Grenoble INP - Ensimag

Manuel Utilisateur De Decac

PROJET-GL GROUPE 49

BENTAIBI Oussama
AÏT HAMMOU Driss
AJJA Hamza
DAOUD YOUSSEF
LEACHOURI Khalil

2^{nd} Year - 25 janvier 2021

Table des matières

I Introduction

Java est un langage de programmation orientée objet créé par James Gosling et Patrick Naughton. Il est maintenant l'un des langages les plus adoptés dans le monde de la programmation. Plusieurs langages dérivés de java ont presque les mêmes propriétés que lui.
Deca est l'un de ces sous langages qui n'a que quelques différences de Java et comme tout autre langage, il a besoin d'un compilateur performant.
L'outil Decac est un compilateur du langage de programmation deca qui est codé par une équipe d'ingénieurs de l'ensimag. Il a pour but de transformer le code écrit en deca en un code assembleur qui peut être interprété par la machine abstraite IMA. Dans ce manuel, on va donner plus de spécifications sur ce compilateur, ses forces, ses faiblesses et surtout comment l'utiliser correctement.

II mode d'emploi

II.1 Entrée et sortie

Decac prend en entrée un ou plusieurs fichiers écrits en Deca avec l'extension ".deca", il génère ensuite en sortie des fichiers assembleurs avec une extension ".ass". Pour exécuter le code assembleur, on utilise l'interpréteur IMA .

II.2 Les options de Decac

II.2.1 L'option Banner : [-b]

Affiche une bannière indiquant le nom de l'équipe.

II.2.2 L'options parse : [-p]

Arrête decac après l'étape de construction de l'arbre, et affiche la décompilation de ce dernier (i.e. s'il n'y a qu'un fichier source à compiler, la sortie doit être un programme deca syntaxiquement correct).

II.2.3 L'option verification : [-v]

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

II.2.4 L'options no check : [-n]

Supprime les tests à l'exécution spécifiés dans les points 11.1 et 11.3 de la sémantique de Deca.

II.2.5 L'option registers : [-r X]

Limite les registres banalisés disponibles à R0 ... RX-1, avec 4 <= X <= 16.

II.2.6 L'option debug :[-d]

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

II.2.7 L'option parallel) : [-P]

S'il y a plusieurs fichiers sources, lance la compilation des fichiers en parallèle (pour accélérer la compilation).

II.3 Syntaxe

Voici la syntaxe qu'il faut écrire sur un terminal pour compiler un fichier Deca :
decac Options [fichier1 fichier2 fichier3....]

III Exemple simple

On va illustrer dans cet exemple les étapes qu'il faut suivre pour compiler un programme :

Voici un code en deca qui calcule la suite de Fibonacci :

```
1
2  // DESCRIPTION : Calcul du 10-ème terme de la suite de Fibonacci
3  // Resultat attendu : 55
4
5  {
6  int F0 = 0;
7  int F1 = 1;
8  int a;
9  int n = 2;
10       while (n<= 10){
11           a = F1;
12           F1 = F1 + F0;
13           F0 = a;
14           n = n + 1;
15           }
16        print(F1);
17     }
18
```

Si on écrit dans le terminal decac fibonacci.deca, on obtient le code assembleur suivant :

```
1  TSTO #4
2  BOV StackOverFlowException
3  ADDSP #2
4  ; start main program
5  ; Generation du tableau des variables
6  code.Object.equals:
7  LOAD #null, R0
8  STORE R0, 1(GB)
9  LOAD code.Object.equals, R0
10 STORE R0, 2(GB)
11 ; Main program:
12 ; Declaration des variables:
13 LOAD #0, R2
14 STORE R2, 2(GB)
15 LOAD #1, R2
16 STORE R2, 3(GB)
17 LOAD #2, R2
18 STORE R2, 5(GB)
19 ; Beginning of main instructions:
20 cond1:
21 LOAD 5(GB), R2
22 LOAD #10, R3
23 CMP R3, R2
24 SLE R2
25 CMP #0, R2
26 BEQ endWhile1
27 LOAD 3(GB), R3
28 STORE R3, 4(GB)
29 LOAD 3(GB), R3
30 LOAD 2(GB), R4
```

– code assembleur

```
31  ADD R4, R3
32  STORE R3, 3(GB)
33  LOAD 4(GB), R3
34  STORE R3, 2(GB)
35  LOAD 5(GB), R3
36  LOAD #1, R4
37  ADD R4, R3
38  STORE R3, 5(GB)
39  BRA cond1
40  endWhile1:
41  LOAD 3(GB), R1
42  WINT
43  HALT
44  DivisionByZeroException:
45  WSTR "DivisionByZeroException"
46  ERROR
47  IOException:
48  WSTR "IOException"
49  ERROR
50  StackOverFlowException:
51  WSTR "StackOverflowException"
52  ERROR
53  OverFlowException:
54  WSTR "OverFlowException"
55  ERROR
56  HeapOverFlowException:
57  WSTR "HeapOverFlowException"
58  ERROR
59  ; end main program
```

– code assembleur

Si on utilise par exemple IMA, on va obtenir le résultat attendu qui est 55 ;

IV Liste des erreurs

IV.1 Erreurs syntaxiques :

Illegal character : Aucun token ne correspond à l'entrée saisie par l'utilisateur. C'est-à-dire que l'utilisateur a fait entrer un caractère qui n'est pas permis.

Unterminated String : L'utilisateur a commencé l'écriture d'une chaîne de caractère, mais ne l'a pas finie en fermant les guillemets.

Circular include : Un fichier est inclus par lui-même. Include file not found : Le fichier entré n'existe pas.

Bad file name : Le nom du fichier entré pour l'Include n'est pas un nom de fichier valide selon la syntaxe de Deca.

Extraneous Input '...' expecting '...' : La suite de tokens produits par le lexer ne peut être analysée correctement par le parser, car le token trouvé ne correspond pas au token attendu.

Left-hand of assignement is not an Lvalue : La partie gauche de l'affectation n'est pas un identificateur.

No viable alternative at input : Le parser n'a pas trouvé de règle correspondant à l'entrée de l'utilisateur.

Bad comment : Un commentaire multiligne nom terminé.

IV.2 Erreurs Contextuels de decac :

IV.2.1 Erreurs Globales :

Identifier must be declared (0.1) : Un identificateur (de classe ou de variable par exemple) utilisé n'a pas été déclaré précédemment.

Cannot find type (0.2) : L'identificateur utilisé pour la déclaration d'une variable par exemple n'est pas défini.

IV.2.2 Erreurs d'affectations :

Right type is not a subtype of the left type! (3.28) : La classe à droite d'une affectation d'objets n'est pas une sous classe de celle à gauche de l'affectation.

Incompatible types : Les deux deux types de la RValue et la LValue ne correspondent pas.

Right type is null! (3.28) : Le type de la RValue est null.

IV.2.3 Opérations arithmétiques :

Types of arithmetic operations must be float and int! (3.33) : L'utilisateur a fait une opération arithmétique sur autres types que int et float.

Modulo types must be int! (3.33) : Les types des deux opérandes de l'opération modulo ne sont pas valides. Cette opération se fait entre deux expressions de type entier.

IV.2.4 Opérations de comparaisons :

Inequality is not available for non-numbers! (3.33) : L'utilisateur a tenté de faire une inégalité entre deux deux opérandes dont un d'eux n'est pas un float ou un int.

Incompatible types for comparaison! (3.33) : Les deux types de comparaisons ne peuvent pas être comparés.

IV.2.5 Opérations booléennes :

Binary boolean operation must be between booleans (3.33) : L'utilisateur a tenté d'exercer une opération booléenne sur des types non booléens.

Unary not is not defined for this type (3.38) : L'utilisateur a utilisé l'opérateur UnaryNot sur une expression non booléenne.

IV.2.6 Print :

Print argument is not printable! (3.31) : Les arguments de print doivent être des chaînes de caractères, des entiers ou des floats.

IV.2.7 If tehn Else :

Condition must be boolean! (3.29) : La condition d'une boucle while ou d'une exécution conditionnelle doit être un booléen. Les entiers 0 et 1 ne sont pas considérés comme des booléens.

IV.2.8 Déclaration des variables :

Instancied type must be a class! (3.42) : Le mot clé 'New' doit être utilisé sur des identificateur qui sont de type Class.

Illegal start of expression : void type inaccessible! (3.17) : La variable déclarée est de type void. Deca n'accepte pas les déclarations de variables de ce type.

Variable is already defined! (3.17) : L'identificateur utilisé pour la déclaration de la variable a été déjà utilisé dans l'environnement courant.

IV.2.9 Déclaration de classes :

Super class must exist! (1.3) : La classe hérite d'une classe qui n'existe pas.

Superclass identifier must be a class identifier! (1.3) : L'identificateur utilisé pour la superclasse n'est pas un identificateur de classe.

Class name is already used! (1.3) : Une autre classe avec le même nom a été déjà déclarée.

IV.2.10 Déclaration de champs et méthdes et paramètres :

Field type must not be void! (2.5) : Le champ déclaré est de type void. Deca n'accepte pas des champs de ce type.

Identifier already used for a method in super class (2.5) : Le nom du champ ne peut pas avoir le même nom qu'une méthode déclarée dans la classe mère.

field already declared! (2.5) : L'identificateur utilisé pour la déclaration du champ est déjà utilisé.

Redefined method must have the same signature of the inherited method! (2.7) : La méthode déclarée est donc une redéfinition de la méthode de la classe mère. Elles doivent donc avoir la même signature.

Redefined method type must be a subtype of the inherited method type! (2.7) : la méthode déclarée est une redéfinition d'une méthode de la classe mère. Les types de retour des deux méthodes doivent donc être compatibles.

Identifier already used! (2.6) : L'identificateur utilisé pour la méthode est déjà utilisé dans la classe courante (par une méthode ou un champ).

Parameter type can't be void! (2.9) : le paramètre déclaré est de type void. Deca n'accepte pas des paramètres de ce type.

Parameter is already used! (3.13) : Le paramètre est déjà déclaré dans l'environnement courant.

IV.2.11 Les appels de méthodes :

Method is not defined (3.71) : La méthode déclarée n'existe pas.

Object must be a class type! (3.71) : La Lvalue de l'appel de méthode doit être un objet de classe (this implicite également). les éléments de différents types comme les entiers n'ont pas de méthodes associées.

Identifier is not a method identifier! (3.72) : L'identificateur de l'appel de méthode doit être un identificateur de méthode.

Argument number doesn't match with method signature! (3.72) : la méthode appelée est une méthode déjà définie dans une classe. Elle doit donc avoir les mêmes types d'arguments que cette classe mère.

IV.2.12 Sélection :

Selection LeftValue expression must be a class! (3.65), (3.66) : La LValue doit être impérativement un objet de type Class. La sélection est donc invalide pour les autres types comme int et float.

Selection right identifier must be a class field! (3.70) : La sélection se fait juste sur les champs, donc la sélection sera invalide si ce n'est pas le cas.

Field is protected! (3.66) : On ne peut pas sélectionner ou accéder à un champ protégé dans le main ou les autres classes qui n'ont aucune relation avec la classe courante.

Expression type is not a subtype of the current class! (3.66) : Pour sélectionner un champ protégé, le type de la LValue doit être un sous-type de la classe courante.

IV.2.13 Cast :

Void type cannot be casted! (3.39) : Un élément de type void ne peut pas être converti.

Incompatible types for cast! (3.39) : Le type du cast et le type de l'expression ne sont pas compatibles.

IV.2.14 Autres :

Cannot use 'this' in main! (3.43) : Le mot clé 'this' fait référence à la classe courante, il peut pas être utilisé quand il s'agit d'un main (classe courante est null).

Void type cannot be returned! (3.24) : Une méthode de type de retour void ne doit retourner rien en fait. Donc, elle ne peut pas retourner un élément de type void.

Instanceof is used only on classes! (3.42) : InstanceOf ne s'applique que sur des classes. On ne peut pas faire un «InstanceOf int» par exemple.

InstanceOf is not correctly used! (3.40) : Cela veut dire que instanceof n'est pas utilisée correctement.

IV.2.15 Erreurs Liées à l'exécution :

StackOverflowException : On a un débordement de la pile.

IOException : Lors de la lecture d'un entier ou d'un flottant, le compilateur vérifie si cette valeur dépasse l'espace de stockage disponible ou non.

OverflowException : Une opération mathématique produit une valeur supérieure à la valeur maximale qu'on peut représenter dans l'espace de stockage disponible.

DivisionByeZeroException : Cette erreur se produit quand on a une division d'un réel ou d'un entier par 0.

HeapOverFlowException :

V Limitations du compilateur :

Le compilateur dans sa forme actuelle permet de vérifier contextuellement et syntaxiquement un programme implémentant le langage objet (les casts et instanceOf également). Cependant, il ne génère pas un code assembleur complet pour ce langage. En effet, le compilateur a encore quelques dysfonctionnement dans quelques parties.

Concernant les méthodes : Lors de la vérification du corps d'une fonction, le compilateur ne vérifie pas si une fonction avec une type de retour autre que void fait un return. Un paramètre qui a le même nom qu'un champ de la classe force le développeur à utiliser le mot clé 'this' pour accéder à ses champs. L'utilisateur peut également ajouter des instructions après avoir fait le return d'une fonction. La performance de return a été testé, mais il reste quelques cas ou elle ne marche pas comme il faut.

Concernant les casts : Les codes générateurs des casts ont été implémentés, mais ils doivent être débogués. Le compilateur ne génère donc pas un code assembleur correct pour ce genre d'opérations.

Concernant les opérations arithmétiques : Les tests qu'on a fait pour tester ces opérations ont parfaitement marché. Et pourtant, il nous reste un test de performance qui ne marche pas, donc bien évidement qu'il reste des problèmes pour ces opérations mais qu'on arrive pas à déterminer.

VI Bibliothèque de math.decah :

Le compilateur est fourni avec une bibliothèque des fonctions mathématiques et trigonométriques , cet ensemble des fonctions est implémenté dans : src/main/resources/include . La fonction ulp implémentée donne les mêmes résultats que la fonction de Java Math.ulp() , et l'aide de cette fonction qu'on a réalisé les tests de précision pour les autres fonctions trigonométriques en utilisant la formule d'erreur : erreur = (x-x')/ulp(x) où x' est la valeur approché , dans notre cas c'est la valeur donnée par nos fonctions trigonométriques et x est la valeur donnée par les fonctions de la librairie Math de Java.

Les fonctions cosinus et sinus sont implémentées à l'aide des approximations polynomiales (les séries de taylor + la méthode de Horner) qui nous donnent une bonne précision sur des intervalles au voisinage de 0 ,cependant pour des valeurs > Pi/4 par exemple , on rencontre des très grandes marges

d'erreur. Ainsi , on a utilisé l'algorithme de cody and Waite qui nous permet de faire la réduction d'un paramètre x quelconque et le remettre à l'intervalle où nos fonctions donnent les meilleurs résultats ,des erreurs importantes existent encore pour des multiples de Pi/2 et Pi .

La fonction arctangente est aussi implémenter à l'aide des séries de Taylor avec une bonne précision

Sur [-1,1] , et pour les valeurs supérieures à 1 on a utilisé une identité trigonométrique pour ramener le paramètre à cet intervalle de confiance, la fonction arcsinus a presque la même précision puisqu'on l'a implémenté à l'aide de la relation mathématique qui la relie à arctangente .

Autres fonctions mathématiques sont implémentées puisqu'elles sont indispensables pour implémenter les fonctions trigonométriques et la fonction ulp (la fonction exponent(x) qui a pour but extraire l'exposant d'un flottant donné en paramètre , la fonction abs(x) qui calcule la valeur absolue de x , la fonction sqrt(x) qui calcule la racine carré ...)

Les fichiers utilisés pour tester les fonctions mathématiques sur java se trouvent à :src/main/java/Extension . (cette répertoire contient aussi un fichier de l'implémentation et un test de la méthode cordic)

Plus de détails sur nos algorithmes seront explicités dans la documentation spécifique à la classe Math.