instruction `cast` avec des classes. Son comportement est indéfini dans ce cas et peut retourner une erreur de type `DecacInternalError`. Soit le programme Deca suivant :

1. Débordement mémoire, entrée/sortie utilisateur, débordement de pile, débordement arithmétique sur les flottants ou sur une division entière par 0, déréférencement de null, absence de return dans une méthode

```
1 class A {}
2 class B extends A{}
3
4 {
5    B b = new B();
6    A a = (A) (b);
7    println("foo");
8 }
```

Le programme ci-dessus peut compiler et être exécuté par la machine abstraite `IMA`. Il affiche la chaîne de caractères "*foo*" sans effectuer la transtypage indiqué à la ligne 6.

Enfin, la dernière limitation du compilateur Deca concerne la taille limite du tas due à la machine abstraite `IMA`. Cette taille maximale est de 10 000 mots. L'unité de mots n'étant pas forcément explicite pour un utilisateur, voici quelques ordres de grandeurs :

— Déclaration d'une classe : 1 mot
— Déclaration d'un champ dans une classe : 1 mot

Si l'utilisateur dépasse cette taille maximale de 10 000 mots, alors l'erreur *heap overflow* est soulevée lors de l'exécution du programme (cf. 1.3.3).

1.3 Messages d'erreur

1.3.1 Analyse syntaxique

Différentes erreurs peuvent être retournées dans l'analyse syntaxique de Deca. Certaines proviennent directement d'ANTLR, et affichent la chaîne de caractère qui provoque l'erreur. Cette dernière sera nommée $chaine$ pour le lexer, et $token$ pour le parser. Ci-dessous, la liste exhaustive des messages d'erreurs et des configurations qui les provoquent.

1. **Erreur :** *token recognition error at : 'chaine'*

   La chaîne de caractères $chaine$ n'est pas reconnue dans la lexicographie de Deca.

2. **Erreur :** *no viable alternative at input 'token'*

   Le parser ne reconnaît pas une ligne comme une règle valide dans la syntaxe de Deca, et s'est arrêté à la position de $token$.

3. **Erreur :** *missing 'token$_1$' at 'token$_2$'*

   Il manque un token $token_1$ à la position de $token_2$ pour former une règle valide dans la syntaxe de Deca.

4. **Erreur :** *extraneous input 'token$_1$' expecting token$_2$*

   Le token $token_1$ est superflu, et peut être retiré ou remplacé par $token_2$ pour former une règle valide dans la syntaxe de Deca.

5. **Erreur :** *mismatched input 'token' expecting {list}*

   Le token $token$ est superflu, et peut être remplacé par un des éléments de la liste $list$ pour former une règle vaide dans la syntaxe de Deca.

6. **Erreur :** *left-hand side of assignment is not an lvalue*

   L'expression se trouvant à gauche du signe = doit être de type `LValue`. Se référer à la règle `assign_expr` dans la syntaxe de Deca.

7. **Erreur :** *literal integer could not be parsed*

   Le littéral entier lève une erreur de type `NumberFormatException` et ne peut pas être traité par le compilateur Deca. Cela peut signifier que l'entier est trop grand.

8. **Erreur :** *literal float cannot be infinite*
   Le littéral flottant est trop grand, et ne peut pas être traité par le compilateur Deca.

9. **Erreur :** *literal float cannot be NaN*
   Le littéral flottant n'est pas un nombre, et ne peut pas être traité par le compilateur Deca.

10. **Erreur :** *literal float could not be parsed*
    Le littéral flottant lève une erreur de type `NumberFormatException` différente des deux précédentes, et ne peut pas être traité par le compilateur Deca.

1.3.2 Analyse contextuelle

Au niveau contextuel, différentes erreurs peuvent être retournées par le compilateur Deca. Ci-dessous, la liste exhaustive des messages d'erreurs et des configurations qui les provoquent. Dans certains cas, le message d'erreur retourné contient le nom de l'instruction (nom de variable, nom de classe…) erronée. Ce dernier étant dépendant du programme en question, nous le nommerons $name$.

1. **Erreur :** *These types are not compatibles. Expected type :* $name_1$*. Current type :* $name_2$
   Le type de l'expression assignée n'est pas compatible avec l'expression assignable. L'erreur peut être relevée lors d'une initialisation de variable, d'un passage de paramètre lors d'un appel de méthode ainsi que pour le type de retour d'une méthode. Se référer à la méthode `assign_compatible()`.

2. **Erreur :** *Expression type must be boolean*
   L'expression au sein d'une condition doit être de type booléen. Ces expressions sont présentes au sein de `if`/`else` et `while`.

3. **Erreur :** *Binary operation is only allowed for int or float types*
   Une opération arithmétique ne se réaliser qu'entre deux éléments de types `int` et/ou `float`. L'erreur peut être provoquée par la somme de deux classes, par exemple.

4. **Erreur :** *Boolean operation is only allowed for boolean type*
   Une opération booléenne, avec les opérateurs && et/ou | |, ne peut s'effectuer qu'avec des opérandes de types booléennes.

5. **Erreur :** *Equals or not equals comparison is only allowed with int, float, class, null or boolean types*
   Une limitation de type est imposée lors d'un test d'égalité et d'inégalité.

6. **Erreur :** *This comparison works only with int or float types*
   Outre l'égalité et l'inégalité, le restant des comparaisons ne sont possibles qu'avec des opérandes de type `int` et/ou `float`. L'erreur peut donc être provoquée par le test d'infériorité entre deux classes.

7. **Erreur :** *Impossible to print this type of element :* $name$
   Seulement des éléments de types `float`, `int` et `string` peuvent être affichés sur la sortie standard.

8. **Erreur :** *The index for array access must be an integer*
   L'erreur peut être provoquée par un accès à un élément via un indice de type `float`. Par exemple, `int elem = myTab[2.0]` provoque cette erreur.

9. **Erreur :** *Identifier for array access must be a 1D or 2D array of float or int*
   L'accès à un tableau ne peut se faire que sur un tableau de dimension 1 ou 2. Le cas échéant, l'erreur sera relevée.

10. **Erreur :** *The origin type ($name_1$) cannot be cast into the destination type ($name_2$)*

    Le transtypage ne peut s'effectuer qu'avec des types compatibles. Pour de plus amples informations, se référer à la méthode `cast_compatible()`.

11. **Erreur :** *Implicit cast works only with int to float*

    Le transtypage implicite ne fonctionne uniquement du type `int` vers le type `float`.

12. **Erreur :** *The super-class name is a not a class : $name$*

    Si la super-classe d'une classe n'est pas un élément de type classe alors l'erreur sera relevée.

13. **Erreur :** *The class name '$name$' is already declared*

    La déclaration d'une classe n'est possible si cette dernière a été déclarée au préalable.

14. **Erreur :** *Void type cannot be declared as a field type*

    Un champ d'une classe ne peut être de type `void`.

15. **Erreur :** *Super-class redefinition identifier must be a field*

    Lors de la redéfinition d'un champ dans une classe étendant une autre, le champ doit être obligatoirement un champ dans la classe étendue. L'erreur sera relevée si le nouveau champ tente de redéfinir une méthode de la classe étendue.

16. **Erreur :** *Field name '$name$' already declared*

    Au sein d'une même classe, si deux champs possèdent le même nom, cette erreur sera relevée.

17. **Erreur :** *Method prototype must be same as inherited*

    Lors de la redéfinition d'une méthode, la signature de celle-ci doit être la même que de la méthode définie dans la super-classe.

18. **Erreur :** *Return type must be a subtype of inherited method return*

    Lors de la redéfinition d'une méthode, la signature de celle-ci doit être la même que de la méthode définie dans la super-classe.

19. **Erreur :** *Super-class symbol must be a method definition*

    Lors de la redéfinition d'une méthode dans une classe étendant une autre, la méthode doit être obligatoirement une méthode dans la classe étendue. L'erreur sera relevée si le nouveau méthode tente de redéfinir une méthode de la classe étendue.

20. **Erreur :** *Method name '$name$' already declared in the class*

    Au sein d'une même classe, si deux méthodes possèdent le même nom, cette erreur sera relevée.

21. **Erreur :** *Void type is not allowed for this parameter : $name$*

    Un argument de méthode ne peut être de type `void`.

22. **Erreur :** *Param name '$name$' already used*

    Deux paramètres d'une même méthode doivent avoir un nom distinct. Le cas échéant, l'erreur sera relevée.

23. **Erreur :** *Void cannot be the type of a variable*

    Une variable ne peut être de type `void`.

24. **Erreur :** *Variable name '$name$' already declared*

    Dans un même contexte, deux variable ne peuvent être définies par le même nom.

25. **Erreur :** *Undeclared identifier :* $name$

    L'erreur est relevée lorsqu'un élément est utilisé sans avoir été déclaré au préalable.

26. **Erreur :** *Undefined type identifier :* $name$

    L'erreur est relevée lorsqu'un type inconnu est utilisé.

27. **Erreur :** *Impossible to find the method :* $name$

    Dans le contexte dans laquelle elle est appelée, le nom de la méthode n'est pas connu.

28. **Erreur :** *InstanceOf works only with two class types operands*

    L'instruction `instanceof` ne peut être utilisée qu'avec des classes. Rappel : cette instruction n'est pas implémentée dans le compilateur.

29. **Erreur :** *This identifier is not a class :* $name$

    Une méthode ne peut être appelée qu'à partir d'une variable instanciant une classe. Le cas échéant, l'erreur est relevée.

30. **Erreur :** *Number of parameters doesn't match method signature*

    Lors de l'appel d'une méthode, le nombre de paramètres fournis doit être le même que le nombre d'arguments attendu par cette même méthode.

31. **Erreur :** *Modulo operation is only allowed for int type*

    L'opération modulo (%) n'est possible qu'entre deux opérandes de type `int`.

32. **Erreur :** *Cannot instantiate a class, the type must be a class*

    Lors de l'instanciation d'une classe avec l'instruction `new`, l'identifiant qui suit doit faire référence à une classe. Par exemple, soit `int x;` une variable, l'instruction `new x();` relève l'erreur en question.

33. **Erreur :** *An array must contain float or int*

    Le type primitif d'un tableau doit être `int` ou `float`.

34. **Erreur :** *Index of array must be an integer*

    Lors de l'initialisation d'un tableau, la taille des dimensions de ce dernier doivent être des éléments de type `int`.

35. **Erreur :** *The dimension of an array cannot be greater than 2*

    Un tableau ne peut être instancié avec une dimension supérieure à 2 (cf. 2.3).

36. **Erreur :** *Not operation is only allowed for boolean type*

    L'opérateur not (`!`) n'est utilisable qu'avec des expressions booléennes.

37. **Erreur :** *Method return type is void*

    Au sein d'une méthode, l'instruction `return void;` n'est pas autorisée et provoque ce message d'erreur.

38. **Erreur :** *Only 'length' field is allowed for array*

    Les tableaux, quelque soit leurs dimensions, possèdent un unique attribut "`length`" permettant d'accèder à leur taille. L'usage d'un autre attribut sur des tableaux provoquera cette errreur.

39. **Erreur :** *Cannot select field from non-class type :* $name$

    Lors de l'accès à un champ d'une classe, la variable doit être une classe instanciée au préalable.

40. **Erreur :** *The identifier '*$name$*' is protected and it is impossible to access it in the main program*

Lors de l'accès à un champ d'une classe dans le programme principal, la visibilité du champ doit être publique. Le cas échéant, l'erreur est relevée.

41. **Erreur :** *Impossible to select this identifier :* $name$

    L'erreur est relevé si l'accès à un champ n'est pas possible pour des raisons autres que celles citées dans les erreurs précédentes (40-41).

42. **Erreur :** *Impossible to use 'this' identifier in the main program*

    L'utilisation de `this` dans le programme principal est proscrite et relève cette erreur.

43. **Erreur :** *Minus operation is only allowed for int or float types*

    L'opérateur unaire moins (`-`) n'est utilisable qu'avec des éléments de type `int` ou `float`.

1.3.3 Génération de code

Certaines erreurs n'apparaissent pas durant la compilation Deca, et sont levées uniquement à l'exécution. Celles-ci sont listées ci-dessous.

1. **Erreur :** *overflow during arithmetic operation*

   Une opération arithmétique a provoqué un débordement, soit dans un calcul sur les flottants, soit dans un division (ou reste de la division) par zéro.

2. **Erreur :** *stack overflow*

   Un débordement de la pile a eu lieu. Cela peut arriver si un utilisateur essaye d'allouer trop de variables globales.

3. **Erreur :** *heap overflow*

   Un débordement du tas a eu lieu. Cela peut arriver si un utilisateur essaye d'allouer trop de classes, trop de champs dans une classe, ou trop d'éléments dans un tableau.

4. **Erreur :** *input/output error*

   Une valeur entrée par l'utilisateur a causée une erreur. Cela peut arriver si un utilisateur rentre un entier quand un flottant est demandé, et inversement.

5. **Erreur :** *null dereference*

   Une variable dont la valeur est `null` a été utilisée pour un appel de fonction ou pour accéder à un attribut.

6. **Erreur :** *end of method* $name$ *without return instruction*

   Une méthode dont la signature est différente de `void` a été accédée sans instruction `return` à la fin.

7. **Erreur :** *index out of bounds*

   Un accès à un tableau a été fait avec un index qui est soit négatif, soit supérieur ou égal à la taille du tableau.

2 Extension

2.1 Présentation

Outre l'implémentation du compilateur comme spécifié par le sujet, une extension a été ajoutée au langage Deca. L'objectif de cette extension est d'étendre le langage Deca pour y ajouter et manipuler des tableaux. Concernant la syntaxe, elle s'inspire fortement de celle de Java. Une bibliothèque de calcul matriciel a également été implémentée dans le langage Deca (fichier *MatrixLibrary.decah*).

2.2 Mode opératoire

2.2.1 Déclaration d'un tableau

La déclaration d'un tableau se fait à l'aide :
- D'un **type primitif** (`int` ou `float`)
- De sa **dimension**, représentée par les paires de crochets suivant directement le type primitif. Le nombre de paire de crochet est égal à la dimension du tableau.
- De son **identifiant**

```
1 <type>[] <identifiant>;
2 <type>[][] <identifiant>;
```

La forme de la déclaration d'un tableau à une dimension (respectivement 2 dimensions) est donnée ligne 1 (respectivement ligne 2).

2.2.2 Initialisation d'un tableau

L'initialisation d'un tableau se fait à l'aide d'une des formes ci-dessous, selon sa dimension. Tous les éléments du tableau créé sont initialisés à 0.

```
1 <type>[] <identifiant> = new <type> [<index>];
2 <type>[][] <identifiant> = new <type> [<index>][<index>];
```

La valeur de l'expression issue de `<index>` *doit être de type* `int`*, elle définit la taille du tableau.*

2.2.3 Assignation

Une fois des tableaux déclarés ou initialisés, il est possible de faire des assignations entre tableaux de même dimension.
Deux assignations ci-dessous à la ligne 3 et 4.

```
1 int[] tableau;
2 int[] second_tableau = new int[<taille>];
3 tableau = new int[<taille>];
4 tableau = second_tableau;
```

2.2.4 Accès au éléments

L'accès aux composantes d'un tableau se fait avec l'identifiant du tableau suivi des indices d'accès entre crochets.

```
float composante;
float[] tableau = new float[<taille>];
tableau[<index>] = <floatant>;
composante = tableau[<index>];
```

La valeur de l'expression issue de `<index>` *doit être de type* `int`*, elle correspond à l'indice de la composante à laquelle on accède. L'index d'accès doit être un entier compris dans l'intervalle* $[0, n-1]$*, où* n *est la taille du tableau. La première composante est accessible avec l'index* 0*, la seconde avec l'index* 1*, et ainsi de suite.*

2.2.5 Accès à la taille

L'accès à la taille du tableau est possible en appelant l'attribut `length` sur un tableau.

```
int taille = <identifiant>.length;
```

2.2.6 Bibliothèque

Comme spécifié dans l'extension TAB, une bibliothèque de calcul matriciel est implémentée. Cette dernière se trouve dans le fichier `MatrixLibrary.decah`. Il est à inclure dans le fichier contenant le code du programme principal. Contenant une classe avec des méthodes de calculs matriciels, il est nécessaire d'instancier la librairie dans le programme principal. La classe à instancier est `AlgebraLib`. Cela se réalise via les instructions suivantes :

```
#include "Library.decah"

{
   AlgebraLib Mat = new AlgebraLib();
}
```

Par la suite, l'objet implémenté doit être utilisé pour appeler les méthodes de calculs matriciels. L'ensemble des méthodes disponibles sont résumées dans les lignes qui suivent. Les détails de ces dernières ne seront pas explicités dans ce manuel utilisateur. Toutes les méthodes retournent des éléments de type `float`.

1. **Méthode :** `float[][] uniformMatrix(int height, int width, float value)`
   Retourne une matrice dont les composantes sont initialisées à la valeur `value`.

2. **Méthode :** `float[] uniformVector(int len, float value)`
   Retourne un vecteur dont les composantes sont initialisées à la valeur `value`.

3. **Méthode :** `void printMatrix(float[][] matrix)`
   Affiche une matrice sur la sortie standard.

4. **Méthode :** `void printVector(float[] vector)`
   Affiche un vecteur sur la sortie standard.

5. **Méthode :** `float[] linspace(float min, float max, float step)`
   Retourne un vecteur dont les composantes sont comprises entre `min` compris et `max` non compris, et espacées par le pas `step`.

6. **Méthode :** `float[][] reshape(int height, int width, float[] vector)`
   Retourne une matrice de dimensions `height` x `width` correspondant au vecteur `vector` redimensionné.

7. **Méthode :** `boolean isSquarred(float[][] mat)`
   Retourne `true` si la matrice `mat` est carrée, `false` sinon.

8. **Méthode :** `boolean sameSize(float[] vec1, float[] vec2)`
   Retourne `true` si les vecteurs `vec1` et `vec2` sont de même taille, `false` sinon.

9. **Méthode :** `boolean sameDim(float[][] mat1, float[][] mat2)`
   Retourne `true` si les matrices `mat1` et `mat2` sont de même taille, `false` sinon.

10. **Méthode :** `float[][] addMat(float[][] mat1, float[][] mat2)`
    Retourne une matrice issue de l'addition des matrices `mat1` et `mat2`.

11. **Méthode :** `float[][] subMat(float[][] mat1, float[][] mat2)`
    Retourne une matrice issue de la soustraction des matrices `mat1` et `mat2`.

12. **Méthode :** `float[][] multMat(float[][] mat1, float[][] mat2)`
    Retourne une matrice issue de la multiplication des matrices `mat1` et `mat2`.

13. **Méthode :** `float[] addVector(float[] vector1, float[] vector2)`
    Retourne un vecteur issu de l'addition des vecteurs `vector1` et `vector2`.

14. **Méthode :** `float[] subVector(float[] vector1, float[] vector2)`
    Retourne un vecteur issu de la soustraction des vecteurs `vector1` et `vector2`.

15. **Méthode :** `float multVector(float[] vec1, float[] vec2)`
    Retourne le flottant correspondant au produit scalaire (euclidien) entre les vecteurs `vec1` et `vec2`.

16. **Méthode :** `float[][] transpose(float[][] mat)`
    Retourne la matrice correspondant à la transposée de la matrice `mat`.

2.3 Limitations

L'implémentation des tableaux en Déca présente quelques limitations :

— Les tableaux ne peuvent contenir que **deux types primitifs** : `int` et `float`.
— La dimension des tableaux est d'au plus deux. D'un point de vue mathématique, il y a donc uniquement la possibilité de créer des **vecteurs** et des **matrices** d'éléments primitifs.
— Aucune opération n'est implémentée nativement dans le langage Deca sur les tableaux. Il n'y a donc pas la possibilité de sommer (+) ou de multiplier (*) des tableaux. Les opérateurs d'égalité (==) et de non-égalité (!=) ne sont pas non plus fonctionnels sur les tableaux.
— Chaque tableau occupe un mot pour stocker sa taille et un mot par élément qu'il contient. La taille maximale d'un tableau est donc de 9999 (se référer aux limitations du compilateur pour comprendre pourquoi).
— Le comportement d'un tableau manipulé sans avoir initialisé est indéfini.