Projet Programmation 1

Premier projet – Compilation d'expressions arithmétiques

Amélie LEDEIN Louis LEMONNIER Yoan GERAN

Octobre 2022

1. Le langage

Le but du projet est d'écrire un programme `aritha` qui traduit des expressions arithmétiques en code assembleur. La syntaxe des expressions à traduire est la suivante où `Int` représente les entiers (syntaxe `4`, `-4`, `+4`) et `Float` les flottants (syntaxe `4.0`, `-4.0`, `+4.0`).

$$\begin{aligned}\texttt{exp} :=\ & \texttt{Int} \mid \texttt{Float} \mid \texttt{(exp)} \mid \texttt{+(exp)} \mid \texttt{-(exp)} \mid \texttt{int(exp)} \mid \texttt{float(exp)} \mid \\ & \texttt{exp + exp} \mid \texttt{exp * exp} \mid \texttt{exp - exp} \mid \texttt{exp / exp} \mid \texttt{exp \% exp} \mid \\ & \texttt{exp +. exp} \mid \texttt{exp *. exp} \mid \texttt{exp -. exp}.\end{aligned}$$

La sémantique respecte les priorités opératoires usuelles. Ainsi, `1 + 2 * 3` correspond à $1 + 2 \times 3$ et non à $(1 + 2) \times 3$.

Le langage considéré est typé ; il contient des entiers et des flottants, et il n'y a pas de conversion implicite de l'un à l'autre. Pour passer d'un entier à un flottant (respectivement d'un flottant à un entier), il faut utiliser `float` (respectivement `int`).

Les opérateurs `+`, `*`, `-`, `/`, `%` et `float` ont en paramètres des entiers ; et les opérateurs `+.`, `*.`, `-.` et `int` ont en paramètres des flottants.

2. Consignes

Écrire un code OCamL traduisant une expression de ce langage vers le langage assembleur. Le programme prendra en entrée le nom d'un fichier (extension `.exp`) contenant une expression et le transformera en un fichier assembleur (extension `.s`). Ainsi, une exécution du programme ressemblera à la ligne de commande suivante, qui produit un fichier `expression.s`.

```
aritha expression.exp
```

Le code assembleur généré devra faire l'opération et afficher son résultat. Aucune opération arithmétique n'est faite dans le code OCamL (il ne s'agit pas de faire l'opération en OCamL et de générer un code assembleur qui affiche simplement le résultat).

Pour ce faire, vous passerez par une analyse syntaxique et lexicale qui vous fournira un AST. Puis vous vérifierez que l'expression est bien typée. Cet AST bien typé (appelé TAST) pourra alors être traduit en assembleur.

Chacune de ces étapes correspondra à un module de votre code ; vous aurez donc plusieurs fichiers. De plus, la traduction vers l'assembleur utilisera le module X86_64 [1] ; sa documentation est disponible ici [2].

3. Conseils

Ne vous y prenez pas au dernier moment et n'hésitez pas à poser des questions.

Pour compiler votre code assembleur, vous pourrez utiliser la ligne de commande suivante (elle génère l'exécutable `asm` à partir du fichier source `test.s`).

```
gcc -no-pie -o asm test.s
```

De plus voici également une commande pour traduire un code C en un fichier assembleur lisible.

```
gcc -S -fno-asynchronous-unwind-tables main.c
```

4. Modalités de rendu

Vous nous enverrez par mail (à nous trois, et pas à toute la promotion), un lien vers un projet Github (vous pouvez également nous rajouter en tant que collaborateur à ce projet). Votre dépôt contiendra :

- votre code **commenté** ;
- le code LaTeX de votre rapport ;
- une dizaine d'exemples variés ;
- un Makefile permettant de compiler votre code et votre rapport.

1. https://www.lri.fr/~filliatr/ens/compil/lib/x86_64.tar
2. https://www.lri.fr/~filliatr/ens/compil/lib/X86_64.html

Le Makefile produira un programe appelé `aritha` et un rapport appelé `rapport.pdf`.

Votre rapport contiendra les éventuelles difficultés rencontrées, et les choix potentiellement effectués. N'hésitez pas à indiquer les cas qui ne seraient pas correctement gérés par votre code. Ce rapport doit rester succint : si aucun problème n'a été rencontré et si aucun bonus n'a été implémenté, une dizaine de lignes suffiront.

Nous ferons passer à votre programme une batterie de tests. Votre note prendra en compte la quantité de tests passés, la qualité de votre code (notamment au niveau des commentaires), et celle de votre rapport (avez-vous bien su expliquer les problèmes rencontrés s'il y en a eu, etc.).

Le projet est à rendre avant le **dimanche 30 octobre 2022 à minuit**.

A. Bonus

A.1. Opérateurs supplémentaires

Vous pourrez ajouter la gestion d'autres opérateurs à votre programme. La factorielle et la puissance sont par exemple de bons candidats.

Vous indiquerez dans votre rapport les opérateurs que vous avez ajouté (et leur syntaxe) et donnerez des fichiers de test utilisant ces expressions.

A.2. Gestion de variables

Vous pourrez également rajouter une gestion simple de variables à votre programme. Nous vous proposons la syntaxe suivante où une ligne correspond à une déclaration de variables et la dernière ligne correspond à l'expression à évaluer.

```
x = 2
y = 2
(x + 2) * (y + 2)
```

Dans ce cas, vous expliquerez les choix effectués dans votre rapport, et là encore, vous donnerez quelques fichiers de test.