Projet Génie-Logiciel
Manuel Utilisateur

BECK BALIHAUT-UTARRE Loula GILLET Maxime
GILLIOT Mathis KRZISCH Pierre-Emmanuel TROMPETTE Thomas
Équipe 27

Janvier 2017

1 Introduction

Ce document est un mode d'emploi du compilateur `decac`. Ce compilateur compile exclusivement le langage Deca. De manière générale, le langage est correctement implémenté. Toutefois, certaines spécifications ne sont pas implémentées.

2 Utilisation du compilateur

La syntaxe d'utilisation de l'exécutable `decac` est :

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

On définit de la manière suivante les options du compilateur deca :

Option	Description
`-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
`-v` (vérification)	Arrête `decac` après l'étape de vérifications
`-r X` (registers)	Limite les registres banalisés disponibles à `R0, ..., R{X-1}`, avec `4 <= X <= 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
`-ARM` (extension ARM)	Lance la génération de code en ARM

Les options `-p` et `-v` sont incompatibles.

Lors d'une mauvaise utilisation des options, l'option display est directement affichée dans le terminal sans compiler.

3 Limitations du compilateur

Ce compilateur possède certaines limitations de langage et d'implémentation. Ci-dessous sont listées ces limitations, avec leurs effets pour l'utilisateur. Pour certaines, la cause de ces limitations est explicitée.

— Option `-n` non implémentée :
Renvoie le message d'erreur ``not yet implemented'' puis affiche l'option display du compilateur.

— Option `-P` implémentée mais test non concluant :
Le programme compile les fichiers demandés mais il n'est pas certain qu'ils soient compilés en parallèle.

— Caractères échappés pas pris en compte :
Le message est affiché avec les caractères à échapper. Par exemple `print(``je dis \'' bonjour\'' '')` affiche : `je dis \''bonjour\''` et non `je dis ``bonjour''`.

— Méthode sans return marche :
Le compilateur ne renvoie pas d'erreur lors de l'oubli d'un return dans une méthode. L'utilisateur devra donc systématiquement vérifier ses méthodes sous risque d'avoir un résultat erroné car pas le résultat de sa méthode.

— `equals` non implémentée :
Si l'utilisateur utilise la méthode `equals` de la classe `Object`, le programme s'arrête normalement (HALT).

— Cast non implémenté :
Renvoie le message d'erreur ``Identifier *className* is not a Exp identifier, you can't call getExpDefinition on it'' puis ``internal compiler error'' avant de s'arrêter. Il est donc impossible d'utiliser cette fonction avec notre compilateur.

— `instanceof` non implémentée :
Renvoie le message d'erreur ``Identifier *className* is not a Exp identifier, you can't call getExpDefinition on it'' puis ``internal compiler error'' avant de s'arrêter. Il est donc impossible d'utiliser cette fonction avec notre compilateur.

— Asm non-implémentée :
Renvoie le message d'erreur ``not yet implemented'' puis ``internal compiler error'' avant de s'arrêter. Il est donc impossible pour un utilisateur d'ajouter "manuellement" du code assembleur.

— Déréférencement de `null` :
Renvoie le message d'erreur ``not yet implemented'' puis ``internal compiler error'' avant de s'arrêter. L'utilisateur saura donc qu'il y a un problème de déréférencement de `null` mais ne saura pas ou déboguer.

— `Selection` et `Assign` mal implémentées (car l'adresse du champ mal récupérée) : Comportement indéfini

4 Messages d'erreur

4.1 Message d'erreur de lexicographie et de syntaxe hors-contexte

Les messages d'erreur de type lexicographique ou de type syntaxique sont les messages d'erreur initiaux.

4.2 Message d'erreur de lexicographie et de syntaxe hors-contexte

Lorsqu'une erreur concernant la syntaxe contextuelle du programme est soulevée, l'utilisateur verra s'afficher une explication de l'erreur ainsi que le numéro de la règle dans la grammaire de Deca qu'il n'a pas respecté.

Voici donc une liste des différentes erreurs qui pourraient apparaître :

— Il n'y a pas de définition associée au nom *name* (règle 0.1)
— Le type *name* n'existe pas (règle 0.2)
— La classe *super* n'existe pas (règle 1.3)
— La classe *name* existe déjà (règle 1.3)
— Le nom *name* est déjà utilisé dans la classe (règle 2.3)
— La classe *super* n'existe pas (règle 2.3)
— Déclaration d'un champ de type void impossible (règle 2.5)
— Le nom *name* ne correspond pas à un champ dans la classe mère (règle 2.5)
— Déclaration d'un paramètre de type void impossible (règle 2.9)
— La classe *class* n'existe pas (règle 3.5)
— Le paramètre *name* existe déjà (règle 3.12)
— Déclaration d'une variable de type void impossible (règle 3.17)
— La variable *name* existe déjà (règle 3.17)
— La condition n'est pas un booléen (règle 3.22)
— Une méthode de type void ne peut pas renvoyer d'objet (règle 3.24)
— Affectation à un objet de type *type*1 une valeur de type *type*2 impossible (règle 3.28)
— Une expression de type *type* n'est pas imprimable (règle 3.31)
— Opération mod impossible avec des objets de type float (règle 3.33)
— Opération logique impossible entre des objets de type *type*1 et *type*2 (règle 3.33)
— Comparaison entre des objets de type *type*1 et *type*2 impossible (règle 3.33)
— Négation sur un objet de type *type* impossible (règle 3.37)
— Moins unaire sur un objet de type *type* impossible (règle 3.37)
— Opération arithmétique entre des objets de type *type*1 et *type*2 impossible (règle 3.37)
— Opération instanceof impossible entre des objets de type *type*1 et *type*2 (règle 3.40)
— *type* n'est pas un type correspondant à une classe (règle 3.42)
— Utilisation de this impossible dans un main (règle 3.43)

— Une méthode doit être appliquée sur un objet (règle 3.43)
— *class*2 n'est pas un type (règles 3.65 et 3.66)
— Le type *class*2 ne correspond pas à une classe (règles 3.65 et 3.66)
— Accès au champ *name* non-autorisé (règle 3.66)
— *name* ne correspond pas à un champ, à un paramètre ou à une variable (règles 3.67, 3.68 et 3.69)
— *name* n'est pas à un champ (règle 3.70)
— *class*2 n'est pas un type (règles 3.71)
— *class*2 n'est pas une classe (règle 3.71)
— *name* n'est pas une méthode (règle 3.71)
— Le nombre de paramètres ne correspond pas à la signature de la méthode *name* (règle 3.74)

4.3 Messages d'erreur d'exécution du code assembleur

L'utilisateur peut également soulever différents types d'erreurs lors de l'exécution du code assembleur IMA. Ces dernières ne seront pas visibles lors de la compilation. Les erreurs pouvant apparaître sont les suivantes :

— Error : Overflow during arithmetic operation
Cette erreur peut apparaître lors d'un calcul provoquant un overflow, par exemple lors d'une addition entre deux flottants qui donne un résultat supérieur à la capacité du système 32 bits.

— Error : Stack Overflow
Cette erreur apparaît lorsque la pile de sauvegarde est pleine. Cela peut par exemple arriver lorsque trop de variables sont déclarées.

— Error : Input/Output error
Cette erreur peut arriver lorsque l'utilisateur doit saisir la valeur d'une variable de manière intéractive. Si le type de la valeur en entrée ne correspond pas à la valeur attendue, cette erreur est levée.

— Error : dereferencement de null
Lors de la sélection d'un champ ou d'une méthode d'un objet, si l'objet en question est null, cette erreur apparaît.

— Error : tas plein
Lors d'un new, on teste que le tas n'est pas plein. En cas d'échec, le programme termine avec l'erreur : “Error : tas plein".

— ** IMA ** ERREUR ** Ligne XX : WINT/WFLOAT avec R1 indefini
Erreur soulevée lors de la tentative d'affichage en sortie d'une varible qui n'a pas été définie au préalable.

— ** IMA ** ERREUR ** Ligne XX : TypeOperation avec op1 : type et op2 : type
Erreur soulevée lorsque l'utilisateur tente d'effectuer une opération arithmétique sans avoir initialisé un des opérandes au préalable.

5 Extension ARM

5.1 Le mode opératoire pour l'utilisation

L'utilisation de l'extension n'est pas encore opérationnelle. Elle le sera lors de la soutenance. Un dossier ARM est présent, contenant un `Makefile` et un `README`.

Plusieurs actions devront être effectuées sur la machine afin de pouvoir exécuter du code en assembleur ARM. Il faut en effet disposer de QEMU sur sa machine, ainsi que des paquets `gcc-arm`. Aucune importation autre que celle de la bibliothèque standard n'est nécessaire.

Le `Makefile` rendu ne gère actuellement qu'un seul exemple. Il compile le fichier `.s` écrit en assembleur ARM (déjà présent dans le dossier source) et lance l'exécutable.

La commande `make` génère :
- les fichiers `.elf` et `.bin` dans le dossier `bin/`,
- les fichiers .o dans le répertoire courant.

Il lance également l'exécution du fichier `.bin` et il supprime les fichiers inutiles.

Dans les prochains jours et jusqu'à la soutenance, voici les différents points qui doivent être implémentés. La plupart d'entre eux sont en cours mais sont principalement présents sur une branch du Git à part afin de ne pas polluer le rendu final.
- Présence d'une option `-ARM`. Si elle est présente lors de la compilation, le code assembleur généré sera de l'assembleur ARM au lieu de l'assembleur IMA.
- Présence d'un script de test à part qui se servira de la même base de test que l'étape C initiale.

L'exemple du répertoire ARM permet d'imprimer l'entier présent dans une variable du fichier `startup.s`. Les fonctions permettant d'imprimer un entier et une chaîne de caractères sont présentes dans le fichier `test_int.c`. Le fichier en assembleur ARM (dans le dossier source) utilise une fonction de `test.c`. Le `Makefile` compile, réalise l'édition de lien et exécute le fichier résultant.

Cet exemple est destiné à résumer notre compréhension du fonctionnement de l'extension ARM. Nous savons désormais comment tester nos programmes en assembleur ARM au fur et à mesure de l'avancement de l'extension. Nous avons également mis du temps à réussir à afficher des entiers en sortie standard, ce qui nous a empêché de rendre une première implémentation fonctionnelle lors du rendu du compilateur sans extension.

5.2 Les limitations de notre extension

En ce qui concerne les limitations de l'extension, nous savons déjà que la génération de code ne sera pas aussi complète que pour l'assembleur IMA.
Seule la partie deca sans objet pourra être générée.
Nous ne pouvons cependant pas encore détailler à quel point cette partie sera implémentée.