Projet Génie Logiciel

Gabriel Soria	**Meriem Lyna Safar Remali**	**Julian Gomez**
Camille Kasprzak	**Charles Breton**	**Arthur Lenne**
Alex Soubeyrand	**Maël Veyrat**	

Manuel utilisateur

Limitations du compilateur

1 Déclarations des variables

Ceci est une anomalie mais il est possible d'uliser des mots-clés en théorie réservés pour les type de variable, pour des noms de variables :

```
float float = 8.9;
print(float);
```

✓

```
float int = 8.9;
print(int)
```

✓

```
float boolean = 8.9;
print(boolean)
```

✓

Tous les codes ci-dessus seront acceptés, mais cela ne devrait pas être le cas.

2 Overflow lors de l'affectation d'une valeur à une variable

La valeur limite dans le négatif comme dans la le positif est 2147483647.

```
  int a = 2147483648;
  int b = -2147483648;
```

✗

En revanche la valeur 2147483648 est disponible pour un résultat positif et négatif :

```
  int a = 2147483647;
  print(a+1);
  // resultat 2147483648

  int b = -2147483647;
  print(b-1);
  // resultat -2147483648
```

✓

3 Lecture d'un float

Lorsque l'utilisateur utlise la fonction ReadFloat(), il devra respecter le format suivant :

```
Print("Lecture du float : ")
Readfloat();
```

Terminal

```
Lecture du float : 15.0
```

✓

Ici la bonne syntaxe est respectée. Même si le nombre désiré est un entier, il faut rajouter le point et un 0 après.

Terminal

```
Lecture du float : 15
```

✗

Cette syntaxe ne sera pas acceptée dans le cas d'un Readfloat, seulement dans le cas d'un ReadInt.

4 Appel de fonction et extension des classes

Soit le code suivant :

```
classe A{
  display();
}

classe B extends A{
  super.display();
}
```

✗

Cette opération n'est pas possible car la machine ne reconnait pas le mot clé super. La solution est la suivante :

```
classe A{
  display_A();
}

classe B extends A{
  display_A();
  display_B();
}
```

✓

Le classe B connaîtra la fonction display_A définie dans la classe A dont elle est tirée. Si nous voulons créer une fonction display propre à la classe B, il faudra utiliser un autre nom que celui utilisé dans la classe mère.

5 Définition variables et classes

```
classe A{
  int x;
  int getNumber(){...};
  x = getnumber;
}
```

✗

Il n'est pas possible d'initialiser un attribut de la classe A avec une méthode de cette même classe.

6 Option -p à la compilation

On rappelle que l'option -p permet de décompiler le code. On génère un nouveau code qui doit être identique à celui de départ.

Il n'est pas possible de tester les include. En effet, le code départ contiendra les `#include <fic>` mais le code généré à la suite du test sera le contenu du fichier inclus.

Il faut ajouter le mot clé `this` devant chaque nom de méthode lorsque l'on utilise l'option -p . En effet, dans le code généré, tous les noms de méthode sont précédés par this : `this.nomMethode`

7 Non Garanti de l'unicité du hashCode

Si une fonction à un nom trop long (>1024 caractères) on utilise le hascode du nom de celle-ci pour définir son étiquette. Or en java, la fonction hashcode ne garantit pas l'unicité, il est donc possible mais **trés trés peu probable** que deux méthodes avec un meme nom (majuscule et minuscule différentes) et ce meme nom dépassant la limite de taille, ai les memes d'étiquettes ce qui générera une erreur du type « l'étiquette code.a.XXXXXX est définie deux fois ».

Les options du compilateur, mode opératoire pour utiliser les extensions

(options de la commande decac, configurations à utiliser etc.).

Options disponibles

`-p` : décompile l'arbre abstrait obtenu par l'analyse syntaxique en code et texte et génère un nouveau code.

`-v` : Vérifie le fichier deca, et ne produit aucune sortie en l'absence d'erreur.

`-P` : compile plusieurs fichiers deca en parallèle (multithreading).

`-r X` : Limite le nombre de registres disponibles (de 4 à 16).

`-n` : compile en ne prenant pas en compte les erreurs à l'exécution.

`-d` : permet de lancer la compilation en mode debug. Quatre niveaux de debug sont dispobibles (`Quiet`, `Info`, `Debug`, `Trace`). Par défaut à `Quiet`, ajouter un `-d` augmente de 1 le nievau de debug.

Les extensions

- Calculs de cos, sin, arctan, arcsin

Pour le calcul de sin(value) et cos(value), nous ramenons un premier temps ramener la valeur étudiée dans l'intervalle $[0;\frac{\pi}{4}]$ avec la fonction `static float sin` et `static float cos` .

Après cela, on applique une fonction tirée de la bibliothèque Java à cette valeur pour obtenir le sin et le cos avec les fonctions `static float _cos` `static float _sin` . On veut calculer cos(v) et sin(v). On calcul un coefficient $q = 2 * \frac{v}{\pi}$, on définie ensuite $x = \frac{\pi}{2} * q + \frac{\pi}{4} - v$
Ce qui nous donne donc :
$\cos(v) = \cos\left(\frac{\pi}{2} * q + \frac{\pi}{4} - x\right)$
$\sin(v) = \sin\left(\frac{\pi}{2} * q + \frac{\pi}{4} - x\right)$

Pour le calcul de arctan avec `static float atan` , il n'est pas nécessaire de ramener la valeur dans un intervalle. artan(x) est calculée suivant des approximations selon l'intervalle dans lequel se trouve x (voir code). Pour le calcul de arcsin(x) avec `static float asin` , la valeure passée en paramètre doit nécessairement être entre [-1;1]. Si cette condition est vérifiée, la fonction calcul des approximations en fontion de si $x < 0.5$ et si $x \geq 0.95$.

La fonction `static float ulp` permet de calculer une estimation de l'imprécision sur une valeur. Elle est tirée de la bibliothèque Java.

Les messages d'erreur

Liste des messages d'erreur qui seront rencontrés par l'utilisateur :

Erreurs lors de la vérification du programme deca

Erreur	Description
`condition is not boolean`	le type de la condition (if ou while) n'est pas `boolean`
`decac without argument not yet implemented`	Il manque un argument qu'il faut fournir après la commande decac
`DeclVar name Invalide !`	nom de variable invalide. Cette erreur survient lorsqu'un identifier est utilisé deux fois.
`DeclVar Type Invalide !`	type de variable invalide.
`ERREUR ARGUMENT PRINT NON VALIDE`	L'argument passé au print n'est pas valide et ne peut pas être affiché
`Error during option parsing`	Erreur lors du parsing
`Error operation <ope> not supported for types : <type1> <type2>`	L'opérateur `<ope>` n'est pas supporté pour les types `<type1>` et `<type2>`
`Error unsupported types`	Cette erreur survient lorsqu'un opérateur arithmétique est utilisé avec des types non supportés (ex : division de booleans).
`Expression <expr> has no Type decoration`	L'expression `<expr>` ne possède pas de type, cette erreur survient lorsqu'un type n'a pas été attribué à `<expr>` lors de la vérification
`Failed to open input file <file>`	Le nom de fichier passé en paramètre n'est pas valide ou n'existe pas.
`Failed to open output file <file>`	Le nom de fichier d'écriture des résultats n'est pas valide ou n'existe pas.
`<fonct> not yet implemented`	Fonctionnalité non implémentée dans le compilateur
`Identificateur non defini`	Cette erreur survient lorsque l'on utilise un identifier qui n'a pas été déclaré.
`Identifier <ident> has no attached definition`	L'identifieur `<ident>` n'a pas de définition …

`Minus does not support type <type>`	le moins « - » ne supporte pas le type `<type>`.
`Not does not support type <type>`	Le « non » ou « ! » ne supporte pas le type `<type>`.
`no viable alternative at input <inpt>`	Le parser n'arrive pas à parser l'expression `<inpt>`. Cette erreur survient lorsque le parser ne sait pas décider quel chemin prendre entre deux alternatives d'arbre syntaxique.
`not yet implemented`	Opération non valide car pas encore implémentée
`Parsing cancelled`	Le parsing est interrompu
`Tree <arbre> has no location set`	`<arbre>` n'a pas de position. Cette erreur survient lors de la vérification du contexte, elle est causée par un setLocation oublié à l'étape du parser.
`Type indefini`	Cette erreur survient lorqu'un identifier de type n'est pas reconnu.

Erreurs dans le code Java

Erreur	Description
`Assertion failed: <raison>`	L'assertion a échoué en raison de : `<raison>`
`Comment <comment> contains carriage return character`	Le commentaire `<comment>` contient un retour chariot. Cette erreur survient lorsque l'utilisateur ajoute un commentaire IMA depuis du code Java contenant un `\n`.
`Comment <comment> contains newline character`	Le commentaire `<comment>` contient un caractère de nouvelle ligne. Cette erreur survient lorsqu'un commentaire ajouté par l'utilisateur manuellement contient `\n`
`ConvFloat does not support type <type>`	ConvFloat ne supporte pas le type `<type>` passé en argument. Cette erreur survient lorsque l'on veut caster en float un type qui ne peut pas l'être

`<function> not yet implemented`	Cette erreur survient lorque l'utilisateur appelle une fonction non implémentée.
`Identifier <ident> is not a Exp identifier, you can't call getExpDefinition on it`	Cette erreur survient lorsque la méthode `getExpDefinition` est appelée sur un Identifier n'étant pas une expression.
`Identifier <ident> is not a field identifier, you can't call getFieldDefinition on it`	Cette erreur survient lorsque la méthode `getFieldDefinition` est appelée sur un Identifier n'étant pas un champs de classe.
`Identifier <ident> is not a method identifier, you can't call getMethodDefinition on it`	Cette erreur survient lorsque la méthode `getMethodDefinition` est appelée sur un Identifier n'étant pas une définition de méthode.
`Identifier <ident> is not a variable identifier, you can't call getVariableDefinition on it`	Cette erreur survient lorsque la méthode `getVariableDefinition` est appelée sur un Identifier n'étant pas une variable.
`Not supported yet.`	Opération non supportée
`Should not be called.`	Ne devrait pas être appelée. Cette erreur survient lorsque l'utilisateur appelle une fonction Java qui ne devrait pas être appelée.

Erreurs dans le code IMA

Erreur	Description
`** IMA ** ERREUR ** Ligne <l> :`	Toues les erreurs sont précédées de ce message qui nous indique la ligne où se situe le problème
`BSR avec operande : <op>`	La fonction BSR a été utilisé avec l'opérande qui n'est pas valide pour cette fonction
`ADD avec op1 : <type1> et op2 : <type2>`	Fonction ADD utilisé avec des paramètres. Les types et ne peuvent pas être utilisés ensembles.
WINT avec indefini	Tentative d'affichage du contenu d'un registre qui est vide. L'information à afficher doit être mise dans le registre R1

`W<wtype> avec <R> <type>`	Tentative d'affichage du contenu d'un registre de type avec la fonction d'affichage W de type Ex : WINT avec R1 float Cela ne sera pas acccepté car on veut afficher ujn float avec une fonction d'affichage de float.

Les extensions

Extension trigo

Le fichier Math.decah est composée des méthodes suivantes :

- float ulp(float value)
- float sin(float value)
- float cos(float value)
- float asin(float x)
- float atan(float x)
- float abs(float x)
- float _pow(float x, int n)
- int _convertToIEEE754(float value)
- float _convertFromIEEE754(int bits)
- float _approx_sqrt(float x)
- float _cos(float value)
- float _sin(float value)

NB : les méthodes commençant par un _ sont des méthodes internes à la classe et ne sont pas prévus pour être appelé en dehors de la classe. On ne les présentera pas dans le manuel utilisateur car elles ne sont pas faites pour être utilisées.

ulp permet de trouver l'écart entre un nombre et le prochain nombre supérieur représenté dans la machine.

ulp ne fonctionne pas pour certaines valeurs, notamment les valeurs 0x1p30, 0x1p40, 2147483647 (ulp renvoie 0 pour ces valeurs-là).

Les fonctions sin, cos, asin, atan, abs sont respectivement les fonctions mathématiques sinus, cosinus, arcsin, arctan, valeur absolue.

Extension Float2Float

Le fichier Float2Float.decah est composée des méthodes suivantes :

- float function(float value)
- float LeftRectangle(float a, float b, int n)
- float MidPoint(float a, float b, int n)
- float Trapezoid(float a, float b, int n)
- float Simpson(float a, float b, int n)

LeftRectangle calcule l'intégrale avec la méthode des rectangles gauches. MidPoint calcule l'intégrale avec la méthode du point milieu. Trapezoid calcule l'intégrale avec la méthode des trapèzes. Simpson calcule l'intégrale avec la méthode Simpson.

Constant folding

Le constant folding est la capacité du compilateur d'optimiser le code en précalculant toutes les opérations déterministes sur des constantes (dont on peut connaître le résultat avant l'exécution). Par exemple, ce code :

```
  {
    int a = 2;
    int b = 3 * 2;
    int c = readInt();

    if (true){
      printlnx(a + 10);
    }

    if (true){
      if (a == 2 && false){
        printlnx(b + 10);
      }
    }


    printlnx(a - b);
    printlnx(a * b);
    printlnx(-b);
    printlnx(c + 10);
}
```

Est equivalent a:

```
  {
    int a = 2;
    int b = 6;

    printlnx(12);


    printlnx(-4);
    printlnx(12);
    printlnx(-6);
    printlnx(c + 10);
}
```

À noter que la variable c a été gardée sous forme de variable car on ne peut pas connaître sa valeur à l'avance.