Manuel Utilisateur
Compilateur Deca

Projet Génie Logiciel, groupe 15

January 23, 2023

Contents

1 Spécificités et limitations du compilateur

1.1 Description du compilateur

Le compilateur permet de compiler des programmes Deca, un sous-langage du langage de programmation Java. Il prend en entrée un fichier écrit dans le langage deca et retourne ce programme traduit en assembleur si le programme est correct, et une erreur sinon.

Pour simplement compiler un programme en .deca, il suffit de trouver le chemin entre le répertoire courant et le répertoire où se trouve le fichier et de taper:

decac chemin/<fichier>.deca

Il y a aussi la possibilité d'utiliser des options pour personnaliser la compilation. La forme générique de l'instruction devient alors:

decac [[-p | -v] [-n] [-r X] [-d]* [-P] [-w] chemin/<fichier.deca>] | [-b]

-b: S'utilise tout seul (ex: decac -b), affiche une bannière indiquant le nom de l'équipe.

-p: Arrête le compilateur après l'étape de validation de la syntaxe, et affiche le programme déca entré dans le compilateur reconstruit par le compilateur après l'analyse syntaxique (i.e. s'il n'y a qu'un fichier source à compiler, la sortie doit être un programme deca syntaxiquement correct).

-v: Arrête la compilation après l'étape de vérifications de la correction syntaxique du programme (ne produit aucune sortie en l'absence d'erreur) Ne peut pas s'utiliser simultanément avec l'option -p.

-n: Ne détecte pas les erreurs spécifiées dans II.3 .

-r X: Limite le nombre de registres de la pile que la machine peut utiliser avec 4¡=X¡=16.. ex: "decac -r 4 fichier.deca" limite les registres utilisables à R0, R1, R2 et R3.

-P: s'il y a plusieurs fichiers sources, lance la compilation des fichiers en parallèle (pour accélérer la compilation)

2 Messages d'erreur possibles

2.1 Erreurs de syntaxe hors-contexte

- Les littéraux d'entiers acceptés par le langage doit être capable d'être écrit sur 32, sinon une erreur syntaxique sera déclenchée.
- Les littéraux flottants sont convertis au numéro le plus proche dans la représentation des flottants : IEEE-754 simple précision, si l'utilisateur donne un littéral de float qui très grands ou très petit par rapport à cette representation une erreur sera déclenchée.

2.2 Erreurs de syntaxe contextuelle

- Utiliser une variable "a" non définie renvoie le message d'erreur suivant: "La variable a n'est pas définie". (règle 0.1)

  Exemple:
  {a = 5;}

- Déclarer une variable à partir d'un type "t" non présent dans l'environnement des types renvoie le message d'erreur suivant: "Le type t n'est pas défini." (règle 0.2)

  Exemple:
  {String str;}

- Déclarer une variable à partir d'un type "t" qui n'existe pas du point de vue du compilateur (comme une classe qui n'a pas été définie ou un type inaccessible à l'utilisateur, comme String) renvoie le message d'erreur suivant: "Le type t n'est pas défini." (règle 0.2)

  Exemple:
  {String str;}
  ou
  {A a;}
  //Le programme ne définit pas A avant de l'instancier.

- Déclarer une seconde fois une variable "a" déjà déclarée renvoie le message d'erreur suivant: "La variable a est déjà déclarée". (règle 3.17)

  Exemple:
  {int a=0;
  int a=1;}

- Déclarer une variable avec le type void renvoie le message d'erreur suivant :"Déclaration de variables de type void impossible". (règle 3.17)

Exemple:
{void a;}

- Initialiser une variable de type t1 ou effectuer une affectation avec une valeur d'un type différent t2 et qui n'est pas compatible à l'affectation renvoie le message d'erreur suivant: "Le type de l'expression de droite est t2 alors que le type attendu est t1". (règles 3.17 et 3.32)

Exemple:
{int a;
boolean b;
a = b;}

- Utiliser une expression dont le type n'est pas boolean provoque le message d'erreur suivant:"la condition doit être de type boolean". (règle 3.29)

Exemple:
{if (5){} }

- Afficher une expression de type boolean renvoie le message d'erreur suivant:"On ne peut pas print de booléens".

Exemple:
{boolean b = true;
println(b);}

- Utiliser le moins "-" unaire sur des expressions dont le type n'est ni int ni float provoque une erreur et renvoie le message suivant: "Le moins unaire ne s'applique qu'aux entiers et aux flottants". (règle 3.62)

Exemple:
{boolean b = true;
boolean c = false;
b = -c;}

- Utiliser une opérande de type différent de int provoque le message d'erreur suivant: "Les opérandes utilisées pour un calcul de modulo doivent être des entiers". (règle 3.51)

Exemple:
int a = 1;
float b = 0.5;
a = a % b;

- Utiliser une expression dont le type est autre que boolean provoque le message d'erreur suivant: "Not ne s'applique qu'au type boolean". (règle 3.63)

Exemple:
{int a = 0;
a = !a;}

- Utiliser une opérande dont le type est ni int ni float dans le cadre d'une opération arithmétiques provoque le message d'erreur suivant: "Les opérations arithmétiques ne sont compatibles qu'avec des int et des float". (règles 3.49 à 3.53)

Exemple:
{float b = 0.5;
boolean c = true;

b = b + c;}

- Utiliser une opérande dont le type n'est pas boolean dans le cadre d'une opération booléenne provoque le message d'erreur suivant: "Les opérations booléennes ne sont compatibles qu'avec des boolean". (règles 3.54 et 3.55)

Exemple:
{int a = 1;
boolean b = true;
b = b && a;}

- -La double déclaration d'une classe provoque le message d'erreur suivant: "Il existe déjà une classe de nom nom_classe ".(règle1.3)

Example:
class A{
}
class A{
}

Si on a une classe A qui hérite d'une classe B alors B doit être définie comme classe et déclarée avant la déclaration de A.

Example1:
class A extends B{
class B{

Example1 provoque le message d'erreur suivant : "La super classe B n'est pas défini"

Example 2:
class A extends int

Example2 provoque le message d'erreur suivant: "une super classe doit avoir un type de classe". condition (règle 1.3).

Pour les champs (Fields) d'une classe: -la double déclaration d'un champ dans une classe provoque le message d'erreur suivant "Il existe déjà un champ du même nom ¡nom_champ > "(règle2.4).

Exemple:
class A
int x;
int x;

-la Sélection ne peut s'appliquer qu'à des objets de classe et le membre de la sélection doit être un champ défini, et public si on a pas dans classe où est défini le champ. Si le champ est protected la classe dans laquelle on fait une sélection doit être une sous classe de celle où est défini le champ et l'objet sur lequel on applique la sélection doit être un sous classe de la classe courante. (règle 3.65 et 3.66).

Example 1:
class A
protected int x;
}
class B{
A a = new A();
void modify(){
// Erreur car B n'est pas un fils de A
a.x = 2;
}
}

Example 2:
class A{
protected int x;
}
class B extends A
int getX(A a)
return a.x;
}
}

génère les messages d'erreurs respectives: “La classe B doit être un sous-classes de A où est défini le champs x et la A doit être doit être un sous classe de B” “La classe B doit être un sous-classes de A où est défini le champs x et la A doit être doit être un sous classe de B”

-Pour l'appel d'une méthode: l'appelant doit être un objet de type classe et l'identificateur de méthode doit être défini comme méthode dans la classe de l'objet appelant ou dans sa super classe avec la même signature. (règles 3.71 3.72 3.73 3.74).

Example 1 :
class A{
}
{
A a = new A();
a.mCall();
}
Example 2:
class A{
void f(){
println(”mthodCall”);
}
void callf(){
int x = 0;
x.f();
}
}
Example 3:
class A{
void f(){
println(”methodCall”);
}
}
{
a.f();
}
Example 4:
class A{
int method;
}
{
A a = new A();
a.method();
}
Example 5:

class A
void f(A a)
}
}
class B
}
{
A a = new A();
B b = new B();
a.f(b);
}

-le type de retour de la méthode redéfinie doit être une sous classe de celui de la méthode qui existait et les paramètres doivent être des sous classes de ceux de celle-ci. si Non il va s'afficher le même message d'erreur. (règle 2.7).

-La redéfinition d'une qui existait déjà dans une super classe et possible mais il doivent avoir le même signature en nombre et en Type et le même type de retour. si non ça va provoquer les messages d'erreurs suivant : "[ERROR] The current method don't have a proper type" (règle 2.7).

-La double déclaration d'une méthode dans une classe ou avec un nom d'un champ qui existait déjà est impossible et provoque le message d'erreur suivant: "il existe déja une méthod ou un champ de même nom " (règle 2.6)

2.3 Erreurs d'exécution du code assembleur

Cette catégorie regroupe les erreurs qui se manifestent à l'exécution, elles sont prises en charge par le code assembleur que fournit le compilateur.

Elles se répartissent en trois catégorie que nous allons décrire ci-dessous:

2.3.1 Erreurs dues à un programme incorrecte

- overflow_error: survient quand une valeur assignée ou calculée par le programme, ou entrée par l'utilisateur par le biais des fonctions readInt et readFloat dépasse la valeur limite pouvant être stockée. Cette valeur est très différente entre les int et les floats.
- div0_error: Survient lorsqu'une division entière ou un modulo par 0 s'apprête à être effectuée lors de l'exécution du programme assembleur.
- missing_return_error: Survient lorsqu'une méthode de type non-void ne possède pas d'instruction return.
- deref_null_error: Survient lorsque le programme tente d'utiliser un objet qui n'a pas été instancié ou un attribut d'objet qui n'a pas été initialisé.
- cast_error: Survient lorsque le programme tente d'effectuer une conversion de type (un cast) impossible. Pour rappel, l'opération de cast s'écrit de façon générique comme suit:

  (Type dans lequel on veut convertir x) (x)

  Les casts possible dans le cadre du langage déca sont:

  – (int)(f), avec f une expression de type float. Dans ce cas le tout prend la valeur de l'entier le plus proche de f.

- (float)(i), avec i une expression de type int. Dans ce cas le tout prend la valeur float correspondant à la valeur entière de i (ex: 4 devient 4.0).
- (C)(v), avec C une classe et v une expression correspondant à une classe. Dans ce cas, le tout est égal à v si c'est bien une instance de la classe C, et lève une erreur sinon.

De manière générale, toute conversion d'une expression dans son propre type sera équivalente à l'opération identité.

2.3.2 Erreurs dues à un programme correcte, dont l'exécution dépende de l'utilisateur:

- io_error: Abrégé de input/output error, survient quand le type d'une valeur entrée par l'utilisateur lors de l'exécution de l'instruction readInt ou readFloat ne correspond pas au type d'entrée de ladite instruction.

2.3.3 Erreurs dues à un programme correcte, dont l'exécution dépasse les limites de la machine:

- stack_overflow_error: Survient lorsque que la machine arrive à court d'espace mémoire allouable pour exécuter un programme. À noter que le nombre de registres utilisable par la machine est artificiellement réductible via l'utilisation de l'option "-r" du compilateur. ex: decac -r 4 fichier_test.deca
- heap_overflow_error : déclanché quand on essaye de reserver plus d'espace mémoire dans le tas que la machine contient.

3 Gestion des tableaux

3.1 Mode opératoire pour utiliser l'extension

Ce compilateur de deca a introduit les tableaux qui sont très pratiques pour faire des opérations sur un nombre larges de variables.Les éléments du tableaux sont indexées à partir 0.

- Déclaration d'un tableau:
  - Syntax: Il en existe uniquement deux varient de tableau qu'on peut définir un tableau 1d et un tableau 2d.
  - Pour déclarer un tableau 1d il faut suivre la règle suivante: "type[] Ident" et pour un tableau 2d: "type[][] Ident".
  - Les types de base du langage deca qui peuvent être utilisé dans la déclaration de tableaux sont : int, boolean, float et Object.
  - Chaque classe ajoutée par l'utilisateur peut être utilisée comme type lors de la déclaration du tableau.
  - Ident représente l'identifiant du tableau.
    Exemple:
    class A{}
    {
    int[] tableauEntier;
    float[][] floatMatrice;
    boolean tableauBoolean;
    Object[] tableauClassObject;
    A[] tableauClassA;
    }

- Initialisation des tableaux:
  - Après avoir déclaré un tableau, afin de l'exploiter il faut lui associer de l'espace mémoire.
  - Syntax: on peut associer de l'espace mémoire uniquement pour les tableaux 1 et 2d similairement à la déclaration de tableau.
  - Pour initialiser un tableau 1d il faut respecter la règle suivant: new type[expr] et pour les tableaux 2d new type[expr][expr].
  - Le new est un token qui doit être présent lors de l'initialisation.
  - Les types possibles sont les mêmes types que ceux décrits dans le paragraphe de la déclaration, et quand un tableau est initialisé on doit l'associer à une déclaration de tableau avec un type et une dimension identique.
  - Les expr qui entre crochets dans l'initialisation de tableau doivent être un entier strictement positif.
    Exemple:
    class A{}
    {
    int[] tableauEntier = new int[5];
    float[][] floatMatrice = new float[15][3];
    boolean tableauBoolean = new boolean[3];
    Object[][] tableauClassObject = new Object[5][10];
    A[] tableauClassA = new A[5];
    }
- Attribuer des valeurs aux éléments du tableau:
  - Après avoir déclaré et initialisé un tableau, on peut associer à ces éléments des valeurs.
  - Pour attribuer une valeur à un tableau 1d il faut respecter la règle suivant: "ident[expr] = expr" et pour les tableaux 2d " ident[expr][expr] = expr"
  - Le type de expr doit être le même que celles des éléments du tableau avec l'identifiant ident.
  - Les expressions qui sont entre les crochets doivent être des entiers positifs car les index commencent par 0 et non pas par 1.
  - Les index doivent être strictement inférieurs à celles de la taille du tableau défini lors de la initialisation, par exemple pour le tableau suivant (int[][] tableauInt = new int[8][9]). Les éléments accessibles sont : tableauInt[0-7][0-8])

```
Exemple;
class A{
int x;
}
{
int[] tableauEntier = new int[5];
float[][] floatMatrice = new float[15][3];
A[] tableauClassA = new A[5];
//Attribution des valeurs aux éléments du tableau
tableauEntier[1] = 5;
floatMatrice[7][2] = 7;
tableauClassA[3] = new A();
}
```

- Récupération des caractéristiques du tableau:

- Après avoir déclaré et initialisé un tableau et Attribuée une valeur à un élément du tableau, l'utilisateur est maintenant capable d'accéder à cette valeur.
- Pour récupérer une valeur d'un tableau 1d il faut suivre la règle suivant: "ident[expr]" et pour les tableaux 2d " ident[expr][expr]"
- Les expressions qui sont entre les crochets doivent être des entiers positifs et inférieur à la taille du tableau.

Exemple:

```
class A{int x;}
{
//Déclaration de tableaux
int[] tabInt = new int[5]; float[][] floatMat = new float[15][3]; A[] tabA = new A[5];
//Attribution des valeurs aux éléments du tableau
tabInt[1] = 5; floatMat[7][2] = 7; tabA[3] = new A();
//Récupération et exploitation des éléments du tableau:
println("tabInt[1] = "tabInt[1]);
tabA[3].x = 2;
println("tabA[3].x = ", tabA[3].x);
println( "2*(int)(floatMat[7][2]) = ", 2*(int)(floatMat[7][2]) );
}
```

Output:

```
tabInt[1] = 5;
tabA[3].x = 2;
2*(int)(floatMat[7][2]) = 14;
```

- Récupération des caractéristiques du tableau:
  - Le compilateur permet à l'utilisateur de récupérer la taille du tableau.
  - L'accès à la taille d'une tableau est très important plusieurs qu'on peut faire avec les tableaux surtout pour le parcours du tableaux, et pour faire des opérations avec les tableaux.
  - Pour accéder à la taille d'une matrice, il faut passer une sélection.
  - Pour une matrice 1d il faut suivre la règle suivante Lvalue.ident avec Lvalue est l'identifiant de la matrice dont on veut sélectionner des éléments et ident est l'élément qu'on veut sélectionner qui ne peut être que size1D dans le cas d'un tableau 1d. le size1D pour un tableau 1D représente le nombre d'éléments dans le tableau.
  - Pour les matrices, il faut suivre la même règle mais on accéder à deux identifiants qui sont size1D i.e nombre de lignes et size2D i.e nombre de colonnes
  - Il n'existe pas d'autre caractéristiques qui sont stockées dans la matrice

Exemple:

```
{
int[] x = new int[5];
int[][] matriceEntier = new int[5][4];
int iterX = 0;
while (iterX ¡ x.size1D){
x[iterX] = iterX;
iterX = iterX+1;
}
println("nb lig=", matriceEntier.size1D, "/ nb col=", matriceEntier.size2D);
}
```

Output:
nb lig=5 / nb col=4

- Exploitation du polymorphism dans les tableaux:
  - Le compilateur permet de définir un tableau avec un type de classe, initialement on peut déclarer un tableau uniquement avec le class Object, mais on peut être cela au tout les classes défini par l'utilisateur.
  - Si on veut créer un tableau qui contient plusieurs classes qui sont différentes mais qui sont dérivées de la même classe comme la classe Object qui est le parent de toutes les classes.
  - On peut dans ce cas crée un tableau de la classe parent, et mettre des classes qui dérive de la classe parent et exploitée ce qui est stockée dans les classes

Exemple:
```
class A{
int x = 1;
}
{
Object[] tableObject = new Object[10];
tableObject[1] = new A();
println(((A)(tableObject[1])).x);
}
```
Output:
((A)(tableObject[1])).x = 1

3.2 Limitations des tableaux

- Le compilateur propose uniquement deux des tableaux 1D et 2D.
- La taille d'un tableau ne peut pas changer lors de utilisation.
- C'est impossible de libérer l'espace associé au tableau.
- Les tableaux ne sont pas initialisées lors de la déclaration, ainsi l'accès à un élément du tableau qui n'a pas de valeur lance une erreur d'exécution.
- On ne peut pas attribuer un tableau de flottants à un tableau d'entier, on peut copier les éléments des tableaux élément par élément mais une attribution directe n'est pas possible.

3.3 Gestion Erreur tableaux

Il en existe d'erreurs contextuelles et des erreurs d'exécution lors de la manipulation des tableaux.

Les erreurs contextuelles:

- Déclaration de tableau avec un type qui n'existe pas. Message d'erreur : Le type <type>[][] n'est pas défini.
- Allocation de tableau avec dimension incompatible. Message d'erreur : Le type de l'expression de droite est <type>[] alors que le type attendu est <type>[][]
- Allocation de tableau avec des types incompatibles. Message d'erreur : Le type de l'expression de droite est <type>[][] alors que le type attendu est <otherType>[][]
- Accès au champs du tableau autres que size1D et size2D. Message d'erreur : Pour accéder au caratéristique du tableau, il utiliser les identificateurs suivants mettre size1D, size2D

- Accès au champ du tableau size2D pour une tableau 1D. Message d'erreur : Le tableau est 1D.

Les erreurs d'exécutions:

- int_allocaterveion_table_must_be_strictly_positive: déclenchée quand l'utilisateur essaie d'allouer de l'espace d'un tableau avec un indice négatif, par exemple int[] table = new int[-5];
- int_selection_table_must_be_positive : déclenchée quand l'utilisateur essaie d'accéder un élément du tableau avec des indices négatifs. Exemple println(table[-2]);
- table_dimension_are_not_respected : déclenchée quand l'utilisateur essaie d'accéder un tableau avec un indice plus large que la dimension du tableau. Exemple int[] table = new int[5]; println(table[15]);
- table_dimension_can_not_be_changed : déclenchée quand l'utilisateur essaye de changer la dimension d'un tableau en faisant par exemple int[] x = new int[5]; x.size1D = 5;

4 Utilisation de la bibliothèque Math

4.1 Qu'est-ce que la bibliothèque Math ?

La bibliothèque Math permet d'effectuer des calculs matriciels dans le langage de programmation deca. Chaque opération proposée par la bibliothèque est effectuée à partir de méthodes présentes dans un fichier *.decah. Pour utiliser ces méthodes dans un fichier *.deca, il est nécessaire de mettre" include "¡nom_du_fichier_¿.decah" en début de code (avant le main). Il existe un fichier *.decah pour chaque opération matricielle disponible, par exemple Sum.decah permet d'effectuer les calculs de somme de matrice. Ces différents fichiers sont présents dans le dossier src/main/resources/include.

Les opérations permises par cette bibliothèque sont détaillées ci-dessous.

Soient A = (a_ij) de taille (m,n), B = (b_ij) et C = (c_ij) de taille (m,n) où i, j, m, n sont entiers et

$$1 \leq i \leq m, 1 \leq j \leq n.$$

- Somme de matrices (fichier Sum.decah) Effectue la somme de matrices d'entiers et de flottants de même dimension. c_ij = a_ij + b_ij

Exemple:

#include "Sum.decah"

{

Sum matrixsum = new Sum();
int[][] a = new int[3][3];
float[][] b = new float[3][3];
matrixsum.setInt2DTable(a,5);
matrixsum.setFloat2DTable(b,6.0);
matrixsum.sommeIntFloat2D(a,b);
matrixsum.printInt2DTable(a);

} Dans un premier temps, on définit l'opérateur et les deux matrices opérandes. Les méthodes setInt2DTable permettent de remplir une matrice avec une valeur. Ensuite on calcule la somme que l'on affecte au premier paramètre de la méthode. Enfin, on affiche le résultat.

- Soustraction de matrices (fichier Difference.decah) Effectue la différence de matrices d'entiers et de flottants de même dimension.
  c_ij = a_ij - b_ij

- Produit de matrice membre par membre(fichier Multiply$_{m}$ember$_{b}$y$_{m}$ember.decah)Calculeleproduitmembreparmembred
c_ij = a_ij * b_ij

- Division de matrice membre par membre (fichier Divide$_m$ember$_b$y$_m$ember.decah)Calculeladivisionmembreparmembredemat
  c_ij = a_ij / b_ij
- Produit matriciel (fichier Produit$_M$atriciel$_N$aif.decah)Calculeleproduitmatricieldedeuxmatricesd′entiersoudeuxmatricesdeflotta
  Exemple:
  #include ”Produit$_M$atriciel$_N$aif.decah”
  {
  MatrixMultiplication matrixMult = new MatrixMultiplication();
  int[][] a = new int[3][2];
  int[][] d = new int[2][3];
  int[][] intRes = new int[3][3];
  matrixMult.setInt2DTable(a,5);
  matrixMult.setInt2DTable(d,3);
  a[0][0] = 1;
  a[2][1] = 9;
  d[1][2] = 4;
  d[1][0] = 0;
  matrixMult.printInt2DTable(intRes);
  }

  Cela définit une matrice de taille 3x2 et une autre matrice de taille 2x3 que l'on remplit respectivement de 5 et de 3. On modifie ensuite certaines valeurs de ces matrices et on calcule le produit matriciel que l'on affecte à intRes.

  - Calcul du déterminant (fichier Det.decah) Calcule le déterminant d'une matrice carrée de flottants ou d'entiers
  - Calcul de la trace (fichier Trace.decah) Calcule la trace d'une matrice de flottants ou d'entiers

– Inversion de matrice (fichier extension$_m$atricielle.decah)Calculel′inversed′unematriced′entiersoudeflottantsC = A^-1
  Exemple :
  include ”extension$_m$atricielle.decah”
  {
  float[][] matInver = new float[3][3];
  float[][] testMatInver ;
  matInver[0][0]=1;
  matInver[0][1]=2;
  matInver[0][2]=3;
  matInver[1][0]=0;
  matInver[1][1]=1;
  matInver[1][2]=4;
  matInver[2][0]=5;
  matInver[2][1]=6;
  matInver[2][2]=4;
  }
  testMatInver=mm.inversionFloat(matInver);
- Transposition (fichier extension$_m$atricielle.decah)Calculelatransposéed′unematriced′entiersoudeflottantsc_ij = a_ji
  Exemple :
  #include ”extension$_m$atricielle.decah”
  {float[][] matTranspo = new float[3][3];
  float[][] testMatTranspo ;

```
matInver[0][0]=1;
matInver[0][1]=2;
matInver[0][2]=3;
matInver[1][0]=0;
matInver[1][1]=1;
matInver[1][2]=4;
matInver[2][0]=5;
matInver[2][1]=6;
matInver[2][2]=4;
testMatTranspo = mm.transposeMatrixFloat(matTranpo);
}
```

//testMatTranpo est la matrice transposée de matTranpo, pour la transposé de la matrice int, la même procédure sauf qu'il faut changer la méthode transposeMatrixFloat au transposeMatrixInt.

- L'aplanissement de matrice (fichier extension$_m$atricielle.decah)retournerunematrice1Dquiestlaflattendelamatriced′entrée.Lar
  Exemple :

```
#include "extension_matricielle.decah"
{float[][] matFlatten = new float[3][3];
float[] testMatFlatten ;
matInver[0][0]=1;
matInver[0][1]=2;
matInver[0][2]=3;
matInver[1][0]=0;
matInver[1][1]=1;
matInver[1][2]=4;
matInver[2][0]=5;
matInver[2][1]=6;
matInver[2][2]=4;
testMatFlatten = mm.flattenMatrixFloat(matFlatten);
}
```

4.2 Limitations de l'extension

La bibliothèque Math ne permet pas de calculer le déterminant de n'importe quelle matrice. En effet, elle ne permet pas de calculer le déterminant de matrices qui comportent au moins un zéro sur sa diagonale le déterminant renvoyé est alors nul sans que le déterminant de la matrice le soit). Cela implique qu'il est également impossible de calculer l'inverse de ce type de matrice.

4.3 Gestion des Erreur

Des erreurs peuvent être envoyées si les dimensions des matrices ne sont pas respectées pour une opération matriciel. Par exemple, une erreur sera envoyée si une somme entre deux matrices de tailles différentes est effectuée.

Exemple:

```
#include "Sum.decah"
{
Sum matrixsum = new Sum();
int[][] a = new int[2][3];
float[][] b = new float[3][3];
matrixsum.setInt2DTable(a,5);
matrixsum.setFloat2DTable(b,6.0);
matrixsum.sommeIntFloat2D(a,b);
```

}

Les deux matrices ne sont pas de même taille donc le programme s'interrompt et renvoie un message d'erreur.

Les erreurs possibles sont les suivantes:

- Effectuer une somme avec deux matrices de taille différentes renvoie ce message d'erreur: "Les deux matrices doivent être de la même taille pour la somme de matrices"
- Effectuer une soustraction de deux matrices de tailles différentes renvoie ce message d'erreur: "Les deux matrices doivent être de la même taille pour la soustraction"
- un produit membre par membre de deux matrices de tailles différentes renvoie ce message d'erreur: "Les deux matrices doivent être de la même taille pour la multiplication membre par membre"
- Effectuer une division membre par membre de deux matrices de tailles différentes renvoie ce message d'erreur: "Les deux matrices doivent être de la même taille pour la division membre par membre"
- Effectuer un produit matriciel entre matrices de taille nxp et qxk où p != q renvoie le message d'erreur: "Taille de matrice incompatible avec le produit matriciel:p != q"
- Calculer le déterminant d'une matrice de taille nxp avec n!=p renvoie le message: "La matrice doit être carrée pour le calcul du déterminant: n != p"
- Calculer la trace d'une matrice de taille nxp avec n!=p renvoie le message: "Impossible de calculer la trace d'une matrice non carrée"
- Calculer l'inverse d'une matrice de taille nxp avec n!=p ou de déterminant nul renvoie le message: " dimension incompatible n!=p"
- Trianguler une matrice de taille nxp avec n!=p renvoie également le message d'erreur: " dimension incompatible n!=p"