

Projet : Compilateur pour le langage MatC

Le but du projet est de réaliser un compilateur pour le langage MatC, depuis le code de haut niveau jusqu'à un code exécutable MIPS.

1 Langage MatC

Le langage MatC (pour « Matrix-C ») est un *domain specific language* (ou DSL) basé sur un langage C très simplifié qui a pour but de faciliter l'écriture de codes manipulant des matrices. Les opérations sur des matrices sont présentées en Section 1.1 et les détails du langage MatC sont présentés en Section 1.2.

1.1 Opérations sur matrices

Le langage MatC comporte des instructions dédiées aux matrices. Dans la suite, on définit une matrice comme étant un tableau à une ou à deux dimensions.

Soient les matrices A et B, MatC inclut les opérations suivantes :

1. Transposition : $\sim A$

La matrice $\sim A$ est la matrice transposée de A, c'est-à-dire la matrice où les lignes et les colonnes de A ont été inversées.

Exemple : Si $A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$, alors $\sim A = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}$.

2. Somme et différence : $A + B$ et $A - B$

La matrice $A + B$ est la matrice où les éléments des matrices A et B ont été additionnés élément par élément, c'est-à-dire où les éléments de même position dans chacune des matrices ont été additionnés.

Exemple : Si $A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$ et $B = \begin{pmatrix} 7 & 8 & 9 \\ 10 & 11 & 12 \end{pmatrix}$, alors $A + B = \begin{pmatrix} 8 & 10 & 12 \\ 14 & 16 & 18 \end{pmatrix}$.

3. Produit et division : $A * B$ et A / B

L'opération $A * B$ est le produit matriciel. Soient $a_{ij}, i = 1..n, j = 1..m$, les éléments de A, i étant l'indice de ligne et j l'indice de colonne. De même, soient $b_{ij}, i = 1..m, j = 1..p$, les éléments de B. La matrice résultat C est telle que ses éléments $c_{ij}, i = 1..n, j = 1..p$ sont définis par :

$$c_{ij} = \sum_{k=1..m} a_{ik} \times b_{kj}$$

Exemple : Si $A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$ et $B = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}$, alors $A * B = \begin{pmatrix} 14 & 32 \\ 32 & 77 \end{pmatrix}$.

Pour la division, il suffit de remplacer ' \times ' par '/'.

4. Extraction : $A[\text{intervalles}][\text{intervalles}]$

L'extraction permet, à partir d'une matrice, d'en créer une autre à partir de certaines lignes et colonnes de la matrice initiale. Les lignes et colonnes sélectionnées sont définies à partir d'une suite d'intervalles séparés par un point-virgule. Un intervalle est un indice de début et un indice de fin séparés par deux points, ou un indice seul.

Exemple : Si $A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}$, alors :

$$\begin{aligned}
A[0..1][*] &= \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{pmatrix}, \quad A[0..1][2..3] = \begin{pmatrix} 3 & 4 \\ 7 & 8 \end{pmatrix}, \\
A[0;2][0..1;3] &= \begin{pmatrix} 1 & 2 & 4 \\ 9 & 10 & 12 \end{pmatrix}, \quad A[0;0][0..1;1] = \begin{pmatrix} 1 & 2 & 2 \\ 1 & 2 & 2 \end{pmatrix}, \\
A[0][*;*] &= \begin{pmatrix} 1 & 2 & 3 & 4 & 1 & 2 & 3 & 4 \end{pmatrix}.
\end{aligned}$$

5. **Opération avec constante** : Chacune des opérations $*$, $/$, $+$ et $-$ peut aussi être utilisée avec une valeur constante. Cela provoque l'application de la constante à chaque élément de la matrice. L'opérateur unaire $-$ (comme dans $-A$), ainsi que $++$ et $--$ peuvent être utilisés.

Exemple : Si $A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$, alors $A * 2$ (ou $2 * A$) = $\begin{pmatrix} 2 & 4 & 6 \\ 8 & 10 & 12 \end{pmatrix}$.

1.2 Description du langage MatC

Le langage MatC est pour l'essentiel un sous-ensemble du langage C auquel on a ajouté un type matrice et des opérations sur les matrices.

1.2.1 Restrictions par rapport à C

Les restrictions par rapport à C sont les suivantes :

- Les seuls types possibles sont l'entier `int` et le réel simple précision `float`, ainsi que les tableaux multidimensionnels et les matrices de `float`, qui sont des tableaux de dimension 1 ou 2. Toutes les variables sont locales et statiques. Il est également possible de déclarer des constantes entières ou réelles.
- Les opérateurs possibles sont $+$, $-$ (unaire et binaire), $*$, $/$, $++$ et $--$.
- Les structures de contrôle possibles sont :
 - les conditionnelles `if` avec ou sans `else`,
 - les boucles `while`,
 - les boucles `for` qui fonctionnent comme des boucles itératives (celles du C sont en fait beaucoup plus générales) : la première partie correspond à l'initialisation d'un itérateur, la seconde à la condition d'arrêt de la boucle et la troisième à la mise à jour de l'itérateur.
- Les appels de fonctions, y compris récursivement, sont possibles. Cependant il vous est demandé de vous concentrer sur le reste avant de tenter d'ajouter le support des fonctions. Dans un premier temps, seule la fonction `main()` sans argument sera présente.
- La bibliothèque standard ne fournit qu'une fonction `printf()` simplifiée et uniquement capable d'afficher une chaîne de caractères ASCII. On dispose aussi des fonctions additionnelles `print()`, qui prend un entier ou un flottant en argument et l'affiche, et `printmat()`, qui affiche les éléments d'une matrice ligne par ligne, en séparant les éléments d'une ligne par une tabulation, et les lignes par un retour chariot. Aucune inclusion de bibliothèque n'est nécessaire pour cela.

1.2.2 Spécificités liées aux matrices

Les spécificités liées aux matrices sont les suivantes :

- MatC dispose du type `matrix`. Ce type permet de déclarer des tableaux de réels (type `float`) à une ou deux dimensions, qui pourront servir d'opérande dans les opérations sur les matrices. Pour déclarer une matrice, le nom d'identificateur doit être précédé du mot clé `matrix` et suivi d'un ou deux entiers entre crochets, comme pour la déclaration d'un tableau en C.

- Les opérations sur les matrices ne sont pas permises avec des opérandes qui ne sont pas de type `matrix`, ou des constantes dans le cas d'opérations avec constante.
- Les opérations sur les matrices devront donner lieu à un contrôle d'erreur rigoureux :
 - **Somme et différence** : Les opérandes doivent avoir le même nombre de lignes et de colonnes.
 - **Produit et division** : L'opérande de gauche doit posséder autant de colonnes que l'opérande de droite possède de lignes.
 - **Extraction** : Toute extraction incohérente par rapport au nombre de lignes et de colonnes de la matrice doit être détectée.

1.2.3 Exemple de programme en MatC

On donne à la figure 1 un exemple de programme qui calcule la solution X d'un système d'équations linéaires à deux équations et deux inconnues $AX=B$.

```
int main() {
    matrix A[2][2]={12,27},{64,42}, IA[2][2], B[2]={1,8}, X[2];
    float det;

    /* Calcul de la matrice inverse de A */
    IA = ~A;
    IA = IA[1;0][1;0];
    IA[0][1] = -IA[0][1];
    IA[1][0] = -IA[1][0];
    /* Calcul du déterminant de A */
    det = 1/(A[0][0]*A[1][1]-A[0][1]*A[1][0]);
    IA = det*IA;

    /* Calcul de X */
    X = IA*B;

    /* Affichage du résultat */
    printf("La solution de l'équation AX=B, avec A = ");
    printmat(A);
    printf("et B = ");
    printmat(B);
    printf("est X= ");
    printmat(X);
    printf("\n");

    return 0;
}
```

FIGURE 1 – Version MatC de résolution d'un système d'équations linéaires $AX=B$ en dimension 2

2 But du projet

L'objectif de ce projet est d'implémenter un compilateur pour le langage MatC, à écrire en C à l'aide des outils Lex et Yacc, qui produit un code exécutable MIPS correspondant aux programmes d'entrée. Ce travail est à réaliser en binôme et à rendre à la date indiquée par vos

enseignants en cours et sur Moodle. Une démonstration de votre compilateur sera faite durant la dernière séance de TP. Vous devrez rendre sur Moodle dans une archive :

- Le code source de votre projet complet dont la compilation devra se faire simplement par la commande « make ».
- Un document détaillant les capacités de votre compilateur, c'est à dire ce qu'il sait faire ou non. Soyez honnêtes, indiquez bien les points intéressants que vous souhaitez que le correcteur prenne en compte car il ne pourra sans doute pas tout voir dans le code.
- Un jeu de tests.

2.1 Assembleur MIPS

Le code généré devra être en assembleur MIPS R2000. L'assembleur est décrit dans les documents `spim_long.pdf` et `spim_short.pdf` accessibles sur Moodle. Des exemples de codes MIPS sont également fournis sur Moodle.

Le code assembleur devra être exécuté à l'aide du simulateur de processeur R2000 SPIM. Celui-ci est installé sur la machine turing. Pour exécuter un code assembleur, il suffit de faire : `spim <nom_du_code>.s`. Si vous désirez installer SPIM sur votre propre machine :

- Sous Ubuntu ou Debian, il suffit de faire : `apt-get install spim`, ou alors...
- Téléchargez le package `spim.tar.gz` sur Moodle
- `tar xvfz spim.tar.gz`
- `cd spim-8.0/spim`
- `./Configure` (Vérifiez qu'il n'y a pas de messages d'erreur : bibliothèque ou logiciel manquant.)
- Modifiez dans le fichier `Makefile` la ligne `EXCEPTION_DIR =` en y saisissant le chemin d'accès au répertoire contenant le fichier `exceptions.s`. Il s'agit du répertoire : `/chemin_vers_spim/spim-8.0/CPU`
- `make`

2.2 Précisions concernant la notation

- Si votre projet ne compile pas ou plante directement, la note 0 (zéro) sera appliquée : l'évaluateur n'a absolument pas vocation à aller chercher ce qui pourrait éventuellement ressembler à quelque chose de correct dans votre code.
- Si vous manquez de temps, préférez faire moins de chose mais en le faisant bien et de bout en bout : on préférera un support de MatC incomplet mais qui génère un code assembleur exécutable à une analyse syntaxique seule.
- Élaborez des tests car cela fait partie de votre travail et ce sera donc évalué.
- Faites les choses dans l'ordre et focalisez sur ce qui est demandé. L'évaluateur pourra tenir compte du travail fait en plus (optimisations du code généré par exemple) seulement si ce qui a été demandé a été fait et bien fait.
- Une conception modulaire et lisible sera fortement appréciée (et inversement). Une bonne démarche est de séparer les différentes étapes de la compilation comme cela a été vu en cours, en définissant des formats intermédiaires (par exemple on préférera une génération de code sous forme de quadruplets génériques (opérateur, opérande_cible, opérande_source1, opérande_source2), suivie de la traduction des quadruplets en assembleur, plutôt qu'une génération assembleur directe). Étant donnée l'ampleur de la tâche, on ne demande pas le passage par un arbre de syntaxe abstraite entre analyse syntaxique et génération de code, mais ceux qui le feront correctement seront des héros.

2.3 Recommandations

Conseil important : écrire un compilateur est un projet conséquent, il doit donc impérativement être construit **de manière incrémentale** en validant chaque étape sur un plus petit langage et en ajoutant progressivement des fonctionnalités ou optimisations (à ce propos, les optimisations du compilateur sont secondaires : si vous écrivez du code modulaire, il sera toujours possible par exemple de changer la représentation de la table des symboles en utilisant une structure de données plus efficace).

Une démarche extrême et totalement contre-productive consiste à écrire la totalité du code du compilateur en une fois, puis passer au débogage ! Le résultat de cette démarche serait très probablement nul, c'est-à-dire un compilateur qui ne fonctionne pas du tout ou alors qui reste très bogué.