Rapport Projet 2
Où l'on parle de rongeurs

Guillaume Coiffier - Léo Valque

2017

Table des matières

Première partie

Généralités

Remarques générales

- **Niveau du binôme :** Intermédiaire
- **Adresse du dépôt Git :** `https://github.com/GCoiffier/Projet-2`
- Les fichiers de tests sont situés dans le dossier Programs. Les fichiers contenant du code fouine ont une extension `.ml`
- Les fichiers compilés pour la machine SECD sont situés dans le dossier Stack_Programs. Ils ont une extension `.code`

1 Comment exécuter notre programme

1.1 Avec Linux

- Pour compiler le programme, utilisez simplement la commande `make`. Celle-ci crée un exécutable appelé *fouine*.
- Pour nettoyer le répertoire de travail, utilisez la commande 'make clean'.
- `./fouine fichier` exécute le code contenu dans **fichier** et renvoie le résultat de ce code (qui doit être un entier)
- `./fouine -debug fichier` commence par afficher le code parsé dans la console, puis exécute le code et affiche le résultat.
- `./fouine -interm sortie fichier` compile le code parsé et le stocke dans sortie. Si aucun fichier de sortie n'est spécifié, le programme affichera le code dans la console.
- `./fouine -machine fichier` compile le code parsé et effectue l'interprétation mixte : ce qui peut être exécuté sur la machine à pile y est exécuté, le reste fait appel à l'interpréteur standard. Le code en entrée doit être un code fouine.
- `./fouine -execute fichier` compile le code parsé et l'exécute sur la machine à pile. Ce code doit être dans le langage de la machine à pile.
- **NB :** il est dans tous les cas possibles de ne pas donner de fichier d'entrée à fouine. Le programme s'éxecute alors en mode interactif et il faut entrer un programme dans la console.

Exemples :

```
> ./fouine
> ./fouine Programs/factorielle.ml
> ./fouine -debug Programs/function.ml
> ./fouine -interm
> ./fouine -interm toto.code Programs/prog1.ml
> ./fouine -execute toto.code
```

```
> ./fouine -machine Programs/prog2.ml
> ./fouine -debug Programs/function.ml
```

1.2 Avec un autre OS

1. Installez Linux
2. Reprendre les instructions de la section précédente.

2 Fouine

2.1 Fouine

Digression sur les fouines avec des photos mignonnes et tout.

2.2 Le langage Fouine

Ce langage ne figure cependant pas dans la liste des 700 langages de programmation proposée par Peter J. Landin [3].

3 Organisation du rapport et du projet

3.1 Organisation du rapport

Ce rapport s'organise en 3 parties, si l'on exclut cette partie d'introduction.
Dans un premier temps, nous parlerons de l'interpréteur fouine, de ses fonctionnalités et de certains aspects importants de son implémentation.
Dans un second temps, nous parlerons de la machine à pile SECD, également implémentée par nos soins.
Enfin, la dernière partie sera consacrée à l'exécution mixte fouine/SECD d'un code.

3.2 Organisation du projet

3.3 Avancement du projet en fonction du temps

Rendu 2

- **Semaine 1**
  - Lexer et parser de base
  - Définition d'un type programme pour les programmes fouine
  - Interprétation des fichiers fouine sans les fonctions (ie expressions arithmétiques, booléennes, `if ... then ... else` et `let ... in`)
- **Semaine 2**
  - Interprétation des fichiers fouine avec des fonctions
  - Interprétation des fichiers fouine avec des fonctions récursives
- **Semaine 3**
  - Gestion des exceptions
  - Gestion des références
  - Gestion du `begin...end` et du `let _ = ...`
  - Ajout de primitives "bonus" `prStr` (qui renvoie 0 et affiche une string) et `prNl` (qui passe une ligne et renvoie aussi 0)

Rendu 3

- Semaine 4
  - Correction des exceptions (on utilise plus `try ... with` de Caml)
  - Implémentation des tableaux
  - Compilation des expressions arithmétiques vers une machine à pile
  - Exécution des expressions arithmétiques sur une machine à pile
- Semaines 5 et 6
  - Corrections de bugs du retour du rendu 2 (réferences, ordre d'execution des fonctions, exceptions)
  - Travail sur le main. Le programme peut désormais lire l'entrée standard dans le cas où on ne donne pas de fichier en argument
  - Extension de la machine à pile qui gère les variables et les branchements conditionnels

Rendu 4

- Semaine 6
  - Ajout d'un lexer, parser pour les instructions de la machine à pile, on peut desormais compiler puis executez plus tard
  - Ajout des fonctions dans la machine à pile
- Semaine 7
  - Ajout des fonctions recursives dans la machine à pile
  - Implémentation des indices de bruijn dans la machine à pile
  - Ajout de l'interpréteur mixte qui envoie sur la machine quand il peut et sinon éxecute normalement
- Semaine 8
  - Correction de bugs sur l'interprétation mixte. Implémentation du transfert d'environnement entre fouine et SECD
  - Rédaction du présent rapport.

Deuxième partie

Le projet

Organisation du code

3.4 Liste des fichiers

3.5 Liste des programmes fouine donnés en exemple

3.6 Bugs repérés mais non corrigés

4 L'interpréteur fouine

4.1 Notre langage fouine

4.2 Structures de données

4.3 Les exceptions

4.4 Aspects impératifs et tableaux

5 La machine à pile SECD

5.1 Présentation

5.2 Implémentation

6 Interface et interprétation mixte

Conclusion

On constate que blibla.

Références

[1] Eddy Caron, editor. *Recueil des projets intégrés de l'année*, volume 1 of *Best sellers du DI*. ENS de Lyon, 2016.

[2] Les M1 du DI. Notre beau projet intégré. In Caron [1], pages 10–22.

[3] Peter J. Landin. *The next 700 programming languages*, volume 9. 1966.