Projet Génie Logiciel

Compilateur Deca

Manuel Utilisateur

Auteurs :
M. Benjamin ARGENSON
M. Jean-Baptiste LEFOUL
M. Thomas POUGET-ABADIE
M. Robinson PRADA-MEJIA
M. Alexandre PROY

Encadrants :
Pr. Philippe BODIGLIO
Pr. Catherine ORIAT

Version du 23 janvier 2017

Table des matières

1 Introduction

Ce document s'adresse à un utilisateur du compilateur **Deca**. Il permet de faciliter la prise en main du logiciel en expliquant d'une part comment l'utiliser et d'autre part en présentant les limites de son utilisation. De plus, ce manuel présente les différentes options disponibles ainsi que les différentes erreurs potentiels liées à une mauvaise utilisation du compilateur.

2 Mode opératoire

2.1 Programme principal

Le programme principal **Decac** est un compilateur du langage Deca. Il s'exécute sur un ou plusieurs fichiers de la forme : <répertoires/nom.deca> et traduit les fichiers en langage assembleur. Il met le résultat dans un fichier de la forme : <répertoires/nom.ass> qu'on peut ensuite exécuter avec **ima** (intérpréteur de la machine abstraite).

La syntaxe d'utilisation de l'exécutable decac est :
decac [[-p | -v] [-n] [-r X] [-d]* [-P] <fichier deca>...] | [-b]
La commande **decac** sans argument permet d'afficher les options disponibles.

Options	Description de l'option
-b (banner)	Affiche une bannière indiquant le nom de l'équipe de développeurs
-p (parse)	Arrête decac après la construction de l'arbre et affiche la décompilation de ce dernier
- v (vérification)	Arrête decac après l'étape de vérification
-n (no check)	Supprime les tests de débordements à l'exécution
-d (debug)	Affiche les traces de debug
-P (parallel)	Lance la compilation des fichiers en parallèle s'il y a plusieurs fichiers sources

Remarques :

- Les options -p et -v sont incompatibles.
- L'option '-b' ne peut être utilisée que sans autre option, et sans fichier source.
- Pour comparer deux objets on peut utiliser `"=="` qui donne le même résultat que la méthode **equals**.

2.2 Extension

Notre extension nous permet de compiler des fichiers .deca en fichiers .class pouvant être exécutés par une machine virtuelle java.
Afin d'utiliser cette option, le compilateur doit être exécuté de la manière suivante : **decac -B <fichier.deca>**.
Suite à l'exécution d'une telle ligne de commande, la compilation du fichier .deca résulte en la création d'un fichier du même nom avec l'extension .class.

Il suffit alors d'utiliser une machine virtuelle java afin d'exécuter le fichier résultant de la manière suivante : **java <fichier>**.

3 Limitations du compilateur

3.1 Programme principal

- Initialisation de variables :
  Une variable non initialisée n'a pas de valeur définie, par conséquent on ne peut y accéder avant qu'elle ne soit affectée d'une valeur. De plus, on ne peut déclarer une variable qu'au tout début du bloc d'instruction **Main**.
- Un objet alloué dynamiquement grâce à l'instruction **new** n'est jamais désalloué.
- Les nombres entiers doivent être compris entre -2^{-31} et $2^{31} - 1$ et les flottants entre 2^{-149}et $(2 - 2^{-23})2^{127}$ norme (IEEE-754).
  De plus, les opérations arithmétiques sur les flottants provoquent une erreur en cas de débordement alors que les opérations sur les entiers sont faites modulo 2^{31}.
- Pour ce qui est des classes, on ne peut déclarer des attributs et des méthodes ayant le même nom. De plus, les champs peuvent être définis comme protégés mais non comme privés (ils sont publics par défauts si rien n'est précisé).
- Les méthodes peuvent être redéfinies dans les sous-classes mais ne peuvent pas être surchargées.
- Ce compilateur a été implémenté de telle sorte qu'il n'utilise qu'au plus deux registres en même temps. Par conséquent l'utilisation de la pile est plus importante ce qui augmente le temps d'exécution du fichier compilé et peut provoquer un débordement de la pile.

3.2 Extension

Il est important de noter qu'au moment de l'écriture de ce manuel d'utilisation, ce compilateur ne permet de créer des fichiers en bytecode uniquement pour des fichiers .deca sans objet. Ainsi, toute utilisation de classes provoque une erreur de la part du compilateur avec l'option **-B**.

En dehors de l'utilisation d'objets, le compilateur avec l'option **-B** permet de compiler tout autre fichier pouvant être compilé sans cette option.

4 Messages d'erreur

4.1 Erreur de lexicographie

Une erreur est levée lors de l'analyse lexicale quand un mauvais token est généré (token **DEFAULT** : token qui ne correspond à rien pour la lexicographie Deca).
Par exemple : #**n**
Le message suivant est alors affiché : *"l'entrée n'a pas été reconnue -> DEFAULT"*

Une erreur est également générée lorsque le nom de fichier en argument de la commande #**include** n'existe pas.
Par exemple : #**include <nom_de_fichier>**
Le message suivant est alors affiché si le fichier ne se trouve ni dans le répertoire courant, ni dans ses sous répertoires : *"<nom_ de_fichier> : include file not found"*

4.2 Erreur de syntaxe hors-contexte

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.

Par exemple :

- L'oubli d'une accolade fermante à la fin du **Main** (**{ int a ;**), génère le message suivant : *"2 :0 : missing CBRACE at '<EOF>'"*
- L'oubli d'un ';' après une déclaration de variable (**{ int a }**), renvoie le message : *"1 :7 : mismatched input '}' expecting COMMA, SEMI"*

Mis à part les erreurs générées par ANTLR (another tool for language recognition), une erreur est levée lorsque lors d'une affectation, l'opérande de gauche ne correspond pas au token **IDENT** (identificateur).
Par exemple : **{ 1 = a ; }**, génère le message d'erreur suivant : *"left-hand side of assignment is not an lvalue"*.

4.3 Erreur de syntaxe contextuelle

Les erreurs de syntaxe contextuelle permettent de rejeter les programmes Deca "mal typés" et d'afficher un message donnant une indication sur l'erreur effectuée.
Ci-dessous, une liste présente les principales erreurs de syntaxe contextuelle et donne le

message d'erreur affiché correspondant à l'exemple donné.

Identificateur (variable, classe ou champs) non déclaré

```
{
    a = 1;
}
```

Message d'erreur : *"a was not declared (rule 0.1)"*

Type non défini

```
{
    toto a;
}
```

Message d'erreur : *"toto is not a valid type (rule 0.2)"*

Classe définie deux fois

```
class A {}
class A {}
```

Message d'erreur : *"A was already declared here : [1, 6] (rule 1.3)"*

Le type de la super classe n'est pas un objet

```
class A extends int {}
```

Message d'erreur : *"Expected class type, got int (rule 2.3)"*

Champs de type void

```
class A {
    void a;
}
```

Message d'erreur : *"Cannot have void type field (rule 2.5)"*

Surcharge d'une méthode

```
class A {
    void p() {}
}
class B extends A {
    void p(int a) {}
}
```

Message d'erreur : *"Signature is different from the one declared here : [2, 2] (rule 2.7)"*

Redéfinition d'une méthode dont le type renvoyé n'est pas un sous-type de la méthode héritée

```
class A {
    void p() {}
}
class B extends A {
    int p() {}
}
```

Message d'erreur : *"The return type is not a subType of the method declared here : [2, 2] (rule 2.7)"*

Utilisation de 'this' dans le programme principale

```
{
    this;
}
```

Message d'erreur : *"Cannot use 'this' outside of a class"*

Paramètre ou variable défini deux fois

```
Class A() {
    void m(int p, float p) {}
}
```

Message d'erreur : *"p was already declared here : [3, 11] (rule 3.17)"*

Paramètre de type 'void'

```
Class A() {
    void m(void p) {}
}
```

Message d'erreur : *"Cannot have void type parameter (rule 2.9)"*

Méthode définie deux fois

```
Class A() {
    void m() {}
    void m() {}
}
```

Message d'erreur : *"m was already declared here : [2, 0] (rule 2.7)"*

Initialisation d'une variable avec un type différent

```
{
    boolean a = 1;
}
```

Message d'erreur : *"expected boolean got int (rule 3.28)"*

Une condition sur un type différent d'un boolean

```
{
    float a = 1.2;
```

```
    if (a = 1.2) {}
}
```

Message d'erreur : *"expected boolean got int (rule 3.29)"*

Incohérence de type dans une opération arithmétique

```
{
    int a = 3 + true;
}
```

Message d'erreur : *"Expected int or float got boolean (rule 3.33)"*

Cast non autorisé

```
{
    int a = (int) (true);
}
```

Message d'erreur : *"Expected int or float got boolean (rule 3.39)"*

Variable de type 'void'

```
{
    void a = 0;
}
```

Message d'erreur : *"Cannot have void type variales (rule 3.17)"*

Méthode utilisée avec un nombre d'argument incohérent avec sa signature

```
class A {
    void m(int a) {}
}
{
    A a = new a();
```

```
    a.m();
}
```

Message d'erreur : *"Expected 1 arguments got 0 arguments"*

Sélection sur un type qui n'est pas un objet

```
{
    int a;
    a.p();
}
```

Message d'erreur : *"Expected Object got : int (rule 3.71)"*

Utilisation de l'opérateur not sur un type différent d'un boolean

```
{
    int a;
    !a;
}
```

Message d'erreur : *"Expected boolean got int (rule 3.63)"*

4.4 Erreur à l'exécution

Description de l'erreur	Contexte	Message généré
Division entière par zéro	3/0;	"Error : you divided by zero"
Déréférencement de null	A a = null; a.m();	"Error : dereferencement de null"
Débordement arithmétique (pour les flottants)	float a=100000.0; a*a*a*a*a;	"Error : Overflow during arithmetic operation"
Débordement de la pile		"Error : Stack Overflow"
Absence de return lors de l'éxécution d'une méthode		"Error : exit of method <nameOfMethod> without return"

5 Exemples

Voici un exemple très simple pouvant être compilé avec l'option **-B** :

```
{
    print("Hello world");
}
```

Afin de visualiser le bytecode généré suite à la compilation d'un tel fichier, il suffit d'utiliser la commande suivante :

javap -c <fichier.class>

Un programme plus complexe utilisant des variables et fonctionnant avec l'option **-B** est le suivant :

```
{

    int answer = 32;
    int numTries = 0;
    boolean done = false;
    int playerGuess;
    playerGuess = readInt();

    while ((playerGuess != answer) && (!(done))) {

        numTries = numTries + 1;

        if (numTries > answer) {

            println("Your guess is too high");

        } else {

            println("Your guess is too low");

        }

        if (numTries == 10) {

            println("you have exhausted all of your tries");
```

```
            done = true;

        }

        println("Enter a new guess : ");
        playerGuess = readInt();

    }

    if (answer == playerGuess) {

        println("You got it");

    }
}
```

Avec ce programme vous pourrez jouez au jeu de la devinette.

ENSIMAG
681, rue de la Passerelle
38400 Saint-Martin-d'Hères