MANUEL UTILISATEUR

Projet GL Equipe 16

Ange Romuald Ossohou

Ait Nadir Lahmouch

Hamza Benjelloun

Oussama Fennane

Younes Zaibila

27 janvier 2020

Cette documentation est un guide qui permettra à l'utilisateur de comprendre le fonctionnement de notre compilateur. Notre compilateur est un compilateur pour un sous-langage de JAVA à savoir le langage de programmation Deca. Ce manuel explique les limitations du compilateur liées à son implémentation, la signification des différents messages d'erreurs, le fonctionnement et les limites de notre extension. Elle explique bien évidemment le mode opératoire pour utiliser l'extension et le compilateur.

I - Mode d'utilisation et description des fonctionnalités du compilateur

I - 1 - Mes premiers pas avec le compilateur Deca

Deca est un langage compilé et exécuté comme sa mère Java. Ainsi pour compiler un fichier source .deca, on utilise la commande en ligne **decac** (en analogie avec **javac**), suite à cela on produit un fichier exécutable .ass qui sera exécuté par la machine abstraite **IMA**, grâce à la ligne de commande **ima**.

Exemple d'utilisation:

> decac toto.deca : génère un fichier toto.ass.

> ima toto.ass : exécute le fichier le programme.

I - 2 - Options du compilateur et syntaxe d'utilisation

La syntaxe d'utilisation de l'exécutable decac est la suivante:

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

Les 7 options proposées par le compilateur sont les suivantes :

- -b : affiche la bannière de l'équipe indiquant le nom des membres de l'équipe.
- -p : arrête decac après la construction de l'arbre, affiche la décompilation de ce dernier.
- -v : arrête decac après l'étape de vérifications (ne produit aucune sortie en l'absence d'erreur).
- -n : supprime les tests de débordement à l'exécution (débordement de pile , arithmétiques et déréférencement null).
- -r X : limite les registres banalisés disponibles à R0 … R{X-1} , avec 4 <= X <= 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, lance la compilation des fichiers en parallèle (pour accélérer la compilation).

NB:

- Les options -v et -p sont incompatibles (ie elles ne peuvent pas s'exécuter simultanément).

- decac sans arguments affiche les 7 options du compilateur.

I - 3- Fonctionnalités du compilateur

Notre compilateur implémente les fonctionnalités du langage Deca, de l'affichage d'une simple suite de caractère comme le fameux "Hello Word", à la gestion des classes pour la programmation orientée objet (POO) en passant par la quasi totalité des fonctionnalités en programmation sans objet (programmation impérative) à savoir les boucles while, structures conditionnelles, séquences d'affectations et d'instruction).

NB: notre langage ne nous permet pas d'implémenter des boucles for.

II - Limitations du compilateur

La plus grosse partie des erreurs est liée à une gestion des registres pas très avancée. Et cela apparaît dans l'exécution de code comportant plusieurs appels de méthodes imbriquées ou des fonctions récursives très poussées.

En effet, il se peut que des registres comportant des informations nécessaire pour la suite de l'exécution soient écrasées ou remplacées, ce qui peut altérer le résultat final. Aussi, une autre limitation est due à la gestion d'erreur en l'occurrence celle des overflows qui peut passer entre les mailles des codes normalement invalides.

III - Message d'erreurs

L'utilisateur trouvera ci-joint une liste exhaustive des erreurs lors de l'utilisation du compilateur

v and p options are incompatible:

apparaît quand on utilise à la fois l'option p et v.

Exemple:

decac -p -v filename

The integer is too large:

quand on affecte un int très grand à une variable (levée par le parseur).

Exemple:

{

```
        int x = 99999999999999999999999999999999;
    }
```

The float is too large:

quand on affecte un nombre flottant très grand à une variable (levée par le parseur).

Exemple:

```
    {
        float f = 99999999999999999999999999999999999.9999;
    }
```

Symbol is already declared in Environment Exp:

quand on déclare deux variables avec le même nom dans le même Body.

Exemple:

```
    {
        boolean x;
        int x;
    }
```

Symbol is already declared in Environment Type:

quand on déclare deux classes avec le même nom.

Exemple:

```
    class Point {
        int x = 0;
        int y = 0;
        int z = 0;
    }
    class Point {
        int x = 0;
        int y = 0;
    }
```

Incompatible type :

quand on affecte à une variable une autre variable de type différent (hormis quand on donne un int à un float où la conversion devient implicite).

Exemple:

```
{
        int x = 0;
        boolean y = false;
        x = y;
}
```

Unsupported Operation :

apparaît quand une opération arithmétique entre deux expressions qui ne sont pas des nombres est présente.

Exemple :

```
{
        int x = 0;
        boolean z = true;
        println(z + x);
}
```

Return type is void :

quand on ne retourne rien dans une méthode.

Exemple:

```
class A{
        int x = 0;
        int getx(){
                return;
        }
}
```

<Class1> must be a subClass of <Class2> :

quand on affecte a un variable de class1 un autre variable de class2 qui n'est pas une sous class de class1.

Exemple:

```
class A {
        int x;
}
class B {
        int y;
}
{
```

```
        A a = new A();
        B b = new B();
        a = b;
    }
```

<_> is not declared :

quand on utilise une variable non déclarée.

Exemple:

```
    {
        boolean f = false;
        float x = 0.0;
        x = y + 6;
    }
```

Undeclared method :

quand on utilise une méthode non déclarée au préalable dans une classe.

Exemple:

```
    class A {
        protected int x = 0;
    }
    {
        A a = new A();
        println(“a = ”, a.getX());
    }
```

the field <_> is not declared :

quand on sélectionne un champ non déclaré dans une classe.

Exemple:

```
    class A{
        int x;
    }
    {
        A a = new A();
        println(“y = ”, a.y);
    }
```

_ incompatible déclaration type:

quand on déclare une variable avec un type qui n'existe pas dans Environnement type.

Exemple:

```
{
    A a = null;
}
```

InstanceOf can only be used with Classes:

quand on utilise InstanceOf avec une variable qui n'est pas de type class.

Exemple:

```
class A {
    int x;
}
{
    int z;
    if(z instanceof A){
    println(" z est de type A");
}
}
```

Impossible Cast :

quand on cast une variable de type class1 en un autre type qui n'est pas une super class de class1.

Exemple:

```
class A {
    int x;
    int y;
}
class B extends A {
    int z;
}
```

```
{
        A a = new A();
        B b = new B();
        b = (B) a;
}
```

<_> must be a class :

quand on appelle une méthode sur un variable qui n'est pas de type class.

Exemple:

```
class A {
        int x;
        int y;
        int getX() {
                return this.x;
        }
}
{
A a  = new A();
int z = 5;
println("x =  " z.getX());
}
```

<_> must be a method :

quand on appelle une méthode non définie sur une variable de type class.

Exemple:

```
class A{
        int x;
}
{
        A a = new A();
        println(a.getX());
}
```

Bad signature for the method <_> :

quand on utilise une méthode avec une mauvaise signature.

Exemple:

```
class A {
        int x;
        void getNumber(int x, int y){
        return x + y;
        }
}

{
        A a = new A();
        println(a.getNumber(0, false));
}
```

<> Is not a class :

quand on fait new A() et A n'étant pas une classe.

Exemple:

```
class A {
        }
{
        A a = new B();
}
```

type must be int for modulo :

modulo nécessite que les deux opérandes soient des int.

Exemple:

```
{
        println( 5 % 3.4);
}
```

cannot use this in main:

quand on utilise this dans le program principal

Exemple:

```
{
        this.x = 0;
```

```
}
```

<_> is not a class:

sélection d'un field d' une variable qui n'est pas de type class.

Exemple:

```
{
    int x = 0;
    println(x.a);
}
```

field is protected :

quand on sélectionne un field protected dans une class.

Exemple:

```
class A{
    protected int x = 0;
}
{
A a = new A();
println(a.x);
}
```

incompatible initialization :

lorsqu'on initialise une variable avec une valeur de type différent.

exemple:

```
{
    int x = false;
}
```

_ must be a subClass of _ : lorsqu'on initialise une variable de type class2 avec une autre variable de type class1 qui n'est pas une superclass de class2.

exemple:

```
class A{
    int x;
 }
class B extends A{
}
{
    B a = new A();
}
```

incompatible type (type must be an int or a float):

quand on fait minus d'une expression qui est de type différent de int ou float.

Exemple:

```
{
        boolean x = false;
        x = -x;
}
```

incompatible print expression type :

lorsqu'on print une expression qui n'est pas de type int, float ou String.

RedefinitionError: _ incorrect signature :

lorsqu'on rédéfinit une méthode avec une mauvaise signature .

exemple :

```
class A{
        int x;
        int getX(int r){
        return x  + r;
}
class B extends A{
        int getX(float r){
        return x + r;
}
```

RedefinitionError: typeMethod must be _: lorsqu'on redéfinit une méthode en changeant le type de retour .

exemple:

```
class A{
        int x;
        int getX(int r){
        return x  + r;
}
class B extends A{
        float getX(int r){
        return x + r;
}
```

IV - La bibliothèque standard (extension)

Pour nos utilisateurs, nous avons opté pour l'implémentation d'une extension **TRIGO** qui comprend les fonctions usuelles trigonométriques à savoir ulp, cosinus, sinus, arcsin et enfin arctan. Le fichier **Math.decah** qui se trouve *src/extension* correspond à l'extension.

IV - 1 - Fonctionnement de l'extension

Comme détaillé plus haut, notre bibliothèque implémente des fonctions trigonométriques sur des flottants. Mais l'implémentation de ces fonctions ont nécessité l'implémentation de fonctions tels que reduce (une fonction permettant de faire le modulo des angles), puissance (comme son nom l'indique, elle permet de calculer les puissances), atanNear0 (permettant le calcul de l'arctan pour des valeurs proches de 0), atanPositive et asinPositive (permettent de gérer la parité des fonctions asin et atan) …

L'utilisation de l'extension se fait en incluant le fichier Math.decah dans le fichier deca dans lequel on aura besoin d'utiliser ces différentes méthodes.

Exemple d'utilisation:

Calcul du arctan d'un angle et de son affichage dans un main.

```
// début du code


 #include "Math.decah"

{

Math m  = new Math();

println(" atan(0.0) =  ", m.atan(0.0));

}

// résultat attendu 0.0

// fin du code
```

vous pouvez trouver le code source de cet exemple dans *src/extension/expUtilisation.deca*

NB: La fonction asin ne prend qu'en paramètre un flottant compris entre -1 et 1 (inclus).

IV - 2 - Limitation de l'extension

IV - 2 - 1 Méthode Ulp

Notre méthode ulp donne des résultats identiques à celui de java. Nous pouvons le constater avec le tableau ci-dessous.

Intervalles	Nombre d'erreurs	Erreur max (en ulp)	pas de la boucle	Nombres de tests
[0 , 1]	0	0	pow(2, -14)	16 384
[1 000, 10 000]	0	0	pow(2, -7)	1 152 128
[10 000, 100 000]	0	0	pow(2, -7)	1 150 128

NB:

- La fonction pow est la fonction puissance
- L'erreur en un point x est pris en compte lorsque la différence absolue entre notre ulp et celui de java est supérieure à notre ulp en ce point

IV - 2 - 2 Méthode Cos

Intervalles	Nombre d'erreurs	Erreur relative moyen	Erreur moyen max	pas de la boucle	Nombre de test
[0.0, pi]	1663	-3.963071E-8	-3.0403606E-7	pow(2, -10)	3207
[pi, 100 pi]	2415	4.147462E-5	6.931883E-4	pow(2, -3)	2489
[100 pi , 100 000]	3115	0.015910711	-5.668972	pow(2, 5)	3116

IV - 2 - 3 Méthode Sin

Intervalles	Nombre d'erreurs	Erreur relative moyen	Erreur moyen max	pas de la boucle	Nombre de test
[0.1, pi]	1974	2.5990856E-7	2.3482336E-7	pow(2, -10)	3207
[pi, 100 pi]	1215	2.0544464E-4	2.1214543E-4	pow(2, -3)	1245
[100 pi , 100 000]	3116	7.686560E-4	-0.3116466	pow(2, 5)	3116

Au vue de la table des erreurs, on constate qu'on a une précision à 6 chiffres au voisinage de zéro, et pour des valeurs très grande on a une perte de précision.

IV - 2 - 4 Méthode Asin

Intervalles	Nombre d'erreurs	Erreur relative moyen	Erreur moyen max	pas de la boucle	Nombre de test
[0.0 , 0.3]	711400	1.1069499E-7	4.9364695E-7	pow(2,-22)	1 258 292
[0.3, 0.6]	74312	6.9339506E-7	3.4045765E-6	pow(2, -18)	78 644
[0.6, 1.0]	354	1.7692203E-6	5.9919876E-6	pow(2, -10)	410

On a utilisé ici l'erreur relative pour le calcul d'erreur, on peut donc garantir aux utilisateurs une précision de l'ordre de 10E-6.

IV - 2 - 5 Méthode Atan

Intervalles	Nombre d'erreurs	Erreur moyenne (en ulp)	Erreur max (en ulp)	pas de la boucle	Nombres de tests
[0, 10]	86526	2.5658689295	21.0	pow(2, -14)	163 841
[10, 10 000]	468787	9.946524E-4	1	pow(2, -7)	1 278 721
[10 000, 1 000 000]	715970	6.8431956386E-6	1	pow(2,-1)	1 980 001

On peut constater que pour de très grande valeur, notre fonction arctan est précise (avec pour erreur max égale exactement à 1 ulp) par contre pour des valeurs proches de 0 du fait de la très faible valeur de ulp en ces points, on a une erreur max de l'ordre de 21 ulp (mais on peut garantir qu'on a une précision de l'ordre de 10E-6).