Projet Génie Logiciel

Manuel d'utilisation

Réalisé par :
Mouez JAAFOURA, Ilyas MIDOU, Meriem KAZDAGHLI,
Salim KACIMI, Mehdi DIMASSI

Date : 09 janvier 2025

Grenoble INP - Ensimag & Phelma
Université Grenoble Alpes

Table des matières

1 Utilisation

1.1 Structure du projet

Le projet se compose de plusieurs modules :

- **src/main/java/fr/ensimag/deca/** : Contient le code source principal du projet.
- **src/test/deca/** : Contient les tests unitaires permettant de valider le bon fonctionnement du projet.
  - **syntax/** : Contient les tests unitaires pour la vérification syntaxique.
  - **context/** : Contient les tests unitaires pour la vérification contextuelle.
  - **gencode/** : Contient les tests unitaires pour la génération de code.
- **docs/** : Contient la documentation du projet ainsi que les fichiers de configuration.

1.2 Commandes disponibles

- **compilation** : Pour compiler un fichier `.deca` en fichier `.ass` en utilisant le module IMA. On utilise la commande suivante :

```
decac [[-p | -v] [-n] [-r X] [-P] <fichier deca>...] | [-b]
```

Option	Description
`-b`	Affiche la bannière du programme et termine l'exécution immédiatement.
`-p`	Arrête après l'analyse syntaxique et affiche l'AST déconstruit.
`-v`	Arrête après la vérification. Aucun résultat affiché s'il n'y a pas d'erreurs.
`-n`	Désactive les vérifications à l'exécution (comme le débordement ou les pointeurs nuls).
`-r <num>`	Limite le nombre de registres disponibles à `<num>` (valeurs entre 4 et 16).
`-P`	Active la compilation parallèle pour optimiser les performances.
`-d`	Active les traces de debug. Répétez cette option plusieurs fois pour augmenter le niveau de détail des traces.

TABLE 1 – Options de la commande `decac`

- **simulation** : Le fichier `.ass` généré sera situé dans le même répertoire que le fichier `.deca`. Pour lancer la simulation, exécutez la commande suivante :

```
ima path/to/file/fichier.ass
```

1.3 Exemple d'exécution

Voici un exemple d'utilisation de la commande de compilation :

```
$ ./src/main/bin/decac -p src/test/deca/codegen/valid/provided/entier1.deca
$ ima src/test/deca/codegen/valid/provided/entier1.ass
```

1.4 Script

Les scripts se trouvent dans le répertoire suivant : **src/test/script/**. Voici les principaux scripts disponibles et leurs fonctions respectives :

- `basic-synt.sh` : Utilise le script fourni par le professeur (**test_synt**) pour exécuter les tests syntaxiques sur notre base de tests valides et invalides :
  - **src/test/deca/syntax/valid/ourTests/** : Tests syntaxiques valides.
  - **src/test/deca/syntax/invalid/ourTests/** : Tests syntaxiques invalides.
- `basic-context.sh` : Utilise le script fourni par le professeur (**test_context**) pour exécuter les tests contextuels sur notre base de tests valides et invalides :
  - **src/test/deca/context/valid/provided/** : Tests contextuels valides.
  - **src/test/deca/context/invalid/provided/** : Tests contextuels invalides.
- `basic-gencode.sh` : Compile les fichiers de test avec notre compilateur avec l'option -P, puis exécute les simulations avec `ima` :
  - **src/test/deca/codegen/valid/ourTests/** : Tests sémantiques valides.
  - **src/test/deca/codegen/invalid/ourTests/** : Tests sémantiques invalides.
- `test-options.sh` : Teste toutes les options du compilateur decac sur notre base de tests valides invalides :
  - **src/test/deca/codegen/valid/ourTests/** : Tests sémantiques valides.
  - **src/test/deca/context/invalid/provided/** : Tests contextuels invalides.

2 Utilisation du langage Deca

Nous avons implémenté toutes les règles représentées dans le polycopié. Ainsi, dans notre langage, il est important de respecter l'ordre suivant :

- **Déclaration des classes** : Les classes doivent être déclarées en premier dans le fichier.
  sinon une erreur lexical indiquant la ligne et l'indice de caractere est sortie de la forme :

```
path/to/file/file.deca:ligne:indice: mismatched input 'class'

expecting {'instanceof','&&', ',', ')', '.', '=', '==', '>=',
```

```
'>', '<=', '<', '-', '!=', '||', '%', '+', '/','*'}
```

.

— **Déclaration du** `main` : Le point d'entrée du programme (`main`) doit suivre cet ordre :

  — **Déclaration des variables** : Voici les types disponibles dans le langage : `boolean`, `int`, `float`, `classes_iden` et les `tableaux` (voir 5.1).
  une declaration apres une ou plusieurs instruction provoque cette erreur.

```
path/to/file/file.deca:ligne:indice:
no viable alternative at input 'a'
```

  .

  — **Affectation des variables** : Les variables doivent être initialisées dans l'ordre de leur déclaration. Sinon, un comportement indéterminé peut se produire. Pour les tableaux (voir 5.2).

  — **Instructions disponibles** :

    — Les instructions d'affichage, telles que `print`, `println`, `printx`, et `printlnx`.

    — Les boucles `while(*condition) { ... }` : La condition peut inclure des affectations, des comparaisons ou des opérations arithmétiques, tant qu'elle retourne une valeur `boolean` à la fin. Sinon, une erreur contextuelle sera générée.

    — Les structures conditionnelles `if (*condition) { ... } else { ... }`.

  — **Restrictions dans le** `main` : Il n'est pas possible d'utiliser un `return` dans le `main`, car il est considéré comme une méthode de type `void`.

Note sur les conditions : Une condition peut inclure des affectations, des comparaisons ou des opérations arithmétiques, mais elle doit toujours retourner une valeur de type `boolean`. Sinon, une erreur contextuelle sera levée.

```
path/to/file/file.deca:ligne:indice:

Les opérandes doivent être de méme type.

Impossible de comparer entre type1 et type2.
```

.

3 Erreurs

3.1 Erreurs contextuelles et syntaxiques

Erreurs	**Explication**
(Type) valable seulement pour un type Boolean.	Une expression utilisée comme condition (par exemple dans un if ou un while) n'est pas de type Boolean

Table 2 – Type invalide pour une condition

Erreurs	**Explication**
L'opération '+ | - | * | /' n'est pas définie pour les **types** !	Vous essayez d'utiliser un opérateur arithmétique sur des types non compatibles, tels que les booléens ou les chaînes de caractères. Assurez-vous que les types des opérandes sont corrects et qu'ils prennent en charge les opérations arithmétiques.

Table 3 – Erreur de type pour une opération arithmétique

Erreurs	**Explications**
Les opérandes doivent être de même type. Impossible de comparer entre `type_gauche` et `type_droite`.	Cette erreur se produit lorsque vous essayez d'utiliser un opérateur de comparaison (comme `>`, `<`, `>=`, `<=`) sur des types non compatibles. Cela peut arriver dans les cas suivants : — Les opérandes de l'opérateur de comparaison sont de types différents et une conversion implicite n'est pas possible. — Les opérandes utilisés avec ces opérateurs ne sont pas des types numériques (comme `int` ou `float`).

Table 4 – Erreur de type pour les opérateurs de comparaison

Erreurs	Explications
Print est valable seulement pour les types : int, float et string.	Cette erreur se produit lorsque vous tente d'afficher une expression dont le type n'est pas pris en charge. Les types autorisés pour une instruction print sont : — int — float — string

Table 5 – Erreur de type pour les instructions print, println, printx, printlnx

Erreurs	Explications
On ne peut pas affecter à type_gauche un type type_droite.	Cette erreur se produit lorsqu'une expression tente d'affecter une valeur à une variable ou une structure dont le type n'est pas compatible. Par exemple : — Une affectation entre deux types incompatibles (e.g., string vers int). — Une affectation qui ne respecte pas la hiérarchie des sous-types (sauf pour les cas spécifiques comme int → float)

Table 6 – Erreur de type lors d'une affectation

Erreurs	Explications
la super-classe nom_super_classe n'est pas définie dans l'environnement.	Lors de la définition d'une classe, le compilateur vérifie que la super-classe spécifiée existe dans l'environnement des types. Cette erreur se produit si : — La super-classe n'a pas été déclarée ou définie avant l'utilisation. — Il y a une faute de frappe dans le nom de la super-classe.

Table 7 – Super-classe non définie

Erreurs	Explications
L'attribut 'nom_attribut' est défini plusieurs fois.	Cette erreur survient lorsqu'un champ (ou attribut) d'une classe est déclaré plusieurs fois avec le même nom dans la même classe. Cela viole la règle de l'unicité des champs dans une classe.

Table 8 – Attribut défini plusieurs fois

Erreurs	Explications
Le type d'un attribut ne peut pas être `type`.	Cette erreur se produit lorsque vous essayez d'attribuer un type non autorisé ou inexistant à un attribut. Cela peut arriver dans les cas suivants : — Le type spécifié n'est pas défini ou n'existe pas dans le contexte du programme. — Le type utilisé est réservé ou interdit (par exemple, des mots-clés spécifiques au langage). — Le type est mal orthographié ou mal déclaré. Solution : Vérifiez que le type spécifié est correctement défini et valide dans le contexte de votre programme.

Table 9 – Type d'attribut non autorisé

Erreurs	Explications
La méthode sous le nom nom_méthode est définie plusieurs fois.	Une méthode avec le même nom a déjà été déclarée dans l'environnement parent, ce qui entraîne un conflit.

Table 10 – Méthode redéfinie plusieurs fois

Erreurs	Explications
Le type `<nom_du_type>` n'est pas défini.	Cette erreur survient lorsque vous tentez d'utiliser un type qui n'est pas défini dans l'environnement des types. Cela peut arriver si le type n'a pas été déclaré ou est mal orthographié.

Table 11 – Type non défini pour un attribut

Erreurs	Explications
La variable `<nom_de_la_variable>` est définie plusieurs fois.	Cette erreur survient lorsqu'une variable est déclarée plusieurs fois dans le même contexte ou bloc de code. Chaque variable doit avoir un nom unique dans son contexte.

Table 12 – Variable redéfinie

Erreurs	Explications
Le paramètre sous le nom `<nom_du_parametre>` est défini plusieurs fois.	Cette erreur se produit lorsqu'un paramètre d'une méthode est défini plusieurs fois avec le même nom. Chaque paramètre dans une méthode doit avoir un nom unique.

Table 13 – Paramètre redéfini avec le même nom

Erreurs	Explications
Le type `<nom_du_type>` est inconnu.	Cette erreur se produit lorsque le type spécifié pour un paramètre de méthode ou une variable n'est pas défini ou est invalide. Cela peut arriver si le type n'est pas déclaré dans l'environnement ou si vous utilisez `void`, ce qui n'est pas autorisé pour les paramètres ou variables.

Table 14 – Type de paramètre inconnu ou non valide

Erreurs	Explications
La variable ne peut pas avoir `void` comme type.	Cette erreur survient lorsqu'une variable est déclarée avec le type `void`, ce qui n'est pas valide. Une variable doit toujours avoir un type concret pour stocker une valeur.

Table 15 – Type de variable non valide

Erreurs	Explications
On ne peut pas affecter à `type_gauche` un type `type_droite`.	Cette erreur se produit lorsqu'une expression tente d'affecter une valeur à une variable ou une structure dont le type n'est pas compatible.

Table 16 – Erreur de type lors d'une affectation

Erreurs	Explications
Le type d'un attribut ne peut pas être `void`.	Cette erreur se produit lorsque vous essayez d'attribuer un type non autorisé ou inexistant à un attribut.

Table 17 – Type d'attribut non autorisé

Erreurs	Explications
Le type `<nom_du_type>` n'est pas défini.	Cette erreur survient lorsque vous tentez d'utiliser un type qui n'est pas défini dans l'environnement des types.

Table 18 – Type non défini pour un attribut

Erreurs	Explications
On utilise Instanceof sur des classes qui ne sont pas définies dans l'environnement-Type.	Cette erreur se produit si les types utilisés avec l'opérateur instanceof ne sont pas des classes définies dans l'environnement des types. L'opérateur instanceof est réservé à la vérification de l'appartenance ou de la sous-classe d'un type objet.

Table 19 – Instanceof sur des types non valides

Erreurs	Explications
Un problème dans les paramètres d'entrée de la méthode redéfinie <nom méthode>. Attendue : <signature super classe> Présentée : <signature classe actuelle>	Les signatures de la méthode redéfinie dans la classe actuelle et dans la super-classe ne correspondent pas. Cela viole les règles de redéfinition des méthodes.

Table 20 – Problème dans les paramètres d'entrée de la méthode redéfinie

Erreurs	Explications
Lors de la redéfinition de la méthode <nom méthode>, le type <type super classe> n'est pas un sous-type de <type classe actuelle>.	Le type de retour de la méthode redéfinie dans la classe actuelle n'est pas un sous-type du type de retour de la méthode dans la super-classe.

Table 21 – Type de retour de la méthode redéfinie incompatible

Erreurs	Explications
L'index doit être de type int.	Cette erreur se produit lorsqu'une expression utilisée comme index pour accéder à un tableau n'est pas de type int. Les tableaux en Deca doivent être indexés par des entiers.

Table 22 – Index non valide pour un tableau

Erreurs	Explications
Expression Invalide. Valide seulement à l'intérieur d'une classe.	Cette erreur se produit lorsqu'une méthode est utilisée dans un contexte où aucune classe n'est définie (e.g., hors d'une classe ou dans un contexte global).

Table 23 – Expression invalide hors d'une classe

Erreurs	Explications
La méthode utilisée sous le nom <nom de la méthode> n'est pas définie dans l'environnement.	Cette erreur survient si la méthode appelée n'existe pas dans l'environnement de la classe actuelle. Cela peut être dû à : Une faute de frappe dans le nom de la méthode. Une méthode non déclarée dans la classe.

Table 24 – Méthode non définie

Erreurs	Explications
Expression Invalide !, valable seulement pour les classes ou pour les tableaux.	Cette erreur survient lorsque new est utilisé avec un type qui n'est ni une classe ni un tableau. Par exemple, utiliser new avec des types primitifs comme int ou float n'est pas valide.

TABLE 25 – Utilisation de new avec un type non valide

Erreurs	Explications
Not ne peut pas être appliqué à une expression de type <nom du type>. Applicable seulement pour les booleans.	Cette erreur se produit lorsque l'opérateur unaire Not (!) est appliqué à une expression qui n'est pas de type boolean. L'opérateur Not est exclusivement réservé aux expressions de type boolean.

TABLE 26 – Type de l'expression incompatible avec l'opérateur Not

Erreurs	Explications
return ne peut pas être utilisé à l'intérieur d'une instruction de type void	Cette erreur se produit lorsqu'une instruction return est utilisée dans une méthode déclarée avec un type de retour void. Dans une telle méthode, return ne peut pas être suivi d'une expression.

TABLE 27 – return dans une méthode de type void

Erreurs	Explications
On ne peut pas retourner un type <type expr> pour une méthode de type <type return>.	Cette erreur se produit lorsque le type de l'expression retournée ne correspond pas au type de retour attendu par la méthode.

TABLE 28 – Incompatibilité entre le type de retour et l'expression retournée

Erreurs	Explications
Invalide Type ! (<nom du type>) , l'utilisation des attributs est valable seulement pour les classes.	Cette erreur se produit lorsque l'expression située à gauche d'une sélection (.) n'est pas de type classe. La sélection d'attributs est réservée aux instances de classes.

TABLE 29 – Type de l'expression gauche d'une sélection invalide

Erreurs	Explications
La classe <nom classe> n'a pas d'attribut <nom attribut>.	Cette erreur survient lorsque l'attribut spécifié n'existe pas dans la classe ou ses superclasses.

Table 30 – Attribut inexistant dans la classe

Erreurs	Explications
l'identificateur This est utilisé seulement à l'intérieur d'une classe.	Cette erreur se produit lorsque l'identificateur this est utilisé en dehors du contexte d'une classe. L'utilisation de this est réservée pour référencer l'instance courante d'une classe dans une méthode ou un constructeur de cette classe.

Table 31 – This en dehors d'une classe

Erreurs	Explications
Cast invalide : impossible de convertir <type source> en <type cible>	Cette erreur survient lorsque le cast explicite entre le type de la variable et le type cible n'est pas autorisé. Les conversions valides sont limitées à : Identité (int → int, float → float, boolean → boolean). Conversion implicite (int → float). Conversion explicite (float → int).

Table 32 – Cast non autorisé

Erreurs	Explications
Type invalide ! (<nom du type>), valide pour un type int ou un type float.	Cette erreur se produit lorsque l'opérateur unaire UnaryMinus (opérateur -) est appliqué à une expression qui n'est ni de type int ni de type float. L'opérateur UnaryMinus est conçu pour être utilisé uniquement avec des types numériques.

Table 33 – Type de l'opérande invalide pour l'opérateur UnaryMinus

Erreurs	Explications
La classe sous le nom <nom du type> est déjà définie.	Cette erreur survient lorsqu'un tableau ou une liste est affecté(e) à une autre, mais que leurs dimensions (tailles) ne correspondent pas.

Table 34 – Incompatibilité des tailles dans une affectation de tableau

Erreurs	Explications
Problème dans les tailles des listes.	Cette erreur se produit lorsqu'une classe est déclaré plusieurs fois avec le même nom dans le même environnement. Chaque type doit avoir un nom unique.

Table 35 – Déclaration multiple d'une classe

Erreurs	Explications
La variable <nom de la variable> n'est pas définie.	Cette erreur se produit lorsqu'une variable ou un identifiant est utilisé sans avoir été déclaré ou défini au préalable dans l'environnement local. Cela signifie que le compilateur ne peut pas trouver de définition correspondante pour l'identifiant utilisé

Table 36 – Utilisation d'une variable non définie

Erreurs	Explications
Expression Invalide !, valable seulement pour les tableaux.	Cette erreur se produit lorsque l'expression sur laquelle une opération d'indexation est effectuée n'est pas de type tableau/liste. Les opérations d'accès par index (e.g., arr[0]) ne sont valides que pour des types définis comme tableaux

Table 37 – Type de la variable non conforme à une liste/tableau

Erreurs	Explications
Problème dans les dimensions du tableau.	Cette erreur se produit lorsque le nombre d'indices spécifiés pour accéder à un tableau multidimensionnel ne correspond pas aux dimensions du tableau.

Table 38 – Mauvaise dimension dans l'accès à un tableau multidimensionnel

Erreurs	Explications
Invalide Type ! (<type>), l'utilisation des méthodes est valable seulement pour les classes.	Cette erreur se produit lorsque vous tentez d'utiliser une méthode sur une expression qui n'est pas de type classe. Seules les instances de classes peuvent appeler des méthodes.

Table 39 – Utilisation de méthodes sur un type non conforme

Erreurs	Explications
L'attribut <nom attribut> est défini comme protected.	Cette erreur survient lorsqu'un attribut défini comme protected est accédé depuis une classe qui ne fait pas partie de la hiérarchie de la classe où l'attribut est défini.

Table 40 – Accès à un attribut protégé depuis une classe non autorisée

3.2 Erreurs de génération de code

Erreurs	Explications
Erreur : Modulo par zéro	Vous tentez d'effectuer une opération de modulo avec un entier divisé par 0. Cela n'est pas permis et génère une erreur.
Erreur : Division par zero	Vous tentez d'effectuer une opération de divison par 0. Cela n'est pas permis et génère une erreur.
Erreur : Débordement flottant	Cette erreur survient lorsqu'il y a un dépassement de capacité (*overflow*) lors d'une opération arithmétique sur des nombres flottants. Cela signifie que la valeur obtenue dépasse la plage autorisée pour un `float`, qui est comprise entre `-3.4028235E38` et `3.4028235E38`.
Erreur : Débordement de la pile	Vous avez dépassé la taille maximale de la pile, qui est limitée à 10 000 adresses.
Erreur : Erreur lecture	Cette erreur se produit lorsque vous entrez une valeur invalide pour une instruction `readInt` ou `readFloat`.
Erreur : Heap overflow	Un dépassement de mémoire *heap* (*heap overflow*) se produit lorsque un utilisateur essaye d'allouer trop de classes, trop de champs dans une classe.
Erreur : Déréférencement `null`	Une variable dont la valeur est null a été utilisée pour un appel de fonction ou pour accéder à un attribut.
Erreur : Return null dans une fonction non-void `null`	Vous avez oublié un return dans une méthode de type non-void.

Table 41 – Liste des erreurs d'exécution

4 Limitations

Notre compilateur présente les limitations suivantes :

— Il ne supporte pas la déclaration de plusieurs méthodes portant le même nom mais ayant des paramètres différents (surcharge de méthodes) dans la même classe. En effet, la vérification actuelle se base uniquement sur le nom de la méthode, sans prendre en compte les types ou le nombre des paramètres. Cependant, une redéfinition (override) d'une méthode dans une sous-classe est correctement prise en charge.

— Notre compilateur ne vérifie pas si le résultat d'un cast explicite `(int)` appliqué à une valeur flottante est dans les limites des valeurs maximales d'un entier. Cela peut entraîner une erreur lors de l'exécution du code généré par le compilateur. Par exemple :

```
{
    float x = 2147483647; // Valeur maximale pour un entier
    int c = 100;
    c = (int)(x + c);    // Cast explicite
    println(c);
}
```

Dans ce cas, le compilateur génère une erreur de type IMA lors de l'exécution :

```
** IMA ** ERREUR ** Ligne 39 :
    WINT avec R1 flottant
```

5 Extension

5.1 Présentation

En plus de l'implémentation du compilateur conformément aux spécifications du sujet, nous avons décidé d'enrichir le langage Deca en y ajoutant une extension dédiée à la prise en charge et à la manipulation des tableaux. Cette nouveauté élargit les possibilités offertes par le langage et le rend plus adapté à des besoins complexes. Pour la syntaxe, nous nous sommes inspirés des langages Java et C, afin de garantir une expérience intuitive et familière pour les développeurs.

De plus, nous avons conçu une bibliothèque spécifique pour le calcul matriciel, nommée `TAB.decah`. Cette bibliothèque propose une gamme d'outils permettant de réaliser des opérations sur des matrices flottantes, directement dans le langage Deca. Grâce à cette extension, Deca devient encore plus puissant et flexible pour les utilisateurs.

5.2 Mode Operatoire

Jusqu'à aujourd'hui, nous avons uniquement implémenté la partie A et B ainsi que la bibliothèque.

5.2.1 Déclaration d'un tableau

La déclaration d'un tableau se fait selon la syntaxe suivante :

— Le type du tableau, qui peut être `int`, `float`, `boolean` ou encore une classe personnalisée.

— Sa dimension, représentée par des paires de crochets vides directement après le type. Le nombre de paires de crochets correspond à la dimension du tableau.
— Enfin, son identifiant, qui est le nom du tableau.

```
<type> [] <identifiant>;
```

Par exemple, pour déclarer un tableau d'entiers à une dimension :

```
int[] tableau;
```

Et pour un tableau à deux dimensions (un tableau de tableaux) :

```
int[][] tableau2D;
```

5.2.2 Initialisation

L'initialisation d'un tableau se fait à l'aide de l'une des formes suivantes :
1ère forme : uniquement pour les types primitifs (`int`, `float`, `boolean`)

```
int[] tableau;
tableau = [1, 2, 3];
```

ou bien

```
int[] tableau = [1, 2, 3];
```

2ème forme : pour tous les types
L'index du tableau doit être un entier positif.
Tous les éléments du tableau sont initialisés comme suit :
— Pour les `int` et `float`, la valeur par défaut est `0`.
— Pour les `boolean`, la valeur par défaut est `false`.
— Pour les objets, la valeur par défaut est `null`.

Exemple :

```
int[] tableau = new int[3];
```

5.2.3 Assignation

Une fois des tableaux déclarés ou initialisés, il est possible de faire des assignations entre tableaux de même dimension. Par exemple, assigner les deux tableaux suivants :

```
int[] tableau1=[1, 2, 3];
int[] tableau2=new int[3];
tableau2=tableau1;
```

5.2.4 Accès aux éléments

La première case d'un tableau est indexée à `0`. Par exemple, pour changer la 2ème case d'un tableau et lui attribuer la valeur `1`, on peut faire comme suit :

```
int[] tableau = [1, 2, 3];
tableau[1] = 1;
```

5.3 Bibliothèque `TAB.decah`

La bibliothèque `TAB.decah` fournit des outils pour la gestion des matrices de flottants et des listes de flottants dynamiques dans le langage Deca. Elle comprend deux classes principales : `MatrixFloat` et `ListFloat`, chacune offrant une série de méthodes pour manipuler des matrices et des listes de manière flexible et efficace.

5.3.1 Classe `MatrixFloat`

La classe `MatrixFloat` est une classe utilitaire pour la gestion des matrices de flottants. Elle permet d'effectuer diverses opérations sur les matrices, telles que :

- **getElement(float[][] matrix, int row, int col)** : Cette méthode permet d'obtenir un élément spécifique dans une matrice 2D en fonction des indices de ligne `row` et de colonne `col`.
- **printMatrix(float[][] matrix, int rows, int cols)** : Affiche la matrice sous forme textuelle en ligne, séparant les éléments par des espaces. Les paramètres `rows` et `cols` spécifient le nombre de lignes et de colonnes de la matrice.
- **transpose(float[][] matrix, int rows, int cols)** : Effectue la transposition d'une matrice, échangeant ses lignes et ses colonnes. Les paramètres `rows` et `cols` spécifient la taille de la matrice.
- **trace(float[][] matrix, int rows, int cols)** : Calcule la trace de la matrice, c'est-à-dire la somme des éléments diagonaux. Les paramètres `rows` et `cols` spécifient le nombre de lignes et de colonnes.
- **determinant(float[][] matrix, int size)** : Calcule le déterminant d'une matrice, même pour les matrices de taille supérieure à 2x2, en utilisant la méthode de réduction de Gauss. Le paramètre `size` spécifie la taille de la matrice carrée.
- **multiply(float[][] A, int rowsA, int colsA, float[][] B, int rowsB, int colsB)** : Multiplie deux matrices `A` et `B`. Les paramètres `rowsA` et `colsA` spécifient les dimensions de la matrice `A`, tandis que `rowsB` et `colsB` spécifient celles de la matrice `B`.
- **add(float[][] A, int rowsA, int colsA, float[][] B, int rowsB, int colsB)** : Additionne deux matrices `A` et `B`. Les paramètres `rowsA`, `colsA`, `rowsB` et `colsB` spécifient les dimensions respectives des matrices.
- **subtract(float[][] A, int rowsA, int colsA, float[][] B, int rowsB, int colsB)** : Soustrait deux matrices `A` et `B`. Les paramètres `rowsA`, `colsA`, `rowsB` et `colsB` spécifient les dimensions des matrices.

— **printMatrix(float[][] matrix, int rows, int cols)** : affiche les elements de la matrice sous une forme lisible et structurée.

— **cmpMatrix(float[][] A, int rowsA, int colsA, float[][] B, int rowsB, int colsB)** : compare les deux matrices `A` et `B`. et renvoie true ou false. Les paramètres `rowsA`, `colsA`, `rowsB` et `colsB` spécifient les dimensions des matrices.

5.3.2 Classe `ListFloat`

La classe `ListFloat` implémente une liste dynamique de flottants. Elle permet de gérer une collection d'éléments de manière flexible, avec des méthodes pour ajouter, accéder, et supprimer des éléments. Les fonctionnalités incluent :

— **init()** : Initialise la liste avec une capacité de 10 éléments. Aucun paramètre.

— **add(float value)** : Ajoute un élément `value` à la fin de la liste. Si la capacité est atteinte, la liste est redimensionnée automatiquement.

— **get(int index)** : Récupère un élément à l'index `index`. Si l'index est hors limites, une erreur est générée.

— **remove(int index)** : Supprime un élément à l'index `index`. Si l'index est hors limites, une erreur est générée.

— **getSize()** : Renvoie la taille actuelle de la liste. Aucun paramètre.

— **quickSort()** : Effectue un tri rapide (Quicksort) sur la liste. Aucun paramètre.

— **binarySearch(float target)** : Recherche un élément `target` dans la liste en utilisant la recherche binaire. La liste doit être triée. Renvoie l'index de l'élément ou `-1` si non trouvé.

— **linearSearch(float target)** : Effectue une recherche linéaire pour un élément `target` dans la liste. Renvoie l'index de l'élément ou `-1` si non trouvé.

— **printList(float[] , int rows, int cols)** : affiche les elements de la liste sous une forme lisible et structurée.

— **cmpList(float[] A, int sizeA, float[] B, int sizeB)** : compare les deux Listes `A` et `B`. et renvoie true ou false. Les paramètres `sizeA`, `sizeB` et `colsB` spécifient les dimensions des Listes.

Cette bibliothèque est particulièrement utile pour les opérations mathématiques impliquant des matrices et des collections de nombres flottants. Elle permet d'effectuer des calculs complexes de manière simple et optimisée, tout en offrant une gestion dynamique des ressources avec les listes de flottants.

6 Conclusion

Ce rapport avait pour objectif de présenter l'état actuel de notre compilateur ainsi que de fournir quelques directives d'utilisation à un utilisateur potentiel. Bien que le projet soit déjà bien avancé et qu'il couvre un large éventail de fonctionnalités, certains

aspects demeurent perfectibles. Nous avons identifié plusieurs pistes d'amélioration et de corrections qui seront abordées dans les prochaines étapes du développement. Nous restons convaincus que ces ajustements permettront de renforcer la stabilité, l'efficacité et la convivialité de notre compilateur, tout en offrant une expérience optimale aux utilisateurs.