Documentation Utilisateur Projet GL

Groupe 44

COLIN-SOHY Mathéo

DIMITRIOU Tristan

EL HIT Oumayma

GAUX Antoine

HARRAUD Paul-Louis

Table des matières

1 Introduction

Ce document a pour but de présenter notre compilateur du point de vue de l'utilisateur. Il comportera les points suivants :

— L'utilisation du compilateur
— Les limites de l'utilisation
— Les messages d'erreurs renvoyés par notre compilateur
— L'utilisation de l'extension
— Les limites de l'extension

La construction de cette documentation est établie selon la version du compilateur récupérée lors du rendu final, à savoir le Lundi 24 Janvier 2022 à 16h.

2 Utilisation du compilateur

2.1 Utilisation simple

L'utilisation du compilateur est semblable à celle des compilateurs classiques. On suppose que l'utilisateur à correctement configuré la variable PATH propre à son Shell, afin de lancer le fichier exécutable lié au compilateur (*à savoir : src/main/bin/decac*) directement via la commande *decac*. Pour compiler un fichier nommé HelloWorld.deca, il faut exécuter la commande suivante (en supposant que le shell est ouvert dans un dossier contenant le programme deca HelloWorld.deca) :

```
decac <source files ...>
```

Il est alors produit un fichier HelloWorld.ass dans le meme répertoire, contenant les instructions assembleur traduites depuis le programme source deca. Cet extension *.ass* est exécutable via ima, à l'aide de la commande suivante :

```
ima out.ass
```

2.2 Les options du compilateur

```
decac -h
```

permet d'afficher le message d'aide contenant toute les options du compilateur.

```
decac -b
```

Affiche la bannière des auteurs du compilateur.

```
decac -d[ddd] <source files ...>
```

Active les messages de debogage, elle peut être répétée jusqu'à quatre fois pour augmenter la verbosité.
Note : Cette option est surtout utile pour le développeur, un peu moins pour l'utilisateur

```
decac -i
```

Utilisation du compilateur via l'entrée standard.
Cette option permet de saisir un programme deca directement dans le Shell ou par la sortie d'une autre commande. Le programme doit être terminé par un EOF (Crlt+D).
Le fichier assembleur est produit dans le répertoire temporaire du système.
Cette option est surtout utile au débogage et aux tests du compilateur.

```
decac -n <source files ...>
```

Désactive les vérifications a l'exécution.
Cette option permet au programme d'ignorer les erreurs suivantes :

- Division entière (et modulo) par zéro
- Débordement arithmétique sur les flottants (inclut la division flottante par 0.0)
- Absence de return lors de l'exécution d'une méthode
- Conversion de type impossible
- Déréférencement de null
- Débordement mémoire (pile ou tas)

```
decac -P [threads] [--] <source files ...>
Exemples : decac -P 2 HelloWorld.deca Hello.deca
           decac -P -- HelloWorld.deca Hello.deca
```

Compile plusieurs programmes en parallèle, et génère le code assembleur correspondant.
L'argument est optionnel.
Le nombre de thread par défaut est 4.
Chaque programme est compilé dans son propre thread, si le nombre de programmes est supérieur au nombre de threads, alors un thread pourra compiler plusieurs programmes, l'ordre est celui des arguments.

```
decac -p <source files ...>
```

Affiche le programme décompilé depuis l'arbre abstrait et s'arrête. (Étape A du processus de vérification)
L'indentation est les commentaires peuvent differer du fichier original.

```
decac -r <source files ...>
```

Définit le nombre de registre maximum à utiliser (Assembleur).
Les valeurs autorisées sont comprises entre 4 et 16 (4 est la valeur par défaut).

```
decac -v <source files ...>
```

Procède à la vérification du code source mais ne produit pas de code assembleur. (Étape A + Étape B du processus de compilation)

```
decac -w <source files ...>
```

Autorise l'apparition de WARNING durant la compilation (Erreurs n'empêchant pas la compilation)

3 Les limites d'utilisation

3.1 Limites contextuelles

Dans la version finale de notre compilateur, la vérification syntaxique effectue toutes les vérifications inhérente au langage essentiel de deca.

Les aspects qui différencient ce sous-ensemble de celui du langage complet, c'est-à-dire le mot clé instanceof et les cast d'un type vers l'autre, ne sont donc pas gérés et peuvent entraîner des erreurs qui ne seront alors pas différentes de celles présentes lors de l'import initial du projet.

On pourra remarquer que lors de cette passe contextuelle, contrairement au langage Java, une variable nommée *int* ou ayant le même nom qu'un type de façon générale, est autorisée et ne posera aucun problème pour la compilation et l'exécution du programme.

On pourra également remarquer l'ajout dans la plupart des fonctions de vérifications contextuelles, d'un paramètre *envTypes* qui permet de recenser les types Builtin ainsi que les types définis par le programme en

soi afin de toujours pouvoir y accéder simplement et de les séparer des variables globlaes qui seron définies ultérieurement.

Enfin, une attention particulière sera apportée à la définition de Labels, qui sont initialisées selon des règles citées dans la partie suivante, lors de la passe contextuelle sur les méthodes afin d'interdire les problèmes liés aux différents noms de méthodes et de classes qui pourraient être donnés par l'utilisateur du compilateur.

Pour le reste, les différentes passes du programme sur l'arbre syntaxique permettent d'effectuer les vérifications fournies par la documentation et de fournir en sortie l'arbre décoré qui permet de simplifier grandement la partie suivante de génération de code.

3.2 Génération de code

La compilation du langage deca s'arrête, comme pour la vérification contextuelle, au langage essentiel. On ne peut donc ni réaliser de cast, ni réaliser d'instanceof. Cela empêche donc de redéfinir la méthode equals de Object de manière pertinente, pour donner un exemple. Un moyen de contourner ce problème est de créer une autre méthode nommée par exemple $equals\_nom\_de\_l\_objet$. Etant donné que le langage deca n'utilise pas le == comme un $equals$, nous avons estimé que le manque était acceptable.

Concernant le langage essentiel, il faut cependant noter 2 limites principales connues.

La première est une gestion partielle de la sauvegarde des registres : la spécification n'est pas entièrement remplie, nous conseillons donc à l'utilisateur d'éviter d'imbriquer "trop" de méthodes. En particulier, il faut éviter d'instancier des objets dans l'instanciation d'un autre objet, et préférer le faire en deux temps, avec une méthode type init.
La seconde limite provient de la gestion du dépassement de pile. On suit l'évolution de la pile d'une manière simpliste, qui fait que l'on ne détecte pas certains dépassement de pile. Pour donner des exemples, un programme qui surcharge la pile en déclarant trop de variables sera détecté, cependant un programme qui surcharge la pile via une boucle while infinie ne le sera pas.

On notera comment sont gérés les noms de Labels. En effet, un point important de la génération de code est de garantir l'unicité des Labels. On garantit cette unicité en associant à chaque "élément" qui le demande un identifiant. Ainsi, chaque instance de *DecacCompiler* génère des nombres uniques à une instance de ce compilateur. Ainsi, l'utilisateur n'a pas de limite autre que contextuelles pour nommer ses variables/classes/méthodes, on réutilise ce nom avec l'identifiant généré pour garantir l'unicité.

4 Messages d'erreurs renvoyés par le compilateur

4.1 Lexicales / Syntaxiques

Erreur	Causes	Message
Caractère / Expression inconnue	Caractère non reconnu par le Lexer	This expression is not recognized by the compiler
Mauvais arrondi d'un flottant	Le flottant tends vers l'infini ou zéro après avoir été arrondi	The rounding of the float tends towards zero
		The rounding of the float tends towards infinity
Flottant non reconnu	Flottant non considéré comme un nombre	This float is not a number (NaN)
Identificateur trop long	L'identificateur dépasse 512 caractères	The identifier is too long (max 512 char)
Entier impossible à coder sur 32 bits	L'entier ne peut être codé signé positif sur 32 bits	This integer cannot be encoded as a 32-bit signed integer
Elseif	En deca le Elseif n'existe pas (Else + If oui)	Missing space between else and if
Mauvaise syntaxe du include	#include <nom_du_fichier> non respecté	Incorrect #include syntax: #include \"filename.decah\"

FIGURE 1 – Liste des erreurs lexicales et syntaxiques

4.2 Contextuelles

[ContextualError 0.1.1]	Identifier non défini	Identifier '_name_' not defined
[ContextualError 0.2.1]	Identifier de type non défini	Type identifier '_name_' not declared
[ContextualError 0.2.2]	Déclaration de var/param/attribut de type string	Cannot declare string variable, field nor parameter
[ContextualError 1.3.1]	extends une classe non définie	Super class identifier _typeName_ unknown
[ContextualError 1.3.2]	extends un objet qui n'est pas une classe	Identifier _typeName_ must be class identifier
[ContextualError 1.3.3]	Redéfinition d'une classe	Type or class _typeName_ already defined
[ContextualError 2.4.1]	Redéfinition d'un attribut	Field _fieldName_ already defined
[ContextualError 2.5.1]	Attribut de type void	Field _fieldName_ cannot be of type void
[ContextualError 2.5.2]	Redéfinition d'un non attribut de la superclasse	Field _fieldName_ cannot be defined as not field in super class environment
[ContextualError 3.12.1]	Paramètres ayant le même nom	Parameter _paramName_ already declared
[ContextualError 3.17.1]	Variable de type void	Variable _varName_ cannot be of type void
[ContextualError 3.17.2]	Redéfinition d'une variable	Variable _varName_ already declared
[ContextualError 3.24.1]	Return dans une fonction return Vide	Cannot have a return in a void type function
[ContextualError 3.28.1]	Type de retour incompatible avec type de la fonction	Incompatible return type (expected _returnType_ got _typeReturned_)
[ContextualError 3.28.2]	Types rendent l'initialisation impossible	Incompatible initialization types (got _varType_ and _exprType_)
[ContextualError 3.29.1]	Condition non booléenne	Condition type must be boolean (got _exprType_)

[ContextualError 3.31.1]	Argument de print de type class	Cannot print ClassType
[ContextualError 3.31.2]	Argument de print de type booléen	Cannot print a boolean
[ContextualError 3.32.1]	Types rendent l'assignation impossible	Incompatible types for assignation (got _leftType_ and _rightType_)
[ContextualError 3.33.1]	Types rendent l'utilisation de l'opérateur impossible	Uncompatible types for _operator_ (got _leftType_ and _rightType_)
[ContextualError 3.33.2]	Conv Float sur un argument non entier	Uncompatible type for float conversion (expected int or float got _exprType_)
[ContextualError 3.42.1]	Types rendent l'utilisation d'un new impossible	Uncompatible new object type
[ContextualError 3.43.1]	Utilisation du this en dehors d'une classe	Cannot use 'this' outside class
[ContextualError 3.65.1]	Appel d'identifier sur un objet non class	Cannot call an identifier on a not class Type object
[ContextualError 3.65.2]	Appel d'un field introuvable sur la classe	Cannot find field _fieldName_ in class _className_
[ContextualError 3.65.3]	Appel d'un field qui n'est pas un field	Identifier _identName_ does not define a field
[ContextualError 3.66.1]	Appel d'un field protected inatteignable	Cannot call parent protected field from child class
[ContextualError 3.66.2]	Appel d'un field protected non atteignable	Cannot access protected field of child class
[ContextualError 3.71.1]	Appel de méthode sur un objet non class	Cannot call a method on a not class Type object
[ContextualError 3.71.2]	Appel d'une méthode introuvable sur la classe	Cannot find method _methodName_ on class _className_
[ContextualError 3.72.3]	Appel d'une méthode qui n'et pas une méthode	Identifier _methodName_ does not define a method
[ContextualError 3.73.1]	Trop d'arguments dans l'appel à la méthode	Too much arguments (expected _signSize_)
[ContextualError 3.73.2]	Pas assez d'arguments dans l'appel à la méthode	Not enough arguments (expected _signSize_ got _argsSize_)
[ContextualError 3.74.1]	Types incompatibles pour l'appel à la méthode	Uncompatible types for methodCall (expected _expectedType_ got _gotType_)

FIGURE 2 – Liste des erreurs contextuelles

[ExecutionError 0.1]	Entrée incorrect	Input error
[ExecutionError 1.1]	Operation impossible (division par zero, conversion imp	Arithmetic error
[ExecutionError 1.2]	Dereferencement d'une adresse null	Dereferencement error
[ExecutionError 1.3]	Pas de return dans une fonction de type de retour non-v	VoidError: no return instruction in non-void method
[ExecutionError 1.4]	Tas plein ou pas de place pour le mot de taille spécifié	Cannot allocate memory
[ExecutionError 1.5]	Pile pleine	StackOverflow

FIGURE 3 – Liste des erreurs d'exécution

5 Librairie Mathématique

5.1 Utilisation

L'extension choisie est : TRIGO.
Pour inclure la librairie il faut saisir au début d'un programme deca :

```
#include "Math.decah"
```

Dans le main il faut instancier la classe Math

```
{
    Math m = new Math();
    m.cos(1.0);
}
```

Les fonctions implémentées sont :

```
float cos(float x)
float sin(float x)
float asin(float x)
float atan(float x)
float ulp(float x)
float pow(float a,int b)
int abs(int x)
int sign(int x)
```

Les cinq premières sont celles demandées par la documentation, les trois suivantes sont utilisées dans celles-ci mais peuvent également être utiles pour l'utilisateur final d'où la mise à disposition de ces autres fonctions.

La fonction $pow(x, n)$ permet donc de calculer x^n

La fonction $abs(x)$ permet de calculer la valeur absolue de x

La fonction $sign(x)$ permet de retourner le signe de x (1 si $x >= 0$, -1 si $x < 0$)

Il existe des versions flottantes pour abs et sign, respectivement fpow et fsign.
Les fonctions trigonométriques sont approximées par l'algorithme Cordic (cos, sin, arcsin, arctan).
Cet algorithme permet de calculer des coordonnées et des angles par des rotations successives de vecteurs, c'est assimilable a une recherche dichotomique dans le cercle unitaire.
Quant à la fonction Ulp, on s'est appuyé sur la méthode Kahan.
(Suite a un problème de fusion, l'extension ne fonctionne que sur la branche *pl*)

5.2 Limites

cos, sin, arctan, arcsin retournent des approximations à 10^{-7} près des valeurs données par la librairie Numpy de Python, notre référence.
Les fonctions cos et sin fonctionnent uniquement dans l'intervalle $[-\pi; +\pi]$.

Le temps d'éxecution de 1000 cos ou sin est d'environ 8s (1700 instructions)
On en déduit que notre boucle CORDIC est capable d'environ 125 calculs par seconde.