LE MANUEL UTILISATEUR

BOUTALEB Youssef
BOUHAR Mounsef
MALYAH Lina
EL KAZDAM Zakaria
EL ASLI Adnan

I. limitations et particularités de notre implémentation

1. Présentation du compilateur

Ce compilateur, dédié au langage Deca (un sous-langage de Java), permet de compiler des fichiers source en utilisant la commande `decac -option .` Une base de tests avec des scripts est fournie pour compiler et vérifier les résultats, enregistrés dans des fichiers .res et .dec.

De plus, le compilateur peut décompiler des fichiers, générant des fichiers Deca corrects en langage source.

Les options suivantes sont supportée par notre compilateur :

- *b : (banner)* → affiche une bannière indiquant le nom de l'équipe.
- *p : (parser)* → arrête decac après l'étape de construction de l'arbre, et affiche la décompilation de ce dernier
- *v : (verify)* → arrête decac après l'étape de vérifications
- *n : (no check)* → supprime les tests à l'exécution.
- *r X : (registers)* → limite les registres banalisés disponibles à R0 ... R{X-1}, avec 4 <= X <= 16
- *d : (debug)* → active les traces de debug. Répéter l'option plusieurs fois pour avoir plus de traces.
- *P: (Parallel)* → s'il y a plusieurs fichiers sources, lance la compilation des fichiers en parallèle (pour accélérer la compilation)

2. Limitation de notre compilateur

Tout d'abord, toutes les parties du projet sont implémentées, y compris le cast et l'opérateur **`instanceof`**, que ce soit en analyse syntaxique, contextuelle ou en génération de code. Cependant, nous n'avons pas eu suffisamment de temps pour implémenter la génération de code pour les méthodes ASM. De plus, quelques problèmes de gestion de mémoire persistent dans la partie objet, et nous n'avons pas eu assez de temps pour les déboguer.

Le sans objet fonctionne parfaitement, même avec l'option -r 4. Cependant, pour la partie essentielle et complète du projet, nous n'avons pas pris en compte cette particularité, notamment l'opérateur **`instanceof`** qui utilise 3 registres.

De même, certaines options n'ont pas été entièrement prises en compte :

- L'option -n est partiellement traitée dans le contexte sans objet.
- L'option -w n'a pas encore été traitée.

II. Message d’erreur

1. listes de messages d’erreurs syntaxique

Pour signaler des erreurs de syntaxe, le parser genere des exceptions DecaRecognitionException. Cependant, il y’a des erreurs syntaxiques specifiques géres par d’autres erreurs qui herites de DecaRecognitionException

---

1.1 Erreur d’inclusion des fichiers

CircularInclude: Signale une inclusion circulaire, c'est-à-dire une tentative d'inclusion du fichier lui-même.
IncludeFileNotFound: Indique une tentative d'inclure un fichier qui n'existe pas.

1.2 Erreur d’affectation

InvalidLValue: Déclenchée lorsque le côté gauche d'une affectation n'est pas un identificateur assignable.

1.3 Erreur d’entrée valeur

FloatTooLarge: Cette erreur est déclenchée lorsqu'un programme Deca utilise une valeur de type flottant dont la partie entière ne peut pas être codée sur 32 bits.
IntegerTooLarge: Cette erreur survient lorsqu'un programme Deca utilise une valeur entière qui ne peut pas être codée sur 32 bits.
FloatTooSmall: Cette erreur se produit lorsque dans un programme Deca, la valeur d'un flottant est différente de zéro, mais son arrondi se fait à zéro.

2. listes de messages d’erreurs contextuelles

2.1 Declaration de classes

Erreur de Déclaration de classe

DoubleClassDeclarationException: Le nom des classes doit être unique. Vous avez déclaré la classe X plus d'une fois.

Erreur de Superclasse :

Si l'intention est d'étendre une superclasse qui n'a pas été déclarée préalablement. L’erreur suivante sera déclenchée.

InvalidSuperclassDeclarationException: La superclasse que vous avez déclarée n'est pas une classe valide.

2.2 Declaration de methodes

Erreurs de Déclaration de méthodes

DoubleMethodDeclarationException: La méthode ne peut pas être déclarée car elle a déjà été définie.

DoubleMethodDeclarationException: La méthode ne peut pas être déclarée dans la sous-classe car elle a déjà été définie dans la superclasse comme un champ.

Erreurs de Redéfinition et Surcharges :

Les surcharges de méthodes, c'est-à-dire la déclaration de deux méthodes avec le même nom mais des signatures différentes, ne sont pas autorisées dans Deca.

La redéfinition de méthode est permise lorsque les deux conditions suivantes sont remplies :

- La signature est identique.
- Le type de retour est un sous-type du type de retour original.

Autrement, les exceptions suivantes seront déclenchées :

MethodRedefinitionException: Les surcharges de méthodes ne sont pas autorisées ; seules les redéfinitions de méthodes avec des signatures identiques le sont.

ReturnTypeMismatchException: La redéfinition de méthode est autorisée, mais le type de retour doit être un sous-type du type de retour original.

Erreurs de Déclaration et d'Initialisation des Paramètres et du Retour

VoidParameterTypeException: Le type void n'est pas autorisé pour les paramètres de méthode.

DoubleParameterDeclarationException: Le paramètre ne peut pas être déclaré car il a déjà été défini.

Il n'est pas autorisé de spécifier un retour dans une fonction dont le type de retour est *void*.

VoidReturnTypeException: Le retour de la méthode ne peut pas être void.

InvalidTypeDeclarationException: Impossible de déclarer une variable de type void.

UnknownTypeException: Le type n'existe pas en Deca.

Erreurs d'appel d’une méthode :

MethodNotFoundException: Il n'y a aucune méthode avec le nom spécifié.

NotAMethodException: L'identifiant n'est pas une méthode, c'est un champ.

UnknownMethodException: La méthode est inconnue ou non déclarée.

IncorrectNumberOfParametersException: Mauvais nombre de paramètres pour la méthode appelée.

InvalidObjectClassException: L'objet sur lequel vous appelez une méthode n'est pas du type de classe attendu.

2.3 Déclaration des champs

Erreurs de Déclaration de champs

DoubleFieldDeclarationException: Le champ ne peut pas être déclaré car il a déjà été défini.

DoubleFieldDeclarationException: Le champ ne peut pas être déclaré dans la sous-classe car il a déjà été défini dans la superclasse comme une méthode.

Il est interdit de déclarer un champ avec le type void. Un champ doit avoir un type défini pour stocker des valeurs.

VoidFieldTypeException: Impossible de déclarer un champ de type void.
InitializationIncompatibilityException: Impossible d'initialiser car l'expression d'initialisation est incompatible.

UnknownTypeException: Le type n'existe pas en Deca.

2.4 Déclaration des variables

Erreurs de Déclaration de variable

DoubleVariableDeclarationException: La variable ne peut pas être déclarée car elle a déjà été définie.

InvalidTypeDeclarationException: Impossible de déclarer une variable de type void.

UnknownTypeException: Le type n'existe pas en Deca.

Erreurs d'usage des opérations

Opérations arithmétiques : si op ∈ {plus, minus, mult, divide}

Pour les opérations arithmétiques, les deux opérandes doivent être de types int ou float ; sinon, une erreur sera déclenchée :

IncompatibleTypesException: Seuls les types float ou int sont acceptés pour cette opération.

Opérations booléennes : si op ∈ {&& , || }

Pour les opérations booléennes, les opérandes doivent être de type boolean ; sinon, une erreur sera déclenchée :

IncompatibleTypesException: Opération non autorisée sur des types incompatibles (non de type boolean).

Opérations de comparaison : si op ∈ { == , != , < , > , <= , >=}

Pour les opérations de comparaisons, les opérandes doivent être de types int ou float ; sinon, une erreur sera déclenchée :

IncompatibleTypesException: Impossible de comparer des éléments de types différents. Assurez-vous que les opérandes ont le même type pour l'opération de comparaison.

Opération de négation : si op ∈ { ! }

Elle n'est autorisée que sur des objets de type *null* ou *boolean*

IncompatibleTypesException: Négation non autorisée sur ce type.

Opération modulo: si op ∈ { % }

IncompatibleTypesException: L'opération de modulo ne peut être appliquée qu'aux types entiers.

Erreur d'Utilisation de print :

Les fonctions print, printx, println et printlnx généreront une erreur en cas d'utilisation d'arguments de types autres que int, string ou float.

InvalidPrintTypeException: Seuls les types int, string ou float sont autorisés pour la fonction print.

Erreur d'Utilisation d'Identificateurs:

UndeclaredIdentifierException: L'identifiant n'a pas été déclaré avant utilisation.

Erreurs de Cast :

L'opérateur de conversion de type, également appelé "cast", s'exprime sous la forme (type)(valeur).

Il a deux principales utilisations :

1- (int)(valeur_flottante), convertit la valeur flottante en arrondissant à l'entier le plus proche vers zéro, évitant ainsi tout débordement.

2- (UneClasse)(v), vérifie que la valeur v est du type spécifié, UneClasse. Si le type statique de v est dérivé de UneClasse, la conversion réussit facilement. Sinon, il est nécessaire de vérifier que le type dynamique de v dérive également de UneClasse. En cas d'échec de la conversion, l'erreur suivante est déclenchée.

TypeCastException: Impossible de convertir [type_1] en [type_2]. La conversion a échoué.

Erreur d'usage de instanceof :

InstanceOfIncompatibilityException: L'opérateur instanceof ne peut être effectuée que sur des objets de type classe.

Erreurs de Sélection :

NonClassObjectSelectionException: L'objet sélectionné n'est pas une classe.

NoSuchFieldException: Il n'y a aucun champ avec le nom spécifié.

NotAFieldException: L'identifiant n'est pas un champ ; il peut s'agir d'une méthode ou d'un autre type.

ProtectedAccessException: Impossible d'accéder à l'attribut car il est déclaré comme protégé.

Erreurs d'utilisation de 'this':

L'utilisation de "this" est restreinte aux méthodes à l'intérieur d'une classe et n'est pas autorisée dans un bloc.

ClassOnlyException: Cela ne peut être appelé que dans une classe et non dans la méthode principale.

3. Liste de messages d'erreurs pour la génération de code

Erreur: Overflow during arithmetic operation

- Toute tentative de division entière ou de calcul du reste par zéro entraîne une erreur.
- Les opérations arithmétiques sur les entiers n'entraînent pas de débordement : les calculs sont effectués en utilisant l'opération modulo 2^{32}
- Un débordement sur une opération arithmétique avec des nombres à virgule flottante provoque une erreur.
- Une division flottante par 0.0 provoque une erreur.

.

Erreur: Input/Output error

Cette erreur apparaît lors de l'utilisation de *readint()* ou *readfloat()*, c'est -à -dire quand le type de l'entrée ne correspond pas au type attendu par la fonction.

Erreur: déréférencement null

Cette erreur apparaît lors de l'utilisation d'un objet *null* ou non initialisé.

Erreur: pile pleine

Cette erreur est provoquée quand l'allocation dans la pile est impossible.

Erreur : allocation impossible, tas plein

Cette erreur est provoquée quand l'allocation dans le tas est impossible.

Erreur : la conversion (cast) est incorrecte :

Lorsque la conversion ne peut pas être effectuée, l'exécution se termine avec ce message d'erreur. Cela se produit dans les cas autres que suivants : d'un entier vers un flottant, d'un flottant vers un entier, de null vers n'importe quelle classe, d'une classe vers n'importe quelle superclass, ou d'un type vers le même type.

III. Extensions de la Bibliothèque Standard : Fonctions Trigonométriques

1. Présentation et guide d'utilisation

L 'extension de la bibliothèque standard réalisée consiste initialement en l'implémentation des fonctions trigonométriques suivantes sin, cos, asin, et atan.

Leur paramètre d'entrée doit se trouver dans le domaine de définition de la fonction. Le cas échéant, une erreur est retournée à l'utilisateur. Autrement, ces implémentations retournent une valeur dans le codomaine de la fonction trigonométrique tel que :

$$cos : \mathbb{R} \to [-1; 1]$$
$$sin : \mathbb{R} \to [-1; 1]$$
$$asin :]-1; 1[\to [-\pi/2, \pi/2]$$
$$atan : \mathbb{R} \to [-1; 1]$$

Pour utiliser les fonctions trigonométriques et inverses de notre API, vous devez d'abord *importer la bibliothèque Math.decah* et créer une instance de la classe Math. Voici comment vous pouvez introduire la classe et obtenir une instance pour accéder aux fonctionnalités offertes :

```
#include "Math.decah"
Math customMath = new Math();
```

Une fois que vous avez instancié la classe Math, vous pouvez accéder à toutes les fonctions mathématiques qu'elle offre, y compris les opérations trigonométriques et inverses.
Maintenant que vous avez créé une instance de la classe Math, vous pouvez utiliser les fonctions trigonométriques et inverses de manière simple.

Exemple d'utilisation de la fonction cosinus :

```
float angleInRadians = 0.5f;
// Remplacez cela par l'angle pour lequel vous souhaitez calculer le cosinus
float cosValue = customMath.cos(angleInRadians);
println("Cosinus pour l'angle " + angleInRadians + " : " + cosValue);
```

Exemple d'utilisation de la fonction arctan :

```
float value = 0.8f;
// Remplacez cela par la valeur pour laquelle vous souhaitez calculer l'arc tangent
float atanValue = customMath.atan(value);
println("Arc Tangent pour la valeur " + value + " : " + atanValue);
```

En intégrant la classe Math dans votre application, vous avez désormais accès à un ensemble riche de fonctions mathématiques, y compris les opérations trigonométriques et inverses.

Ces fonctionnalités peuvent être utilisées dans divers domaines, offrant une précision numérique dans des situations telles que la modélisation, la simulation et l'analyse de données. N'oubliez pas d'instancier la classe et d'inclure la bibliothèque Math.decah avant d'utiliser ces fonctionnalités pour bénéficier de l'ensemble complet de la bibliothèque mathématique.

Une dernière fonction est présente dans l'API fournie : l'ULP.

L'ULP, ou Unit in the Last Place, est une mesure de la précision relative des nombres à virgule flottante. Elle représente la plus petite différence possible entre deux nombres consécutifs dans un ensemble de valeurs réelles.

Notre API propose une fonction ULP pour permettre aux utilisateurs d'évaluer la précision relative d'une valeur spécifique. Pour utiliser cette fonction dans vos applications, la méthode ulp prend en paramètre un nombre à virgule flottante et retourne l'ULP correspondant.

float ulp(float value)

Pour évaluer l'ULP d'une valeur spécifique, suivez ces étapes simples dans votre application :

```
float valeur = 1.0f; // Remplacez cela par la valeur pour laquelle vous souhaitez évaluer l'ULP
float precision = customMath.ulp(valeur);
println("ULP pour " + valeur + " : " + precision);
```

Le résultat obtenu représente la plus petite différence possible entre la valeur spécifiée et la valeur suivante dans la représentation des nombres à virgule flottante. Une valeur d'ULP plus petite indique une précision plus élevée entre deux nombres consécutifs.

L'utilisation de la fonction ULP dans notre API permet aux utilisateurs de quantifier la précision relative de leurs données à virgule flottante, ce qui peut être crucial dans des domaines tels que les sciences, l'ingénierie et d'autres applications nécessitant une précision numérique.

2. Limitations de notre extension

Il est essentiel de comprendre les limitations de certaines extensions pour une utilisation judicieuse. Passons maintenant à la section suivante, où nous

examinerons attentivement les limitations inhérentes à ces extensions pour garantir une intégration efficace et précise dans vos applications.
Ces limitations se font ressentir majoritairement au niveau de la précision de notre API.

Nous avons fait le choix de comparer les performances en précision de notre bibliothèque Deca à celles obtenues en Java. Ce choix est judicieux pour plusieurs raisons. Tout d'abord, Java est largement utilisé dans le développement logiciel et est réputé pour sa portabilité, sa fiabilité et son utilisation courante dans des applications diverses. En comparant nos résultats à ceux de Java, nous pouvons fournir une référence pratique et significative pour les utilisateurs, leur permettant d'évaluer la qualité de notre bibliothèque mathématique par rapport à une solution bien établie.

De plus, Java dispose d'une bibliothèque standard robuste qui offre des fonctions mathématiques éprouvées, notamment celles liées aux opérations trigonométriques et inverses, mais aussi l'ULP. En comparant les performances de notre bibliothèque Deca à celles de Java, nous pouvons éclairer les utilisateurs sur la manière dont nos extensions s'alignent en termes de précision et d'efficacité, fournissant ainsi des informations essentielles pour prendre des décisions éclairées lors du choix d'une bibliothèque mathématique pour leurs projets.

Une première étape de vérification est réalisée pour la méthode ULP implémentée. Dans le processus de comparaison des valeurs ULP générées par notre bibliothèque Deca avec celles obtenues par la bibliothèque Math de Java, nous avons effectué plusieurs itérations en générant des nombres aléatoires. Pour chaque nombre généré, nous avons calculé les valeurs ULP correspondantes à l'aide des deux bibliothèques, puis nous avons enregistré ces résultats dans un tableau. L'objectif était de mesurer la différence entre les valeurs ULP calculées par nos propres méthodes et celles fournies par Java.

Cependant, lors de l'exécution des itérations, nous avons observé que la différence obtenue entre les valeurs ULP était systématiquement nulle. Cette observation peut être attribuée à la grande précision des calculs en virgule flottante dans les bibliothèques standard, ce qui conduit à des résultats extrêmement proches. En d'autres termes, les valeurs ULP générées par notre bibliothèque Deca et celles fournies par Java sont pratiquement identiques.

Cette convergence des résultats témoigne de la qualité de nos implémentations et confirme que les deux bibliothèques fournissent des valeurs ULP cohérentes pour les nombres générés. Ces observations renforcent la fiabilité de notre bibliothèque Deca dans la gestion précise des nombres à virgule flottante et

démontrent sa capacité à rivaliser avec les fonctionnalités de précision de la bibliothèque Math de Java.

Voici des extraits de tableaux obtenus :

Valeur aléatoire	Valeur obtenue	Valeur attendue	Différence
105442508.0966717700	8.0000000000e+00	8.0000000000e+00	0.0
204188473.0441630000	1.6000000000e+01	1.6000000000e+01	0.0
336772314.4132407300	3.2000000000e+01	3.2000000000e+01	0.0
941840309.9522254000	6.4000000000e+01	6.4000000000e+01	0.0
861096882.0621190000	6.4000000000e+01	6.4000000000e+01	0.0
965441084.8371772000	6.4000000000e+01	6.4000000000e+01	0.0
7724054.5829050370	5.0000000000e-01	5.0000000000e-01	0.0
325635554.5025673000	3.2000000000e+01	3.2000000000e+01	0.0
994528292.6258913000	6.4000000000e+01	6.4000000000e+01	0.0
586214725.3477075000	6.4000000000e+01	6.4000000000e+01	0.0

pour des valeurs comprises entre 0 et 1000000000

Valeur aléatoire	Valeur obtenue	Valeur attendue	Différence
582.1137503108	6.1035156250e-05	6.1035156250e-05	0.0
96.6815919599	7.6293945313e-06	7.6293945313e-06	0.0
881.5367994012	6.1035156250e-05	6.1035156250e-05	0.0
495.1133612501	3.0517578125e-05	3.0517578125e-05	0.0
717.9623291956	6.1035156250e-05	6.1035156250e-05	0.0
67.3338485832	7.6293945313e-06	7.6293945313e-06	0.0
403.4939387177	3.0517578125e-05	3.0517578125e-05	0.0
838.3089050064	6.1035156250e-05	6.1035156250e-05	0.0
44.6475342307	3.8146972656e-06	3.8146972656e-06	0.0
472.4943807198	3.0517578125e-05	3.0517578125e-05	0.0

pour des valeurs comprises entre 0 et 1000

Valeur aléatoire	Valeur obtenue	Valeur attendue	Différence
0.6167369748	5.9604644775e-08	5.9604644775e-08	0.0
0.1069164362	7.4505805969e-09	7.4505805969e-09	0.0
0.6077691081	5.9604644775e-08	5.9604644775e-08	0.0
0.3127717291	2.9802322388e-08	2.9802322388e-08	0.0
0.8496606263	5.9604644775e-08	5.9604644775e-08	0.0
0.7564326526	5.9604644775e-08	5.9604644775e-08	0.0
0.9395998555	5.9604644775e-08	5.9604644775e-08	0.0
0.2910864603	2.9802322388e-08	2.9802322388e-08	0.0
0.3640650596	2.9802322388e-08	2.9802322388e-08	0.0
0.9413396469	5.9604644775e-08	5.9604644775e-08	0.0

pour des valeurs comprises entre 0 et 1

Notre démarche consiste ensuite à comparer les résultats des fonctions trigonométriques (sin, cos, asin, atan) entre la bibliothèque Math et la bibliothèque standard de Java (Math). Pour évaluer la précision relative, on utilise la différence en ULP, qui mesure la plus petite variation possible entre deux nombres consécutifs dans une représentation à virgule flottante. Cela offre une évaluation fine de la qualité des résultats en tenant compte de la représentation précise des nombres. La

comparaison en ULP sert ainsi d'indicateur objectif pour évaluer la performance de la bibliothèque Math par rapport à la bibliothèque standard de Java.

Voici quelques extraits des résultats obtenus suivant la fonction et le domaine d'étude :

Valeur aléatoire	Valeur obtenue	Valeur attendue	Diff ULP
0.1121445153	0.1123809218	0.1123809179	0.0
-0.9933138361	-1.4550931454	-1.4550930162	1.0
-0.8151735751	-0.9530286789	-0.9530288039	2.0
-0.0828705309	-0.0829656720	-0.0829656778	0.0
0.7653204582	0.8715391159	0.8715390601	0.0
0.8553854742	1.0262945890	1.0262945531	0.0
0.4536118537	0.4708142281	0.4708139796	8.0
0.5771832027	0.6152752638	0.6152751096	2.0
-0.4927771770	-0.5152782202	-0.5152784661	4.0
-0.5513382114	-0.5839673281	-0.5839674173	1.0

pour l'arcsin entre -1 et 1

Valeur aléatoire	Valeur obtenue	Valeur attendue	Diff ULP
5.5850536564	1.3936244249	1.3936243959	0.0
2.1610709152	1.1374038458	1.1374037863	0.0
1.8011469056	1.0639681816	1.0639681874	0.0
1.8143160018	1.0670540333	1.0670538290	1.0
8.7893359566	1.4575092793	1.4575092308	0.0
3.8452250125	1.3163695335	1.3163694416	0.0
0.6162862208	0.5523086786	0.5523086735	0.0
2.0106023225	1.1092602015	1.1092602244	0.0
8.8059169796	1.4577207565	1.4577207284	0.0
9.2925499594	1.4635958672	1.4635957829	0.0

pour l'arctan entre 0 et 10

Valeur aléatoire	Valeur obtenue	Valeur attendue	Diff ULP
477.3280599601	1.5687013865	1.5687013346	0.0
536.8288187402	1.5689336061	1.5689335377	0.0
87.4536745273	1.5593622923	1.5593621998	0.0
842.6785920909	1.5696096420	1.5696096353	0.0
700.0373044426	1.5693678856	1.5693678325	0.0
335.5438147456	1.5678161383	1.5678160989	0.0
668.3543646269	1.5693001747	1.5693001156	0.0
125.8325892858	1.5628495216	1.5628494272	0.0
76.5367716715	1.5577315092	1.5577314551	0.0
318.1468589027	1.5676531792	1.5676531347	0.0

pour l'arctan entre 0 et 1000

Valeur aléatoire	Valeur obtenue	Valeur attendue	Diff ULP
0.8472769641	0.6620262861	0.6620264600	2.0
3.1131951079	-0.9995968342	-0.9995968168	0.0
2.9053275337	-0.9722189903	-0.9722189889	0.0
0.5830796906	0.8347709179	0.8347709417	0.0
0.5944711222	0.8284447193	0.8284448237	1.0
1.2892079248	0.2778818607	0.2778818417	0.0
0.2503277278	0.9688312411	0.9688312885	0.0
0.3480761394	0.9400306344	0.9400306616	0.0
0.1840898656	0.9831032753	0.9831032596	0.0
3.0056776989	-0.9907777309	-0.9907777724	0.0

pour le cosinus entre 0 et π

Valeur aléatoire	Valeur obtenue	Valeur attendue	Diff ULP
918994.3715128902	-0.9993523955	-0.9998103007	7682.0
789957.0393563312	-0.9251042008	-0.9110310949	236107.0
595896.5758217452	0.7393113375	0.7526549414	223868.0
950962.8569574978	-0.9204326272	-0.9283390607	132647.0
223221.6385073399	0.4639623165	0.4661159062	72262.0
850207.9110421134	0.2353844643	0.2590103384	792752.0
869137.7149343037	-0.9342727065	-0.9213118141	217447.0
628761.0140236754	-0.9530918598	-0.8864130994	1118684.0
578534.7599197470	-0.7966099977	-0.4993194316	9975416.0
224043.9936170086	-0.2538929582	-0.2544414244	18403.0

pour le cosinus entre 0 et 1000000

La fonction sinus a le même comportement. En effet, le calcul du sinus et du cosinus sont très similaires dans notre implémentation.

Les fonctions trigonométriques, telles que atan et asin, fournies par notre bibliothèque Math, démontrent une précision remarquable, souvent atteignant quelques ulps de différence par rapport à leurs homologues de la bibliothèque Java standard. Ces deux fonctions sont particulièrement fiables dans la plage d'entrée définie, généralement [-1, 1]. Cependant, il est important de noter que les fonctions sin et cos peuvent présenter une précision légèrement moindre, surtout lorsque les angles s'éloignent du cercle trigonométrique unitaire. Les erreurs peuvent s'accumuler en fonction de la distance par rapport à ce cercle. Cependant, pour de nombreuses applications courantes et calculs usuels, ces petites variations n'ont généralement qu'un impact négligeable. En conséquence, notre bibliothèque offre un équilibre optimal entre précision et performance pour un large éventail de scénarios d'utilisation.