ENSIMAG

Grenoble INP

Manuel Utilisateur

Projet GL

Groupe gl 49

Valentin OLLIER

Victor SAUNIER

Sofien CHENGUEL

Wael FEGUIRI

Hamza HASSOUN

Janvier 2020

Table des matières

1 Introduction

Ce document est un manuel ayant pour dessein de guider l'utilisateur pour utiliser notre compilateur répondant aux critères fournis dans le cahier de charges.
Une spécification des limitations du compilateur ainsi que des exemples d'erreurs retournées pour des programmes mal typés sera aussi détaillée.

2 Utilisation du compiltaeur decac

Le compilateur est lancé via la commande decac [**options**] [**fichierssources**].
Les fichiers sources doivent obligatoirement être des fichiers deca donc avec le suffixe **".deca"**.
Le lancement du compilateur produit un fichier assembleur .ass contenant la génération de code assembleur du programme deca compilé par l'utilisateur.
Le résultat de ce fichier assembleur peut ensuite être testé via l'interpréteur ima en lançant la commande [**ima**] [**fichiersource.ass**] pour évaluer le résultat de notre programme.
Les options du compilateur seront détaillés dans la section suivante.

3 Listes de options du compilateur

3.1 -b : Banner

Affichage de la bannière indiquant le nom de l'équipe.

3.2 -p : Parse

Arrêt de decac après l'étape de construction de l'arbre, et affichage de sa décompilation.
En présence d'un seul fichier source à compiler, la sortie doit être un programme deca syntaxiquement correct

3.3 -v : Verification

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

3.4 -n : No check (Option non implémentée dans notre compilateur)

Suppression des tests de débordement à l'exécution :
- Débordement arithmétique
- Débordement mémoire
- Déréférencement de null

3.5 -r X : Registers

Limite les registres banalisés disponibles àR0 ... RX-1, avec X variant de 4 à 16

3.6 -d : Debug

Active les traces de debug. Répéter l'option plusieurs fois pour avoir plus de traces

3.7 -p : Parallel (Option non implémentée dans notre compilateur)

Permet la compilation de plusieurs fichiers source en parallèle.

4 Caractéristiques et Limitations du compilateur

4.1 Analyse lexicale et syntaxique

Limitations des entiers et flottants :
-Pour les entiers , le compilateur ne peut prendre en entrée que les entiers représentés sur 32 bits donc entre -2^{31} et 2^{31-1} .
-Pour les flottants , Les littéraux flottants sont convertis en arrondissant si besoin au flottant IEEE-754 simple précision le plus proche.

- Une division flottante par 0.0 provoque une erreur.

4.2 Analyse contextuelle

-Lors de l'analyse d'un code deca le comportement du compilateur du point de vue d'une analyse contextuelle respecte les 3 passes suivants :

4.2.1 Passe 1 : Vérification des noms de classe

Lors de la passe 1, on vérifie le nom des classes et la hiérarchie de classes.
Cette passe récupère donc le nom des classes pour pouvoir les reconnaître lors de l'analyse de leurs corps.

4.2.2 Passe 2 : Vérification des déclarations de champs et méthodes

Lors de la passe 2, on vérifie les champs et la signature des méthodes des différentes classes. On hérite de l'environnement construit au cours de la passe 1. Si une classe hérite d'une superclasse alors la redéfinition d'une méthode dans la classe fille est telle que les signatures de la méthode dans la superclasse et la classe qui hérite soient identiques.

4.2.3 Passe 3 : Vérification des méthodes

-Vérification de la définition des variables.
-Vérification du corps des méthodes

Limitations contextuelles sur notre compilateur :
-return int , return float , return boolean passent et ne renvoient pas d'erreurs.
-La sélection direct des champs d'une méthode est possible sans instancier avec New la classe dans le main.

4.3 Génération de code

-La partie de génération de code à travers la commande **decac** permet la génération du fichier assembleur comme expliqué précédemment. Des erreurs sont retournés dans le cas de débordement de pile , de lecture erronnée d'entiers ou flottants dépassant l'espace mémoire disponible , d'allocation impossible ou encore d'une division entière ou flottante par zéro.
-La modification de champs est mal implémentée dans cette partie du compilateur et ne

fonctionne pas correctement(On peut déclarer une classe avec des champs mais on ne peut pas apporter des modifications ces champs par la suite.
-La génération de code pour le InstanceOf n'est pas implémentée dans ce compilateur.

5 Exemples de messages d'erreurs et configurations incorrectes

5.1 Appel à une méthode avec une mauvaise signature

Ici en initialisant "a" sans faire appel à New l'appel de notre appel à sa méthode devient incorrect.
"On ne peut appeler une méthode que d'une classe définie" est retourné en message d'erreur.
Ceci est un exemple d'implémentation :

```
class A { void methode () { } }
{
A a ;
a.methode(5) ;
}
```

5.2 Cast en une classe non définie

Ici le cast par exemple d'un entier en une classe est impossible
"No viable alternative at input 'obj' " est retourné en message d'erreur.
Ceci est un exemple d'implémentation :

```
{
int obj ;
obj = (classe) obj ;
}
```

5.3 Déclaration des champs

5.3.1 Déclaration d'un champ déjà déclaré

Ici on ne peut pas dans le champs des attributs avoir deux variables ayant le même nom.
"Un champ ou une méthode a déjà le même nom dans cette classe " est retourné en message d'erreur.
Ceci est un exemple d'implémentation :

```
class A {
int a ;
boolean a ;
}
```

5.3.2 Déclaration d'un type non reconnu

Ici on ne peut pas déclarer des variables de types non reconnus par notre compilateur à savoir int, float et boolean. Notons que le type String aussi n'est pas reconnu par le language deca
"Le type du champ n'est pas défini " est retourné en message d'erreur.
Ceci est un exemple d'implémentation :

```
class A
```

{
champ a ; }

5.3.3 Déclaration d'un champ de type void ou null

Ici on ne peut pas déclarer des variables de types null ou void.
"On ne peut pas avoir de champ de type null ou void " est retourné en message d'erreur.
Ceci est un exemple d'implémentation :
class A
{
void a ; }

5.4 Utilisation de New avec un type déjà reconnu par Deca

Ici on ne peut pas appliquer New à int ou float ou boolean. Elle ne peut pas non plus être utilisée pour des noms de nouvelles variables de types cités ci-dessus.
"On ne peut déclarer que des classes avec New " est retourné en message d'erreur.
Ceci est un exemple d'implémentation :
{
int n = new int() ; }

5.5 Utilisation de return

5.5.1 Retourner une méthode

On ne peut pas appliquer return pour une méthode.
"On ne peut déclarer que des classes avec New " est retourné en message d'erreur.
Ceci est un exemple d'implémentation :
class A { }
class B {
A methode() {
return A ; }
}

5.5.2 Retourner un type différent du type de la méthode

Return doit retourner le même type de la méthode définie.
"Le type renvoyé en pratique doit correspondre à un sous-type du type attendu " est retourné en message d'erreur.
Ceci est un exemple d'implémentation :
class A {
boolean methode() {
return 2.5 ; }
}

5.5.3 Utilisation de return dans le main

On ne peut pas appliquer return dans le main ou encore dans une classe qui doit renvoyer void.
"On ne peut pas faire de return dans le main ou dans une classe qui doit renvoyer void " est retourné en message d'erreur.
Ceci est un exemple d'implémentation :
class A { }

{
return 2 ;
}

5.5.4 Retourner une sous-classe

On ne peut pas appliquer return pour renvoyer une clase ou encore une sous-classe qui hérite d'une classe mère déjà définie.
"Variable non définie " est retourné en message d'erreur.
Ceci est un exemple d'implémentation :
class surclass { }
class sousclass extends surclass {

sousclasse methode() {
surclass retout = new surclass() ;
return retour ; }

5.5.5 Retourner un type

On ne peut pas appliquer return et renvoyer int ou float ou boolean.
"Ne doit pas passer " est retourné en message d'erreur.
Ceci est un exemple d'implémentation :
class A
{
int methode() {
return int ; }
}

5.6 Utilisation de Extends

5.6.1 Déclaration d'une classe qui hérite d'elle même

Une classe ne peut héritée que d'une autre classe déjà définie. Dans le cas contraire un message d'erreur sera renvoyé et cet héritage est donc impossible.
"La superclasse n'est pas définie " est retourné en message d'erreur.
Ceci est un exemple d'implémentation :

class A extends A { }
{ }

5.6.2 Déclaration d'une classe qui hérite d'une classe non définie

Une classe ne peut héritée que d'une autre classe déjà définie. Dans le cas contraire un message d'erreur sera renvoyé et cet héritage est donc impossible.
"La superclasse n'est pas définie " est retourné en message d'erreur.
Ceci est un exemple d'implémentation :

class A extends B { }
{ }

5.6.3 Déclaration d'une classe qui hérite d'un type

Une classe ne peut héritée que d'une autre classe déjà définie. Si cette classe de laquelle on veut hériter n'est pas réellement une classe un message d'erreur sera renvoyé

et cet héritage est donc impossible.
"La superclasse donnée n'est pas une classe " est retourné en message d'erreur.
Ceci est un exemple d'implémentation :

class A extends float { }
{ }

5.7 Déclaration des paramètres d'une méthode

5.7.1 Les deux paramètres ont le même nom

Dans la déclaration des paramètres d'une méthode il faut s'assurer qu'ils n'ont pas le même nom.
"Une autre variable déjà déclarée a le même nom " est retourné en message d'erreur.
Ceci est un exemple d'implémentation :

class A {
void methode (int a , boolean a) { }
}
{ }

5.7.2 Le paramètre a un nom inconnu

Dans la déclaration des paramètres d'une méthode il faut s'assurer que la variable soit d'un type reconnu par le language deca.
"Le type du paramètre n'est pas défini " est retourné en message d'erreur.
Ceci est un exemple d'implémentation :

class A {
void methode (inconnu a) { }
}
{ }

5.8 Déclaration des méthodes

5.8.1 Deux méthodes ayant le même nom

On ne peut pas avoir dans une classe deux méthodes avec le même nom.
"Une méthode a déjà le même nom dans cette classe. " est retourné en message d'erreur.
Ceci est un exemple d'implémentation :

class A {
int methode () { return 1 ; }

int methode () { return 2 ; }
}
{ }

5.8.2 Méthode de type non reconnu

On ne peut pas avoir dans une classe une méthode d'un type différent des types reconnus par le language deca (void,int ,boolean,float).

"Le type de retour de la méthode n'est pas défini " est retourné en message d'erreur.
Ceci est un exemple d'implémentation :
class A {
nondef methode () {}
}
{ }

5.9 Déclaration des classes

5.9.1 Déclaration d'un type en class

On ne peut pas déclarer des classes ayant pour nom un type défini du language deca à savoir boolean,int ou float.
"Une autre classe de même nom a déja été déclarée " est retourné en message d'erreur.
Ceci est un exemple d'implémentation :
class boolean { }
{ }

5.9.2 Déclaration d'une classe déjà déclarée

On ne peut pas déclarer deux classes ayant le même nom.
"Une autre classe de même nom a déja été définie " est retourné en message d'erreur.
Ceci est un exemple d'implémentation :
class A { }
class A { }
{ }

5.10 Utilisation de this

On ne peut pas utiliser this dans le main.
"this ne peut être appelé que dans une classe " est retourné en message d'erreur.
Ceci est un exemple d'implémentation :
{
float x = 8.6 ;
float y = this.b
}

5.11 Utilisation de la selection

5.11.1 Sélection à partir d'une classe non déclarée

Pour sélectionner une méthode ou un attribut d'une classe il est essentiel que cette classe soit déjà définie
"Variable non définie" est retourné en message d'erreur.
Ceci est un exemple d'implémentation :
{
boolean bool = A.b ;
}

5.11.2 Sélection sans instance de classe

Pour sélectionner une méthode ou un attribut d'une classe il est essentiel que cette classe soit déjà définie et qu'une instance de cette classe soit initialisée dans le main avec New.
"Ne doit pas passer" est retourné en message d'erreur.
Ceci est un exemple d'implémentation :

```
class A {
int a;
}
{
int b = A.a;
}
```

5.11.3 Sélection d'un champ non défini (attribut ou méthode)

Pour sélectionner une méthode ou un attribut d'une classe il est essentiel que ce champ soit initialement déclarée dans cette classe.
"Le champ n'est pas défini" est retourné en message d'erreur.
Ceci est un exemple d'implémentation :

```
class A {
void methode() { }
}
{ A a = new A();
float x = a.champ;
}
```

Ceci est un autre exemple d'implémentation :

```
class A {
int x; { }
}
{ A a = new A();
boolean bool = a.methode();
}
```

6 Annexe

6.1 Exemples d'insctructions en deca et des fichiers assembleurs fournis par le compilateur

int x ; println(0.5*(x*x)) ;

```
; start main program
; Main program
; Beginning of main instructions :
LOAD 0, R2
STORE R2, 1(GB)
WNL
HALT
; end main program
```

```
class A {
int n ;
int m ;
}
class B {
A a ;
int p ;
}
```

```
; start main program
; Début du programme
STORE R0, 1(GB)
STORE R0, 2(GB)
TSTO 3
BOV pile_pleine
ADDSP 3
; Mainprogram
HALT
; initialisationdeschampsdelaclasseA
init.A :
LOAD 0, R2
LOAD − 2(SP), R1
STORER 2, 1(R1)
LOAD 0, R3
LOAD − 2(SP), R1
STORER 3, 2(R1)
RTS
; initialisationdeschampsdelaclasseB
init.B :
LOAD null, R4
LOAD − 2(SP), R1
STORER 4, 1(R1)
LOAD 0, R5
LOAD − 2(SP), R1
STORER 5, 2(R1)
RTS
; MethodeEquals
code.Object.equals :
```

$PUSHR2$
$PUSHR3$
$LOAD-2(LB), R2$
$LOAD-3(LB), R3$
$CMPR2, R3$
$BEQtrue$
$ADD0, R0$
$BRAfin$
$true:$
$ADD0, R0$
$fin:$
$BRAfin.Object.equals$
$WSTR"Erreur : sortiedelaméthodeequalssansreturn"$
WNL
$ERROR$
$fin.Object.equals:$
$POPR3$
$POPR2$
RTS
$pile_pleine:$
$WSTR"Erreur : lapileestpleine"$
WNL
$ERROR$
$dereferencement_null:$
$WSTR"Erreur : dereferencementdenull"$
WNL
$ERROR$
$tas_plein:$
$WSTR"Erreur : allocationimpossible, tasplein"$
WNL
$ERROR$
$;endmainprogram$