Rapport projet 2, où l'on parle de rongeurs, ou De la difficulté de travailler avec des personnes moins compétente que soit

Emile Martinez et accessoirement Mathias Berry

1 Introduction

Flexion, liée, jeu, et paf on introduit mais ce rapport et non le ballon.
Ce rapport est donc à propos du projet fouine effectué par Emile Martinez et Mathias Berry dans le carde de l'UE de la L3 de l'ENS de Lyon *Projet de Programmation* qui a pour but de nous faire coder en CamL une sous partie du langage associé CamL. Nous commencerons au paragraphe 2 par détailler les ontéractions à l'intérieur du binôme avant de détailler très succintement parce que ce n'est pas intéressant (étant la même pour tout le monde) au paragraphe 3 la structure de notre projet, suivi au paragraphe 4 des particularités de notre code et enfin au paragraphe 5 nous ferons, comme son nom l'indique, et en toute originalité, un bilan.

2 Mathias ou De l'éducation

Le binôme était constitué de Mathias Berry et de Emile Martinez.
Ce binome composé de deux personnnes extrêmement sympathique était quelque peu hétérogène. En effet, la notable différence de QI mena à biens des conflits. Néanmoins, Emile a commencer à inculquer à Mathias quelques rudiments de bon comportements informatique, comme le fait de mettre des commentaires dans ses programmes, de git pull avant de commencer à écire, de ne pas modifier frénétiquement les fichiers sur lesquels sont censé travailler les autres, etc ... avec cependant des succès modérés.

De plus, l'aride communication entre nous a pu parfois, notamment pour le rendu 3 occasionner un peu de précipitation à rendre notre travail et donc entraîner des imprécisions, Mathias Berry ayant échoué à faire une programmation propre.

3 Structure du code

Le code est structuré de la manière suivante :

- ⋆ `main.ml` faisant le lien entre les fichiers
- ⋆ `lexer.ml` et `parser.ml` ayant pour but de traduire en un type Caml une chaine de caractère issue d'un fichier texte représentant le programme à exécuter.
- ⋆ `expr.ml` définissant le type représentant un programme et la fonction pour l'afficher
- ⋆ `eval.ml` pour évaluer les expressions.
- ⋆ `type.ml` qui génére les contraintes sur les types
- ⋆ `resoud.ml` qui unifie les types monomorphement et participe à gérer l'affichage des types

4 Particularités de notre code

4.1 Gros point positif

La première chose qui est un gros atout de ce projet, et qui lui est propre, est qu' Emile Martinez a participer à son élaboration, et cela, peu peuvent s'en vanter. Quand il sera président du monde, les lignes du code source de ce projet pourront se revendrent à prix d'or.

4.2 Autre point positif, négligeable par rapport au premier

Déjà, il a une tête à marcher, et ça c'est super. Ensuite, quelques détails ont était poffinées, comme par exemple le traitement de la commande

```
let rec f =
    let rec a = fun x ->
    if x = 0 then 1
             else a(x-1) + f(x-1)
    in a
in prInt (f 10)
```

4.3 Ce qui ne va pas vraiment

Ce qui ne va pas c'est que Mathias Berry est mauvais, ce qui est quand même dommage. Emile Martinez a bien essayé de rattraper cela mais ca a était vain. Et plus sérieusement, il y a notamment un problème, qui concerne le typage. En effet, comme on ne teste pas l'égalité entre des types mais qu'on unifie le type référencer par un entier avec un autre type, quand on veut en comparer

deux, il faut utiliser un nouvel entier. Comme on stocke tous dans un tableau, il faudrait connaitre à l'avance le nombre de telle comparaison, ce qui n'est pas aiser (voire même très dur). Par conséquent, on a juste mis un tableau très grand, en espérant ne voir aucun gugus essayer de faire tourner un vrai programme long sur notre fouine.

5 Bilan

On est vraiment très fort, surtout si l'on compare à ce que peut produire le milieu académique [1] même si en l'occurence je n'ai pas lu ce document, mais je le sais car je suis omniscient.

Références

[1] Peter J. Landin. The next 700 programming languages. *Commun. ACM*, 9(3) :157–166, 1966.