université PARIS-SACLAY | FACULTÉ DES SCIENCES D'ORSAY

Projet IPF
LDD2 IM

Récapitulatif séance 1

Objectif

On partage ici quelques questions qui ont été soulevées par plusieurs personnes

1 Binomage

Le binomage est la règle par défaut pour tout le monde. En effet, il faut trouver un équilibre entre
— avoir un nombre raisonnable de projets à corriger
— que le projet permette de développer des compétences de collaboration
— qu'il y ait suffisamment à faire pour chacun

On veut donc éviter un trop grand nombre de personnes seules (par choix) et éviter à des personnes de se retrouver seules sans trouver de binôme. Les trinômes sont exclus, car sinon il n'y a pas assez à faire pour chaque membre. Un formulaire sera prochainement mis en ligne sur eCampus pour permettre à chacun de donner son nom, celui de son binôme et l'URL du projet FramaGit utilisé, ou le cas échéant d'indiquer que vous n'avez pas de binôme et en cherchez un, ou que vous souhaitez faire le projet seul (soumis à autorisation).

2 Framagit

Tout le monde devrait avoir activé son compte Framagit et ajouté les trois utilisateurs mentionnés dans la feuille 1 comme membres développeurs (si vous ne le faites pas, nous ne pourrons pas ramasser vos projets !). Vous pouvez prendre un moment durant la séance 2 pour faire les questions Git de la feuille 1 qui était restées en suspens.

3 Fichiers .mli et utilisation de `dune`

Lors qu'un fichier `.mli` existe (en plus d'un fichier `.ml`), il donne la liste de toutes les valeurs « exportées » par le `.ml` et donc utilisables. Généralement, le fichier `.mli` ne laisse abstraits certains types (pas de possibilité de connaître leur représentation interne) et certaines fonctions auxiliaires.

L'utilitaire `dune` essaye de construire le programme `huff.exe`. Pour cela, il analyse le fichier `huff.ml`. Il regarde en particulier tous les modules utilisés par ce dernier (par exemple `Huffman`) et va compiler tous les fichiers `.ml` et `.mli` correspondants (par exemple `hufmann.ml`). Une conséquence est que si vous modifiez un fichier `.ml` mais que ce dernier n'est pas une dépendance de `huff.ml` alors il ne sera pas recompilé. Ça peut être problématique, en particulier pour `heap.ml` et `heap.mli`. Une solution pour cela est de faire, dans le fichier `huff.ml`

```
1 open Heap
2 (* ... reste du fichier *)
```

Cela forcera le fichier `heap.ml` à être compilé. Bien sûr, quand votre projet sera avancé, le fichier `hufmann.ml` contenant l'algorithme de compression fera naturellement appel à `Heap` et sera alors une dépendance. Vous pourrez alors retirer le `open Heap` du fichier principal.

4 Lecture des caractères d'un fichier

Beaucoup de groupes ont écrit la fonction de lecture d'un fichier sous cette forme :

```
1
2  let rec stats file =
3    let cin = open_in file in
4    let rec loop () =
5       try
6          let b = input_byte cin in
7            (* ... faire quelque chose avec b *)
8          loop ()
9       with End_of_file ->
10         (* fin de fichier, renvoyer le resultat final *)
11   in loop ()
```

Ceci est problématique car l'appel récursif `loop ()` n'est pas en position terminale (car la fonction pourrait « revenir » pour faire le `with`). Une façon de contourner le problème est la suivante :

```
1
2  let input_code cin =
3     (* une fonction auxiliaire qui gère directement l'exception *)
4     try
5       input_byte cin
6     with End_of_file -> -1
7
8  let rec stats file =
9      let cin = open_in file in
10     let rec loop () =
11           let b = input_byte cin in
12           if b < 0 then
13             (* fin de fichier, renvoyer le résultat final *)
14           else
15           let .. = ... (* ... faire quelque chose avec b *) in
16           loop ()
17
18   in loop ()
```