Manuel Utilisateur

OUAZZANI CHAHDI Mohammed
BENTOUIMOU Yassine
ABDELWAHID Amine
AIT DRISS Salma
LEMDAGHRI ALAOUI Ibrahim

23 janvier 2023

Table des matières

1 Introduction

Dans le cadre de notre projet GL, nous avons réussi à programmer un compilateur pour le langage Deca. Ce compilateur a pour but de détecter les erreurs lexicales, syntaxiques, contextuelles et lors de la génération du code assembleur. Vous pouvez, avec notre compilateur, détecter toutes les erreurs possibles en langage deca. Nous vous précisant où se situe le problème dans votre code (nom du fichier, son emplacement, la ligne et la colonne), accompagné d'un message d'erreur qui vous explique la nature de votre erreur et comment vous pourriez éventuellement la résoudre.
Dans ce manuel nous vous expliquerons le mode d'utilisation de ce compilateur, ces limitations aisni que les différentes erreurs qui pourrait s'afficher à votre écran.

2 Limitation du compilateur

En ce qui concerne la partie sans objet c'est-à-dire les instructions, les opérations arithmétiques, booléennes et de comparaison, les assignement, ... à priori fonctionne bien et sans défaut.
Les limitations de ce compilateur résident dans la taille des registres que vous pouvez manipuler. Avec l'option -r , si vous choisissez un nombre de registres proche de 4, des erreurs de compilation peuvent s'afficher.
Les limitations pour la partie objet sont dans les instructions « cast » et « AsmMethodBody » . En effet, vous ne pouvez pas faire de cast avec un type class, ou générer le code d'une méthode avec un asmBody, car ces instructions ne sont pas implémentés. L'instruction instanceof a été implémenté, testée sur certains cas, mais pas suffisemment bien tésté. L'instruction instanceof a été implémenté de telle sorte à considérer les liaisons statiques de l'objet, au lieu des liaisons dynamiques. Aussi, la gestion des erreurs pour le retour des fonctions n'a pas été bien traité. Il est donc possible que des erreurs improbables surviennent lors de votre compilation, causé éventuellement par l'une de ses limitations ou des tests très complexes. Ceci est dû au fait que l'équipe a pris du retard lors de la génration du code pour l'instruction ifThenElse et dans la résolution de certains problèmes d'assemblage entre instructions dans la partie sans objet.

3 Les messages d'erreurs

3.1 Erreurs lexicales

3.1.1 $< file's\_name >$:*ligne* :*colonne* :token recognition error at : ' :' ...

ex :

```
{
int :; }
```

3.2 Erreurs syntaxiques

3.2.1 $< file's\_name >$:*ligne* :*colonne* : mismatched input ... expecting{...}

Votre code ne respecte pas la syntaxe deca, voila ce que vous pouvez écrire à la place.

ex :

```
{+7;}          // l'opérateur "+" doit avoir une opérande à gauche
```

3.2.2 file's name : include file not found

Le fichier que vous voulez inclure est introuvable.

ex :

```
include "hex.deca"          //le fichier hex.deca n'existe pas
```

3.2.3 left-hand side of assignment is not an lvalue

Vous faites l'assignement à une expression qui n'est pas une lvalue (identifier ou selection)

ex :

{ *int x ;*
5=x ; }

3.2.4 Circular include for file $< fiel'sname >$

Votre fichier s'inclus lui même. Il faut enlever un include d'un des fichiers.

3.2.5 FLOAT assigned can't be rounded either is way too big or way too small

Vous avez saisie un flottant soit beaucoup trop petit et il a été reconnu comme 0, soit beaucoup trop grand et il a été reconnu comme infini.
ex flottant très grand :
{ *float a = 5.0e100000000000 ;*}

ex flottant très petit :
{ *float a = 0.1e-100000 ;*}

3.2.6 INT is way too big for 32 bits

L'entier que vous avez saisi est trop grand. Les entiers sont codés sur 32 bits. L'intervalle toléré est [2 147 483 648, 2 147 483 647].
ex entier très grand :
{ *int a = 3000000000 ;*}

3.3 Erreurs contextuelles

3.3.1 Assignment Error

Cette erreur s'affiche lorsque vous assigner une expression à une autre qui ne sont pas de types compatibles. Les types compatibles sont lorsqu'une expression à droite est un sous type de l'expression à gauche, dans le cas d'un assignement d'un int à un float la conversion se fait implicitement.
ex :
{ *int a = true ;*}

3.3.2 Not boolean

Ce message s'affiche lorsque vous écrivez dans une condition (de : if, else if ou while) une expression non booléenne.
ex :

```
{
    if(1){                  // la condition dans le if n'est pas un booléen
        print("yes") ;
    }
}
```

3.3.3 Type not accepted

Vous ne pouvez pas afficher une expression autre qu'un entier, un flottant (décimal ou en hexadécimal) ou une chaine de caractère. Tout autre type d'expression sera refusé.
ex :
{ *print(true) ;*}

3.3.4 Erreur dans la condition $< Operator'sName >$ operands not permitted

Vous comparez deux expressions incomparables :
• Les opérateurs de comparaison : Avec les opérateurs : $\leq, \geq, >, <, =, !=$ vous pouvez comparer des entiers entre eux, des flottant entre eux ou un flottant et un entier.
Avec les opérateurs : = et != vous pouvez en plus comparer des expressions de type boolean et des objets.

• Les opérateurs arithmétiques : Avec les opérateurs arithmétiques (=, -, *, /) les seules expressions sur lesquels vous pouvez effectuer des opérations sont les expressions de type flottant ou entier. Avec le modulo (%) seules les expressions de type entier sont permises.
• Les opérateurs unaires : Pour l'opération "not" : (!) il faut que l'opérande soit de type booléen. Pour l'opération unaire (-) il faut que l'opérande soit de type entier ou flottant.

ex 1 :
{ boolean a = (1 <= true) ;}

ex 2 :
{ boolean a = (!2) ;}

3.3.5 Types not permitted for $< BooleanOperator'sName >$

Vous ne pouvez pas utiliser les opérateurs booléens (&& et ||) sur des expressions de type autre que booléen.
ex :
{ boolean a = 1 2.0 —— "non" ;}

3.3.6 Cast types are not valid

Les types des expressions ne sont pas compatibles.
ex :
```
{
    int c ;
    boolean a = (boolean) (c) ;
}
```

3.3.7 Cast with void type is not valid

Vous ne pouvez pas faire de cast de type void
ex :
{int m, c ;
m = (void) (c) ;}

3.3.8 Type = void impossible

Vous ne pouvez pas déclarer une variable de type void.
ex :
{int m ;
void c ;}

3.3.9 Le type de la variable n'est pas déclaré

Vous avez oublié de déclarer la variable
ex : {x ;}

3.3.10 Double déclaration de la variable

Vous ne pouvez pas déclarer 2 fois la même variable dans le même environnement.
ex1 :
```
{
    float x ;
    int x ;          // vous avez déjà déclaré la variable x dans le main
}
```

ex2 :
class A {

```
    Void methode(){
        int a ;
        boolean a ;          // vous avez déjà déclaré la variable a dans cette methode
    }
}
```

3.3.11 Parametre can't be void type

Vous ne pouvez pas déclarer un paramètre d'une méthode de type void, ou appeler une méthode avec un paramètre de type void.
ex :

```
class A{
    void a(void c){}
}
```

3.3.12 $< Type1 >$ is not a subtype of $< Type2 >$

Lorsque vous voulez changer la définition d'une méthode dans une class, il faut que la méthode ait la meme signature que celle définit dans sa superClass. C'est-à-dire, ils doivent avoir le même nom, même nombres de paramètres, et un type de retour sous-type de celui qui figure dans la méthode de la superclass. Chaque paramètre doit avoir le même type que celui dans la superClass et dans le même ordre.
ex :

```
class A {
    void a(){} ;
}
class B extends A {
    int a(){} ;          // int n'est pas un sous type de void
}
```

3.3.13 This method's signature doesn't match the one in superclass

Lorsque vous voulez changer la définition d'une méthode dans une class, il faut que la méthode ait la meme signature que celle définit dans sa superClass. C'est-à-dire, ils doivent avoir le même nom, même nombres de paramètres, et un type de retour sous-type de celui qui figure dans la méthode de la superclass. Chaque paramètre doit avoir le même type que celui dans la superClass et dans le même ordre.
ex :

```
class A {
    void a(int c){} ;
}
class B extends A {
    void a(float k){} ;
}
```

3.3.14 void type can't be a field

Vous ne pouvez pas déclarer un champ de type void.
ex :

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

3.3.15 superClass not defined

Vous ne pouvez pas hériter d'une class qui n'est pas encore déclarée.
ex :

class A extends B{} // la class B n'a pas encore été déclarée
class B {}

3.3.16 field not found

Vous ne pouvez pas sélectionner un champ qui n'existe pas.
ex :

```
class A {
    void method(){}
}
{
    A b = new A() ;
    b.a ;          // le champ a n'existe pas
}
```

3.3.17 expression before DOT is not a class Type

- Field selection

Vous ne pouvez pas sélectionner un champ avec un objet qui n'est pas de type class.
ex :

```
class A {
    int a ;
    void method(){}
}
{
    int b ;
    b.a ;          // b n'est pas de type class
}
```

- Call Method

Vous ne pouvez pas faire un appel d'une méthode avec un objet qui n'est pas de type class.
ex :

```
class A {
    protected int a ;
    void method(){}
}
{
    int b ;
    b.method() ;          // b n'est pas de type class
}
```

3.3.18 method can't be called in main without an object

Vous ne pouvez pas faire un appel d'une méthode dans le main, sans préciser l'objet.
ex :

```
class A {
    protected int a ;
    void method(){}
}
{
    method () ;          // appel de la méthode sans objet
}
```

3.3.19 Object doesn't have this method

La méthode à laquelle vous faites appel n'existe pas dans la class ou dans la superclass de l'objet.
ex :

```
class A {
```

```
    protected int a;
    void method(){}
}
{
    A c = new A();
    c.m ();          // la méthode m() n'existe pas dans la class A (class de l'objet c) ni dans la class
Object (superclass de A)
}
```

3.3.20 number of param not valid

Vous devez faire l'appel de la méthode avec le même nombre de paramètre.

ex :

```
class A {
    protected int a;
    void method (int c){}
}
{
    int d = 4, c = 5;
    method (d, c);          // la méthode ne devrait avoir qu'un seul paramètre au lieu de deux
}
```

3.3.21 declaring an attribut already in superClass and is not a Field

Vous avez déclaré un champ/attribut dont le nom a déjà été déclaré dans une super class mais pas autant que champs. Essayez de changer le nom de votre champs.

ex :

```
class A {
    void b(){}
}
class B extends A {
    int b;
}
```

3.3.22 can't redefine a field as a method

Vous avez déclaré une méthode dont le nom a déjà été déclaré dans une super class mais pas autant que méthode. Essayez de changer le nom de votre méthode.

ex :

```
class A {
    int b;
}
class B extends A {
    void b(){};
}
```

3.3.23 Access to protected field

Le champ est déclaré "protected" et la classe courante n'est pas une sous classe d'une classe où le champ a été déclaré.

ex :

```
class A {
    protected int a;
    void method (int c){}
}
{
    A b = new A();
```

```
    int c = b.a;          // le champ a est protégé
}
```

3.3.24 Impossible d'appeler return lorsque le type est void

Le type de retour de la méthode est de type void, elle ne devrait rien retourner.
ex :

```
class A {
    int a;
    void method (){
        return this.a;
    }
}
```

3.3.25 Impossible d'appeler this dans main

Dans le main on ne déclare pas de champs, et le main n'a pas de super class. Par conséquent, vous ne pouvez pas faire appel à « this » dans le main.
ex :

```
{
    int a;
    this.a;
}
```

3.3.26 can't perform an instance of

L'expression à droite doit être de type class ou null et le type à gauche est necessairement de type class.
ex :

```
{
    int a;
    if( a instanceof int ) {        // int n'est pas une class
        println(x);
    }
}
```

3.4 Erreurs dans la génération du code

3.4.1 Erreur : valeur entree ne correspond pas à un entier codable

Vous avez saisie un entier qui n'est pas codable sur 32 bits
ex :

```
{
    int x = readInt();          // vous entrer un entier ¿ 2 147 483 646 par exemple 3 000 000 000
    println(x);
}
```

3.4.2 Erreur : valeur entree ne correspond pas à un flottant codable

Vous avez saisie un flottant qui n'est pas codable sur 32 bits
ex :

```
{
    int x = readFloat();          // vous entrer un flottant très grand
    println(x);
}
```

3.4.3 Erreur : Division entiere par 0

Cette erreur s'affiche lorsque vous essayez de diviser par l'entier 0, ou de calculer le reste de la division par 0.

ex 1 :

```
{
    int x = 5/0;
}
```

ex 2 :

```
{
    int x = 5%0;
}
```

3.4.4 Erreur : debordement arithmetique $-->$ non codable ou division par 0.0

Cette erreur s'affiche lorsque vous essayez de diviser par un flottant de valeur 0.0, ou quand les flottants ne sont pas codable sur 32bits.

ex :

```
{
    float x = 5.0/0.0;
}
```

3.4.5 err_stack_overflow, Erreur : la pile est pleine

Cette erreur s'affiche lorsqu'il y a un débordement de la pile.

3.4.6 Erreur : allocation impossible, tas plein

Cette erreur s'affiche lorsqu'il y a un débordement du tas.

3.4.7 Erreur : conversion de type impossible

La convertion entre ces types ne peut pas être effectuée.

3.4.8 déréférencement de null

Vous faites un appel de methode sur une instance qui n'est pas été alloué dynamiquement (dans le tas) ;

3.4.9 Erreur de lecture

Pour readInt() vous devez saisir un entier.
Pour readFloat() vous devez saisir un flottant.

ex 1 :

```
{
    int x = readInt();          // si vous rentrer : true, vous aurrez une erreur
}
```

ex 2 :

```
{
    int x = readFloat();          // si vous rentrer : false, vous aurez une erreur
}
```

3.4.10 Erreur : Accès à des variables non initialisées

Vous essayer d'accéder à une variable qui n'a pas été initialisée.

3.5 Erreurs internes

Il s'agit là des éventuelles erreurs internes du compilateur. Si jamais vous rencontrer une des erreurs suivantes, sachez que ce n'est pas de votre faute, merci d'en informer les constructeurs en envoyant le message d'erreur ainsi que le ou les tests sur lesquelles vous l'avez reçu.

3.5.1 Identifier $< Identifier'sName >$ has no attached Definition

3.5.2 Identifier $< class'sName >$ is not a class identifier, you can't call getClassDefinition on it

3.5.3 Identifier $< Method'sName >$ is not a method identifier, you can't call getMethodDefinition on it

3.5.4 Identifier $< Field'sName >$ is not a field identifier, you can't call getFieldDefinition on it

3.5.5 Identifier $< varaible'sName >$ is not a variable identifier, you can't call getVariableDefinition on it

3.5.6 Identifier $< expression'sName >$ is not a exp identifier, you can't call getExpDefinition on it

3.5.7 Cannot print expression

3.5.8 Tree $< ClassName >$ has no location set

3.5.9 Parsing cancelled

3.5.10 Expression ... has no Type decoration

3.5.11 Expression cannot be printed

3.5.12 Shouldn't be initialized

3.5.13 Cannot load the expression

3.5.14 cannot be used for boolean expressions

3.5.15 Operandes pour la division non valide

3.5.16 Error in parsing

3.5.17 Comparaison operator shouldn't be parsed

4 Le mode opératoire pour utiliser vos extensions (options de la commande decac, configurations à utiliser etc.)

Voici la commande pour les options decac [[-p — -v] [-n] [-r X] [-d]* [-P] [-arm] ¡fichier deca¿...] — [-b]
On a pas tenu compte lors du parsing de l'ordre. Veuillez au maximum l'ordre de la commande ci-dessus
NB : L'option -d n'a pas été implémentée.

4.1 -p : (parse) décompilation

La décompilation vous permet d'avoir un programme deca similaire ou équivalent à celui que vous mettez en entrée. Il vous permet de vérifier votre code deca. Si vous décider de redécompiler le résultat obtenu, vous trouverez un code deca identique. Il s'agit de l'idempotence de la double décompilation. Pour pouvoir lancer votre décompilation sur un fichier.deca, il suffit de vous placer dans le répertoire gl24 et d'écrire la commande suivante :

4.2 -P : (Parallèlisme)

Permet de compiler plusieurs fichier_test deca simultanément (en parallèle) pour accélérer la compilation.

4.3 -b : (banner)

Affiche une bannière indiquant le nom de l'équipe : ******* team gl24 *******
NB : L'option -b doit etre appeler sans source file et sans options

4.4 -v : (vérification)

Arrête decac après l'étape de vérifications (ne produit aucune sortie en l'absence d'erreur)

4.5 -n : (no check)

Supprime les tests à l'exécution spécifiés dans 3.4

4.6 -r X : (registers)

Limite les registres banalisés disponibles à R0 ... RX-1, avec 4≤X≤16
NB : L'option -r doit etre suivi d'un nombre entre 4 et 16

4.7 -arm : (extension ARM)

Il exite une autre option concernant l'extension. Le compilateur dispose d'une option -arm qui permet de compiler vers une architecture ARM 32 bits.
decac -arm < $fichier.deca$ > permet de générer l'assembleur adéquat pour cette architecture

Une solution pour exécuter l'exécutable ARM32 est d'utiliser une machine Linux avec l'émulateur Qemu, voici les commandes pour l'installer :

- sudo apt install gcc-arm-linux-gnueabi
- sudo apt install qemu-user

Puis pour lancer l'execution, nous avons fournis le script run_arm : **./run_arm** < $executable$ >

NB : L'option -p et -v ne sont pas compatibles
NB : si vous faites rentrer une autre option que celles mentionnés si dessus un message d'erreur apparaitra : "L'option ... is incorrect"

5 Les limitations de vos extensions

Le sous-langage deca supporté par le compilateur est le sous langage sans objet. La manipulation des flottants nécessite une unité de calcul en virgule flottante (FPU) ce qui n'est pas géré par Qemu. Ces manipulations ne sont pas fournis dans cette version.
Concernant la limitation des registres pour l'architecture ARM, elle n'est pas implémentée. Puisque ils ne sont pas assez testé, l'affichage des entiers et les operations (a part l'addition) ne sont pas fournis dans cette version.