Manuel Utilisateur

Equipe 10

25 Janvier 2021

Contents

1 Introduction **2**

2 Options **2**

3 Erreurs **2**
- 3.1 Lors de la compilation 2
- 3.2 Lors de l'exécution 5

4 Extension TRIGO **6**
- 4.1 Utilisations de l'extension 6
- 4.2 Limitations de l'extension 6

5 Limitations du compilateur **7**
- 5.1 Limite de tas 7
- 5.2 Absence de tableau 7
- 5.3 Optimisation possible du code assembleur 7

1 Introduction

Le compilateur est appelé avec la commande *decac*, prenant en argument un fichier d'extension .deca. Celui-ci va réaliser dans l'ordre les analyses lexicale, syntaxique et contextuelle. Une fois que la compilation est terminée, un fichier d'extension .ass de même nom que le fichier .deca entré est créé dans le même répertoire. Il contient la génération de code assembleur du programme deca. Ce fichier est ensuite exécuté par la commande *ima fichier.ass*

2 Options

Les options du compilateur sont :

- -b : *banner*, affiche une bannière avec le nom de l'équipe ainsi que les noms des membres la composant.
- -p : *parse*, le compilateur s'arrête à la construction de l'arbre et en affiche la décompilation.
- -v : *verification*, le compilateur s'arrête après les vérifications.
- -d : *debug*, active les traces de debug. Répéter l'option plusieurs fois pour avoir plus de traces.
- -n : *no check*, supprime les tests à l'exécution spécifiés dans les points 11.1 et 11.3 de la sémantique de Deca.
- -r X : *registers*, limite les registres banalisés disponibles à R0 ... RX-1, avec 4 <= X <= 16.
- -P : *parallel*, compile plusieurs fichiers en parallèle (s'il y en a plusieurs).

3 Erreurs

3.1 Lors de la compilation

Les erreurs possibles sont :

- **"Le type $TYPENAME n'est pas affichable (les types affichables sont int , float et string)"**, un type différent de float, int ou string est dans un print.
- **"les opérateurs doivent être de type numérique (int ou float)"**, une variable de type différent de int ou float est appelée dans une opération.
- **"les deux opérateurs doivent être de type boolean"**, un ou deux opérateurs non booléens sont appelés dans une opération booléenne.

- **"types non compatibles pour faire la comparaison"**, on essaie de comparer deux opérateurs de types différents et incompatibles.
- **"une variable ne peut pas être de type void"**, on essaie de déclarer une variable sans type ou avec un type void.
- **"double définition de la variable $VARNAME"**, on déclare deux fois une variable avec le même identifiant.
- **"identificateur non défini"**, on appelle un identificateur qui ne correspond pas à une variable.
- **"type non défini"**, le type d'une variable n'est pas défini.
- **"Modulo n'est applicable que sur les entiers"**, on essaie d'appliquer un modulo sur un opérateur non int.
- **"type boolean attendu"**, une expression de type différent de boolean est en argument d'un Not.
- **"unaryMinus est applicable sur les types numériques"**, une expression de type différent de float ou int est en argument d'un UnaryMinus (-).
- **"la condition doit être de type boolean"**, une condition n'est pas de type boolean.
- **"type incompatible"**, on essaie d'assigner à une variable une valeur de type incompatible avec celui de la variable.
- **"$CLASSNAME1 ne peut être cast en $CLASSNAME2"**, on essaie de faire un cast sur une classe qui n'est pas une sous classe de l'autre.
- **"un type class était attendu !"**, s'il s'agit d'une classe, il faut que le valeur droite de l'expression soit de type classe.
- **"types incompatibles pour faire la comparaison."**, on essaie de faire la comparaison entre deux types incompatibles (exemple : string < boolean).
- **"Le champ superClass doit être une classe."**, lorsqu'on associe à une classe une super classe de type différent de class.
- **"Super classe introuvable."**, si la super classe n'est pas définie.
- **"Une telle classe est déjà définie."**, on essaie de déclarer une classe déjà définie.
- **"type inexistant."**, lors d'une déclaration d'un champ avec un type indéfini.

- **"un champ ne peut pas être de type void."**, on essaie de déclarer un champs de type void ce qui est interdit.
- **"nom de champ déja utilisé."**, lors d'une déclaration d'un champ qui existe déjà dans la classe en question.
- **"type de retour incompatible."**, lors d'une redéfinition d'une méthode avec un type de retour incompatible avec celui de la méthode mère.
- **"Signature incompatible."**, lors d'une redéfinition d'une méthode avec une signature différente de celle des méthodes mères.
- **"double définition d'une méthode."**, lors de la définition d'une méthode qui existe déjà dans la classe.
- **"type inexistant."**, lors de la déclaration d'un paramètre avec un type non défini.
- **"un paramètre ne peut pas être de type void."**, lors de la déclaration d'un paramètre de type void ce qui est interdit.
- **"nom du paramètre déja utilisé."**, lors de la déclaration d'un paramètre avec un nom déjà utilisé.
- **"La méthode $METHODNAME n'existe pas pour la classe $CLASSNAME."**, lorsqu'on fait appel à une méthode qui n'existe pas pour la classe mise en jeu.
- **"Le nombre d'arguments entrés dans $METHODNAME ne correspond pas à sa signature."**, lorsqu'on appelle une méthode mais le nombre de paramètres entrés n'est pas le bon.
- **"Argument $ARGUMENTINDEX de type $ARGUMENTTYPE ne correspond au type du $ARGUMENTINDEXe argument de la méthode $METHODNAME."**, lorsqu'on utilise un paramètre de type incompatible avec celui dans la définition de la méthode lors d'un appel de méthode.
- **"$VARIABLENAME n'est pas un objet."**, lorsqu'on fait appel à une méthode pour une variable qui n'est pas un objet.
- **"$IDENTIFIER n'est pas une instance de classe."**, lors de l'utilisation de instanceOf pour un attribut qui n'est pas une instance de la classe.
- **"$TYPE n'est pas un objet deca."**, lors de l'utilisation de instanceOf pour un objet non reconnu par deca.
- **"Instanceof ne s'applique pas sur les types primitifs."**, lors de l'utilisation de instanceOf pour des types inappropriés.

- **"Classe non définie."**, lors de l'utilisation de new avec une classe non définie.
- **"new déclare les classes uniquement."**, lors de l'utilisation de new pour déclarer une variable qui n'est pas une classe.
- **"l'appel de méthode est utilisable uniquement pour les instances de classe"**, lors d'un appel à une méthode via une variable de type primitif.
- **"l'appel d'attribut est utilisable uniquement pour les instances de classe"**, lors d'un appel à un attribut via une variable de type primitif.
- **"l'attribut $FIELDNAME n'existe pas pour la classe $CLASSNAME"**, lors d'un appel à un attribut via une classe ne possédant pas cet attribut.

3.2 Lors de l'exécution

- **"Erreur : Input/Output erreur ligne $LIGNE position $POSITION"**, lors d'un mauvais format entré suite à un appel de la méthode ReadInt() ou ReadFloat().
- **"Erreur : Cast impossible ligne $LIGNE position $POSITION"**, lors d'un cast entre deux types non compatibles.
- **"Erreur : pile pleine"**, lorsque la pile ne dispose pas assez de place pour stocker les différents paramètres du programme.
- **"Erreur : division par zéro non autorisée ligne $LIGNE position $POSITION"**, lors d'une division par zéro.
- **"Erreur : deferencement de null"**, lors de l'utilisation d'un objet non initialisé ou initialisé à null.
- **"Erreur : Overflow pendant la $OPERATION ligne $LIGNE position $POSITION"**, lors d'un débordement sur une opération arithmétique sur des flottants.
- **"Erreur : modulo par zéro ligne $LIGNE position $POSITION"**, lors d'un modulo par zéro.
- **"Erreur : allocation impossible, tas plein $LIGNE position $POSITION"**, lorsque trop d'objet ont été instancié et qu'il n'y pas plus de place mémoire pour en déclarer un nouveau.

(Pour plus d'informations, consultez le fichier errorList.pdf)

4 Extension TRIGO

4.1 Utilisations de l'extension

Pour l'extension nous avons implémenté une classe Math qui contient les méthodes *cos*, *sin*, *atan*, *asin*, *ulp* et *getPi*.

Afin d'utiliser ces méthodes, il faudra donc ajouter *#include "Math.decah"* en haut de votre fichier, puis pour appeler les méthodes, il vous faudra créer un objet de la class Math (du type *Math math = new Math();*). Il ne restera plus qu'à appeler les méthodes avec l'objet créé :

- **objet.cos(float f)** renvoie le cosinus du flottant f
- **objet.sin(float f)** renvoie le sinus du flottant f
- **objet.asin(float f)** renvoie arcsin(f)
- **objet.atan(float f)** renvoie arctan(f)
- **objet.ulp(float f)** renvoie ulp(f), c'est-à-dire le pas entre ce flottant et le flottant le plus proche
- **objet.pi(void)** renvoie une valeur approchée de pi.

4.2 Limitations de l'extension

La méthode utilisée pour calculer les valeurs de cos, sin, atan et asin est la méthode de CORDIC, qui génère des imprécisions avec les flottants. La précision est donc notre principale limitation sur l'extension.

Pour les fonctions trigonométriques la méthode de CORDIC génère une imprécision de l'ordre de 10^{-8}. De plus, celle ci ne fonctionne que sur l'intervalle $[-\frac{\pi}{2};\frac{\pi}{2}\ ]$. Ainsi, afin de calculer le cosinus d'une valeur plus grande que $\frac{\pi}{2}$, nous sommes obligés de soustraire un certain nombre de fois π. Par contre, cela va multiplier l'imprécision que nous avons sur la valeur de π et donc augmenter fortement l'imprécision sur le calcul total.

De plus, calculer le cosinus ou le sinus d'une très grande valeur augmente le temps d'exécution le temps de ramener le flottant à un modulo π. Le temps supplémentaire est remarquable à partir d'une entrée supérieure ou égale de 10^3. Une entrée supérieure à 10^7 provoque un temps d'attente d'une dizaine de secondes.

5 Limitations du compilateur

5.1 Limite de tas

Lors de l'exécution du fichier assembleur, en l'absence de "garbage collector" et de "free", le tas peut rapidement se remplir. Ainsi, l'utilisateur peut vite être limité dans son code, le compilateur empêchant la génération d'un trop grand nombre d'objets.

5.2 Absence de tableau

Le langage DECA ne supportant pas les types tableaux, notre compilateur propose une bibliothèque implémentant les listes chaînées. Cette bibliothèque est cependant limitée. Seules la strucure même de liste chaînée et une méthode d'insertion en queue de liste sont implémentées. Les méthodes d'insertion en tête, d'insertion par indice, de tri etc, sont laissées à la discrétion de l'utilisateur.

5.3 Optimisation possible du code assembleur

L'objectif premier de ce compilateur est d'être parfaitement fonctionnel. L'optimisation du code assembleur généré vient alors en second plan. C'est pourquoi, il est possible que certaines instructions du code assembleur soit redondantes et ajoutent donc des cycles inutiles.