GRENOBLE INP UGA Ensimag

École Nationale Supérieure
d'Informatique et Mathématiques
Appliquées de Grenoble

Documentation de utilisateur

Equipe 03

Benhachem Youssef
Conrado Ivan Garcia Reyes
Bouhout Ilyas
Lemrabet Soufiane
Klou Anas

Grenoble - France
24 Janvier 2021

Instructions

Pour compiler il suffit de taper en ligne de commande “decac test.deca”, celà va créer un fichier test.ass qu'on exécute en tapant “ima test.ass”.

Le compilateur peut recevoir des options pour changer le fonctionnement, après les options les arguments doivent être le nom ou noms des fichiers à compiler.

-b (banner) :	Affiche une bannière indiquant le nom de l'équipe.
-p (parse)	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).
-v (verification) :	Arrête decac après l'étape de vérifications (ne produit aucune sortie en l'absence d'erreur.
-n (no check)	Supprime les tests à l'exécution spécifiés dan spécifiés dans les points 11.1 et 11.3 de la sémantique de Deca.
-r X (registers)	Limite les registres banalisés disponibles à R0 ... R{X-1}, avec 4 <= X <= 16.
-P (parallel)	s'il y a plusieurs fichiers sources, lance la compilation des fichiers en parallèle (pour accélérer la compilation)
-d (debug)	Active les traces de debug. Répéter l'option plusieurs fois pour avoir plus de traces.

N.B. Les options '-p' et '-v' sont incompatibles.

Limitations du compilateur

- Les cast des types ne sont pas implémentés dans le compilateur.
-L'utilisation de la méthode Integer.parseInt() et de Float.parseInt() impose que le nombre des entiers et de flottants ne dépassent pas la limite désigné dans la documentation de ces deux fonctions : ils doivent être codé sur 32 bits au maximum .
-Il est déconseillé d'utiliser un appel des méthodes qui retournent une valeur au sein des expressions conditionnelles (if, while ,else..)
-Une erreur de débordement de pile peut être levée pour une nombre d'appels récursifs
-Il est conseillé d'éviter de travailler avec les objets et d'utiliser un code sans objet du fait d'un comportement inattendu lors de l'appel de plusieurs class
-L'option "-n" n'est pas implémentée , lors d'un appel d'une telle option des messages d'erreur (par exemple Division par zéro) peuvent apparaître .

Les Différents message d'erreurs :

Lors de la compilation, des erreurs peuvent s'afficher sous forme nomFichier.deca:ligne:colonne. Et les différentes formes d'erreurs que vous pourriez trouver sont comme suit :

I - les erreurs contextuelles Sans Objet:

- 1.1- La variable var(nom de la variable) n'est pas déclarée:

cette erreur s'affiche quand une variable est utilisé sans qu'elle soit déclaré

- 1.2- Les deux types Type1 et Type2 sont incompatibles

cette erreur s'affiche quand une opération est faite entre deux types incompatibles, ce genre d'opération peut être une affectation(affecter un booléens à entiers),une opération arithmétique(addition entre booléens et entiers) ou opération booléenne(1 && 2).

- 1.3- La condition doit être un booléen : veut dire que que la condition de if,else,if ou while n'est pas un booléen, exemple, while(1){} va afficher l'erreur en question
- 1.4- Void/String n'est pas un type pour initialiser : veut dire qu'une variable de type void ou string vient d'être déclarée,exemple,(void x/ string s).
- 1.5- Type n'est pas défini : veut dire qu'un type n'est pas défini, exemple(double x)
- 1.6- Print ne prend en argument le type t : veut dire qu'on a essayer de donner à println ou print un argument qui n'est ni entier,ni float ni String, exemple println(true).
- 1.7- la variable var est déjà déclarée : exemple (int x; float y)
- 1.8- Minus ne support pas le type t1: veut que "-" est appliquée sur un type autre que float ou int, exemple (boolean b; boolean a = -b)
- 1.9- Mauvaise initialisation : veut dire initialiser un entier avec un float ou autre type.

II - Les erreurs contextuelles Objet :

En plus d' erreurs que l'analyse contextuelle sans objet peut afficher on trouve aussi d'autres formes d'erreurs liées à la partie objet et déclaration de classe .

2.1 - La classe A n'est définie : veut dire qu'un objet est créé mais son type n'est pas encore défini dans l'environnement type, exemple {A a =new A()}

2.2- La superClass B n'est pas définie : veut dire qu'une classe A essaye d'hériter d'une classe inexistante,example class A extends B{} avec B non défini

2.3- L'attribut "x" n'est pas défini dans Object(la classe racine): veut dire que l'attribut "x" n'est définie dans la hiérarchie d'une classe A .

2.4- L'attribut "x" est protégé : veut dire que vous essayez d'accéder directement à un attribut protégé hors de la classe, exemple :

```
class A{                    class B{
        B b;                        protected int y;
        void setY(){                int x;
                b.y = 12;           A a;
```

```
        }
    }                   }
```

2.5- méthode “M” n’est pas définie dans une classe “A”: appel d’une méthode qui n’est pas définie dans l’héritage de la classe A

2.6-deux Types B et C sont incompatibles : lors de l’affectation d’un objet de B par un autre objet de C, mais B et C ne sont pas une classe et et sa superClasse.

2.7- Le type retour attendu et le retour ne sont pas compatible : veut dire qu’une méthode retourne un type différent de son type, par exemple (int toInt(float){return x;})

2.8-Manque d’argument : celà veut dire qu’une méthode ne prend pas le nombre exacte d’argument, exemple :

```
class A{                                    {
    int add(int n,int m){ return n+m;}          A a = new A();
    }                                           a.add(12,16,20);//a.add(12) }
```

2.9-L’argument param de la méthode doit être de type T : cette erreur s’affiche quand une méthode prend en argument un paramètre qui n’est pas de même type que T ou sous type de T. par exemple toInt(float x) au lieu de toInt(int x).

2.10- Instanceof compare entre objet et Classe : c’est à dire que soit la partie droit de instanceof n’est pas une Classe soit que la partie gauche n’est pas un objet;

2.11- La méthode est déjà définie : celà veut dire qu’une méthode prend déjà ce nom dans la classe en question, sans prendre en considération le type de retour et les signatures. exemple :

```
class A{
        int a,c;
        int set(int b){
            a = b;
            return a;}
        void set(int c){ c =b;}
    }
```

III-Erreurs lexicographiques :

3.1- En générale lorsqu’il manque une parenthèse ou un point virgule, une erreur, qui explique ce qu’il faut ajouter, s’affiche.

3.2- Erreur qui n’accepte pas des nombres trop large par exemple 10...(100 fois)..0

VI-Erreurs liées au code assembleur

- **Error: Overflow during arithmetic operation** : débordement pendant les opérations arithmétiques sur les nombre réels .
- **Error: Division by zer**o : Veut dire division par zero ou modulo zero
- **Error: Input/Output error** : s'affiche quand on fait appel au fonction ReadInt() et ReadFloat() mais on ne saisi pas des entiers ou des réels, exemple

{int a = readInt();} puis en saisi “Bonjour :)”.

- **Error: print float only in hexa form :** veut dire qu’on essaye d’afficher quelque chose qui n’est pas float sous forme hexadécimale, exemple {printlnx(“Hello”)}

- **Erreur : dereferencement de null :** s’affiche quand on manipule un objet null(non initialiser), c'est -à -dire accéder aux attributs ou appliquer des méthodes sur cet objet.

Extension Trigo:

Documentation utilisateur

1-Introduction :

Les performances des fonctions implémentées sont mesurées en deux temps , d'abord en utilisant la structure *double* en *Java*, ensuite en utilisant la structure *float*.

Le temps de calcul est mesuré à l'aide de la méthode *Sytem.nanoTime().* Afin d'être le plus précis possible, on mesure pour un grand nombre d'échantillons (100 000) de 100 entrées le temps nécessaire pour le calcul de l'une des fonctions, et on prend après l'espérance (la moyenne arithmétique), ceci donne le temps pour 100 entrées, ce temps est ensuite divisé par 100 pour obtenir le temps pour une seule entrée.

La précision est assurée à l'aide de la fonction *assertEquals* de la classe *Assertions* .

2-Tableau de performances du CORDIC :

2.1- Structure double :

Le tableau suivant est pour l'implémentation avec la structure *double*.

Fonction	Temps pour une entrée (en ms)	Temps pour 100 entrées (en ms)	Précision
cos	0,00841 ms	0,095 ms	1E-13
sin	0,00800 ms	0,093 ms	1E-13
atan	0,00197 ms	0,082 ms	1E-13
asin	0,00215 ms	0,113 ms	1E-13 *
ulp	0,00010 ms	0,003ms	1E-5

* La précision E-13 de l'*asin* est pour x tel que |x|<0.975, pour x tel que |x|>0.975 la précision diminue tant que |x| -> 1

2.1- Structure float :

Le tableau qui suit est selon l'implémentation avec la structure *float.*

Fonction	Temps pour une entrée (en ms)	Temps pour 100 entrées (en ms)	Meilleur précision
cos	0,00162 ms	0,081 ms	1E-6
sin	0,00084 ms	0,084 ms	1E-6
atan	0,00264 ms	0,264 ms	1E-7
asin	0,00246 ms	0,246 ms	1E-6
ulp	0,00005 ms	0,005 ms	1E-8

* La précision E-6 de l'*asin* est pour x tel que |x|<0.917, pour x tel que |x|>0.917 la précision diminue en s'approchant de 1. (Plus de détail dans le tableau ci-dessous)

*ulp nous garantit une précision de E-6 dans tous les cas, et une précision de E-8 dès que |x|>1

3-Précision détaillée des différentes fonctions :

- cos et sin :

  La précision de ces deux fonctions trigonométriques dépend de l'intervalle de départ :

  : La précision est de 1E-6

  $\mathbb{R} \setminus [-9\pi, 9\pi]$: La précision oscille entre 1E-6 ET 1 E-5

  Explication et idées pour amélioration :

  Les fonctions cos et sin sont **2π**-périodiques, et comme l'implémentation repose sur les égalités trigonométriques entre le cos et le sin pour prolonger les deux fonctions sur tout l'axe réel, l'imprécision vient donc de la fonction « *_anglePrincipal* » . *Comme piste d'amélioration, on pourra au lieu de faire une suite de soustraction dans une boucle while, d'abord localiser l'angle d'entrée dans un intervalle de la forme* **[kπ, (k+1)π]**, *et économiser une suite de soustractions à une multiplication et une addition en plus de comparaisons (ces dernières ne représentent pas une source d'imprécision).*

  ⇨ Après implémentation de l'idée ci-dessus, malheureusement ça n'a pas été suffisant pour augmenter la précision en dehors de **[-9π, 9π]** .

- atan :

  [-0.274 ,0.274] : La précision est de 1E-7

  $\mathbb{R} \setminus$ **[-0.274 ,0.274]** : La précision est 1E-6

- asin :

  La précision l'*asin* pour x tel que |x|<0.917 est 1E-6, ensuite pour x tel que |x|>0.917 la précision diminue en s'approchant de 1. (Plus de détail dans le tableau ci-dessous). On se contente des intervalles positifs ; la précision est pareille pour **[-1,0].**

Intervalle	[0,0.915]	[0.915,0.943]	[0.943,0.965]	[0.965,0.983]	[0.983,0.995]	[0.995,1]
Précision	1E-6	1E-5	1E-4	1E-3	1E-2	1E-1

- ulp :

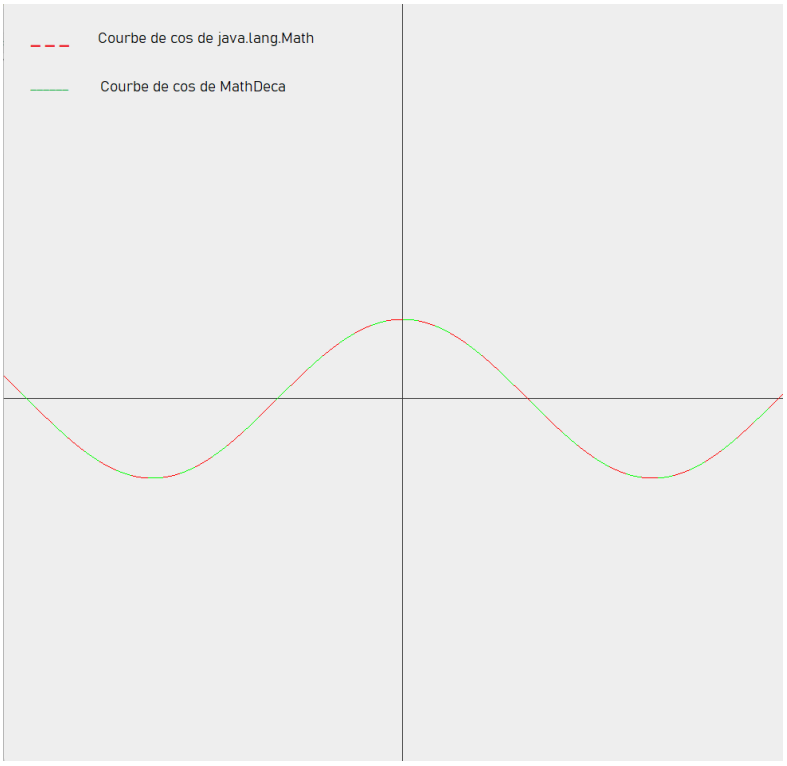
  ulp nous garantit une précision de E-8 dès que |x|>= 1 , et une précision qui varie entre E-6 et E-7 dans]-1,1[

Intervalle	] -∞ ,-1]	]-1,-0.25]	[-0.25,0.25]	[0.25,1[	[1,+∞ [
Précision	1E-8	1E-7	1E-6	1E-7	1E-8

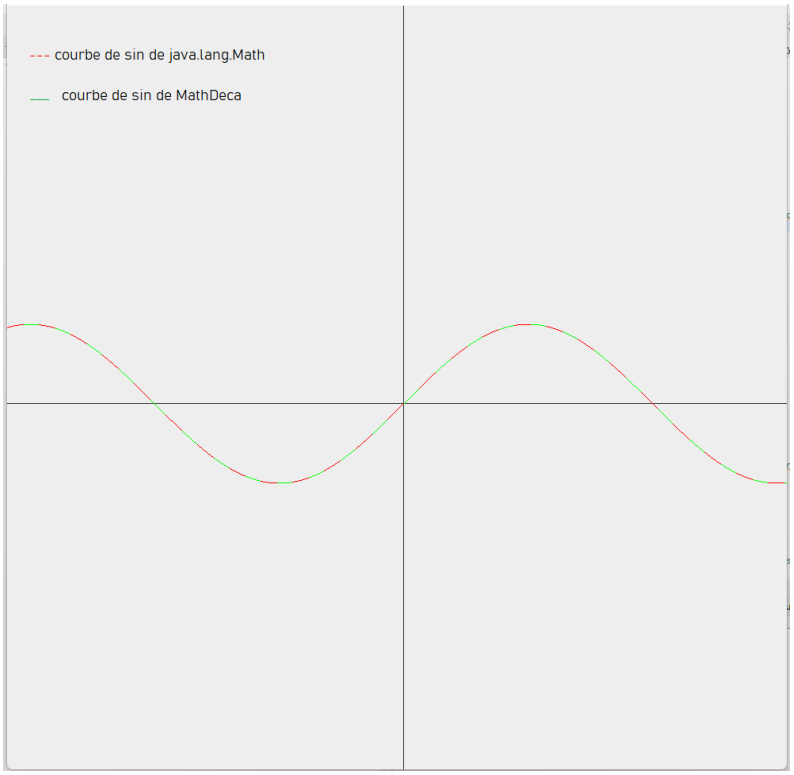
3-Courbe des fonctions:

Les courbes ont été tracées à l'aide des classes de java.awt (abstract window toolkit). On a dû faire notre propre traceur de fonctions, l'unité de ce dernier était de 100 pixels (c'est-à-dire chaque 1 pixel = 0.01). Le traçage est fait sur l'intervalle **[-10,10]** = 1000 pixels pour les deux axes. Les courbes en **- - -** rouge tiret sont les fonctions de la bibliothèque Math de Java, et en **——** **vert continu** les fonctions de notre bibliothèque.

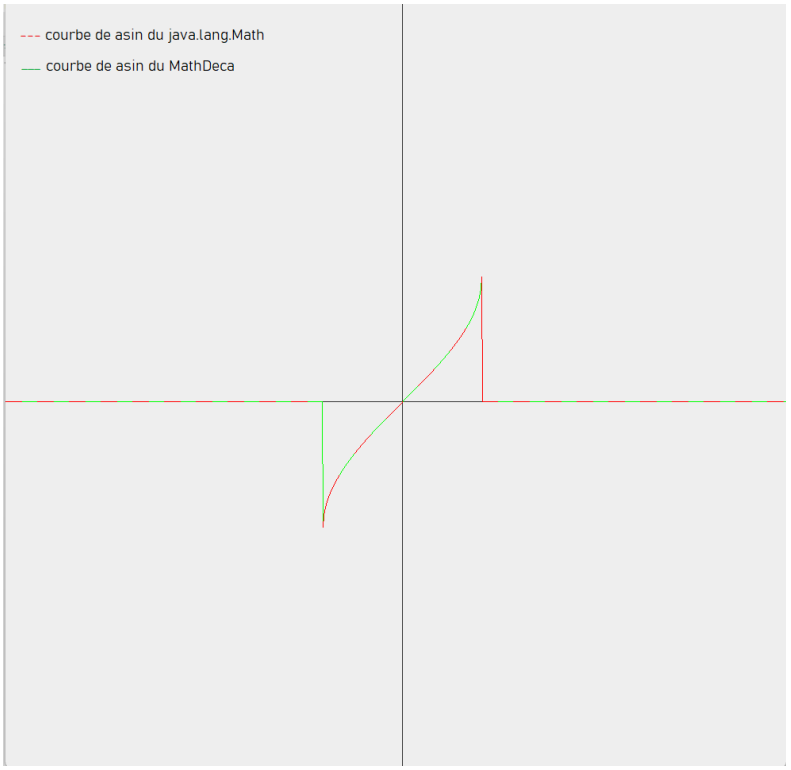
- Fonction cosinus



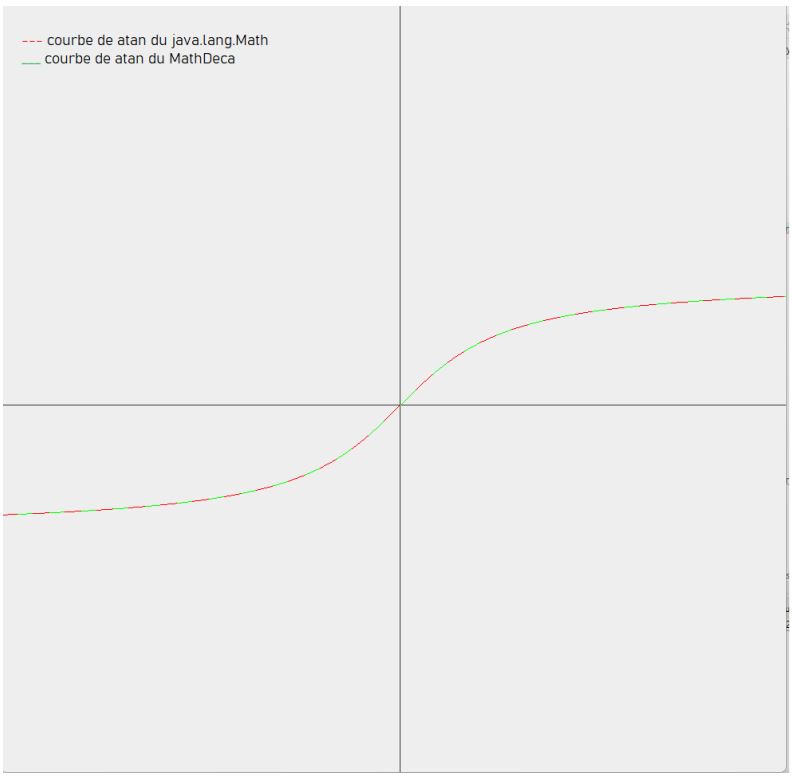
- Fonction sinus



- Fonction tangente



- Fonction arcsin :

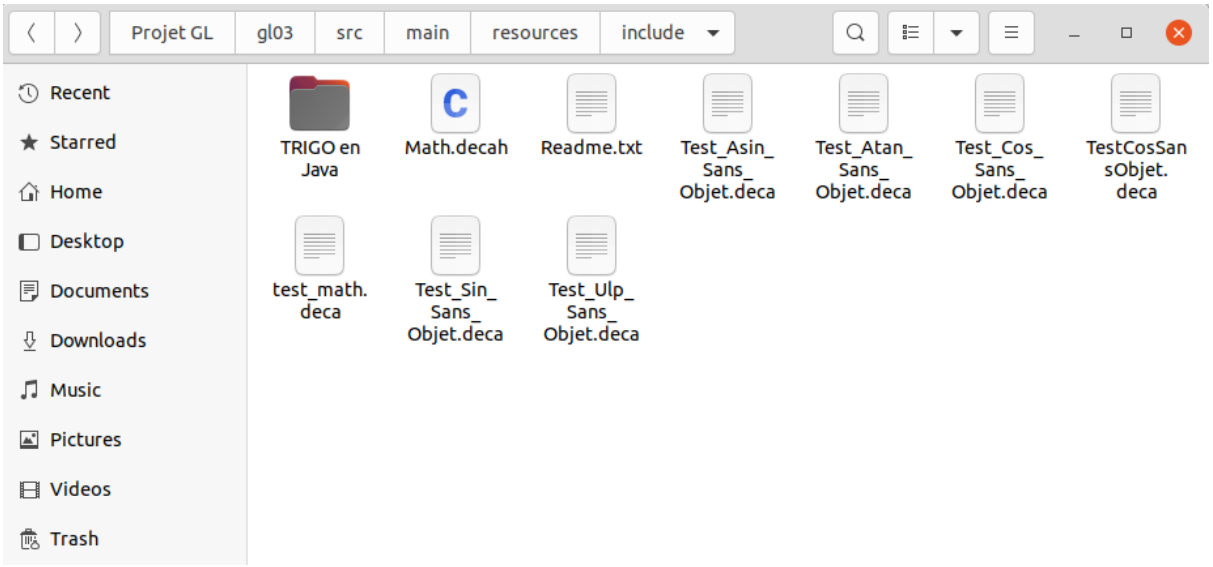


Commentaire :

Les fonctions coïncident parfaitement, et on ne peut distinguer avec l'œil nue la différence entre les deux courbes, c'est d'ailleurs la raison pour laquelle on a décidé de tracer les courbes de la bibliothèques Math de Java en tiret.

4- Instructions utilisateur pour le compilateur:

La partie extension du compilateur se situe sur le répertoire ***src/main/resources/include***



Notre compilateur présente des petits bugs au niveau objet, et comme le fichier math.decah est implémenté en objet , on a décidé de le faire des tests en mode sans objet, comme illustré par l'image ci-dessus, il y a cinq fichiers dont le nom suit la forme suivante : ***Test_Fonction _Sans_Objet.deca***, où ***fonction*** représente la fonction voulue. Le fichier ***readme.txt*** détaille les entrées des fonctions des fichiers tests sans objet, et comment les utiliser pour changer la valeur d'entrée, l'utilisateur a donc le choix d'utiliser la version sans objet de l'extension.

Le dossier « ***TRIGO en Java*** » représente une version Java de la partie extension , ce dossier est en fait un projet Maven qui pourra être exécuter indépendamment de tous les autres fichiers. C'est de ce fichier qu'on a tracé les courbes et mesuré la précision des fonctions.

Rq : En Java, on codait en float parce que double n'est pas valable en Deca, et comme ça conserve la précision.

D'une part, ce dossier contient deux fichiers sources Java : **ForMath.java** et **MathDeca.java** ; comme son nom l'indique ForMath.java contient les fonctions pour (For) MathDeca.java qui lui contient les fonctions trigonométriques avec la fonction ulp.

D'autre part, dans le répertoire des tests, on a à l'aide de **Junit 5** réalisé des tests pour chaque fonction de **ForMath.java** et **MathDeca.java**

Pour compiler et exécuter tous les tests, il suffit d'avoir Maven sur son pc et exécuter la commande : **mvn test** dans le terminal dans dossier ***TRIGO en Java*** .