Projet de programmation, Rendu 1

tableur

Rendu à faire en binôme, à rendre pour le ***lundi 1er février 2021 à 23h59*** *au plus tard. Téléversez une archive qui compile dans la partie "Premier rendu : un tableur" de la page du portail des études.*

Un tableur est un programme permettant de manipuler un tableau dans lequel on stocke et calcule des formules. Nous manipulerons un tableau bidimensionnel de *cellules*, et on supposera pour simplifier que toutes les valeurs que l'on manipule sont des flottants.

La fonctionalité essentielle que l'on cherche à implémenter est la possibilité de modifier une cellule du tableau, ce qui entraîne de recalculer (au moins) l'ensemble des cellules impactées par cette modification.

Le programme prendra en entrée une suite de *commandes*, dont voici un exemple :

```
1  B2=A1
2  Show A4
3  C1=SUM(D1;MULT(D2;D3;D4))
4  ShowAll
```

On a des commandes pour afficher soit une cellule, soit l'ensemble du tableau.

À la ligne 1, on recopie la *valeur* de la case `A1` en `B2` (pas la formule, puisque la formule en `B2` c'est `A1`).

La sortie du programme consiste uniquement en l'affichage déclenché par les appels à `Show` et `ShowAll`. À noter qu'il y a une différence entre une valeur non définie (qui sera affichée avec un *souligné* _) et zéro.

N.B. : pour les gens qui excellent en manipulation de tableurs, il n'y a pas vraiment lieu de faire la différence entre référence absolue et référence relative dans le contexte de ce projet.

1 Version *débutants*

Vous trouverez sur la page du portail des études une archive contenant plusieurs fichiers, qui sert de point de départ pour ce rendu. En principe, les explications fournies en commentaire et dans le fichier README devraient être suffisantes.

N'hésitez pas à nous faire signe par mail[1] si certains points vous semblent obscurs, ou si certaines décisions vous font hésiter.

Votre travail consiste à (dans l'ordre ! Ne passez à $n+1$ que lorsque n marche) :

1. Traiter la mise à jour d'une cellule, qui engendre le (re)calcul de toute la feuille.

   Tout se passe dans `sheet.ml`. Il vous faudra lire et comprendre l'ensemble des fichiers avant de coder.

2. Dans `command.ml`, le traitement de `Show` et `ShowAll` dans la fonction `run_command` fait un appel systématique à `recompute_sheet`. Modifiez le code de façon à éviter de tout recalculer lorsqu'aucune cellule n'a été modifiée. En revanche, on doit recalculer *à chaque modification*, et non pas à chaque affichage.

3. Ajouter la fonction MAX parmi les opérateurs offerts. Cela signifiera aller éditer les fichiers `lexer.mll` et `parser.mly`. Les modifications sont suffisamment simples pour que vous puissiez faire cela "à l'aveugle", on éclaircira en cours de semestre les captivants mystères liés à ces fichiers.

   L'idée c'est d'aller voir ce qui se passe pour `SUM` (ou pour `MULT`), et de faire pareil.

[1] `amelie.barbe@ens-lyon.fr, etienne.miquey@ens-lyon.fr, daniel.hirschkoff@ens-lyon.fr`

***.** Tout au long du développement, enrichir le répertoire `tests/` de fichiers de tests que vous aurez faits vous-mêmes, pour vérifier *au fur et à mesure* (et pas à la fin !) que les choses marchent comme prévu.

Caml et Linux. Si vous êtes débutant(e), ce rendu est aussi l'occasion de vous dégourdir en ce qui concerne l'utilisation de Caml sous Linux. Quelques remarques dans cette optique sont rassemblées sur cette page.

Voir aussi la partie 4 ci-dessous pour les instructions pour le rendu.

2 Version *intermédiaires*

Votre travail consiste à (comme pour les débutants, ne passez à $n+1$ que si n marche et est testé) :

1. Faire ce qui est demandé pour les débutants.
2. Faire une seconde version du programme, où l'on gère les dépendances entre cellules, de manière à éviter si possible de tout recalculer lorsqu'on modifie une cellule.

   Il vous faudra reprendre le type associé aux cellules, afin d'y ajouter de quoi gérer les dépendances entre cellules.

   NB : les modifications doivent être déclenchées par la mise à jour, et non par l'affichage. Même s'il n'y a pas de `Show` ou `ShowAll`, on modifie quand même.

   (à partir du moment où cette version plus sophistiquée du programme fonctionne, ce qui est bien sûr souhaitable, inutile d'inclure la version plus simple correspondant au "menu débutants")
3. Ajouter une fonctionnalité au programme : si la mise à jour d'une cellule engendre une boucle (par exemple, faire `A1=B2` suivi de `B2=A1`), on ignore la commande qui engendrerait une boucle et on passe à la suivante.
4. Ajouter une option à l'exécutable : si on lance l'exécutable `main.native` avec l'option `-paf`, si une boucle est introduite, on arrête l'exécution du programme et on affiche `PAF` (en majuscules, sur une ligne, en début de ligne, en allant à la ligne après le `PAF` : merci de respecter cette consigne).

   Vous trouverez ici un petit exemple vous donnant des indications pour la gestion des options[2].

3 Version *avancés*

Vous devez faire ce qui est demandé aux intermédiaires, avec en plus :

1. **Nombres.** Remplacer les flottants par des nombres (et gérer souplement les conversions, de manière à pouvoir additionner un entier et un flottant sans que cela pose problème) ;
2. **Intervalles.** Ajoutez la possibilité de manipuler des intervalles dans le tableau. Par exemple, on pourra saisir `SUM(A2;B5:B7)`, ce qui permettra d'éviter d'écrire `SUM(A2;B5;B6;B7)`.
3. **Plusieurs Tableaux.**

   Ajoutez la possibilité de gérer plusieurs tableaux (ou "feuilles de calcul"). La convention est que ceux-ci s'appellent `s1`, `s2`, ..., jusqu'à mettons `s10` (et la numérotation des tableaux commence à 1).

   ```
   SwitchTo s2
   B1=SUM(A1;A3;MULT(A2;A1))
   SwitchTo s1
   Show C4
   ```

   Cette extension est notamment utile pour l'extension suivante.

   N.B. : il ne vous est pas demandé de pouvoir faire des références entre tableaux, i.e., pas de `A4=SUM(s2.A4,19)`.

[2] Ne changez pas la manière dont l'exécutable récupère le fichier : on reste sur `./main.native < tests/t1.txt`. Pour ce faire, passez la fonction `(fun s -> ())` à `Arg.parse`, et faites ensuite un appel à `parse()`.

4. **Fonctions.**

   Ajoutez la possibilité de traiter des tableaux comme des fonctions, ce qui permettra d'écrire des choses comme :

   ```
   D1=s2(B1;B3)
   ```

   On adoptera les conventions suivantes : par défaut, toutes les fonctions de ce style ont exactement deux arguments (qui sont deux cellules, pas des intervalles). Lorsqu'une fonction est appelée, la valeur des arguments est recopiée dans les cases `A2` et `A3`, et le résultat est calculé dans la case `A1`. Autrement dit, si on fait `D1=s2(B1;B3)` dans `s1`, cela aura pour effet de recopier la valeur de `B1` et `B3` depuis `s1` vers les cases `A2` et `A3` de `s2`, de recalculer `s2`, et de recopier la valeur contenue dans la case `A1` de `s2` dans la case `D1` de `s1`.

   Vous pouvez aussi envisager la généralisation naturelle aux fonctions à n arguments.

   Pensez à tester comme il faut cette fonctionnalité.

   Une fois que ce qui est demandé ci-dessus marche, et a été testé, vous pouvez vous lâcher et étendre le langage avec des choses exotiques (fonctions récursives, d'ordre supérieur, exceptions, modules, monades, types dépendants, ...). Ou aussi lancer une fenêtre dans lequel on peut observer la mise à jour des valeurs dans le tableau au fur et à mesure que le script est exécuté.

   Si vous êtes avancés en Caml, vous pouvez consulter cette page et vous en inspirer (mais ne changez pas les manières d'interagir avec le programme ! cf. ci-dessous).

   Aparté, à titre informatif. L'extension proposée au point 4 est peut-être un peu déroutante; elle vient de travaux comme celui-ci, qui a conduit à des choses comme ça (on en donne ici une version très simplifiée).

   L'idée de départ est que les tableurs sont utilisés, de manière plus ou moins intuitive, par plein de gens qui n'ont pas l'ambition de savoir programmer. Cet outil n'est pas loin d'être un langage de programmation, et, pour des utilisations plus poussées, se voit l'objet de greffes avec des langages plus ou moins recommandables. L'approche promue ici est d'ajouter *à l'intérieur même du tableur* (sans "greffe") une notion absolument fondamentale en programmation, à savoir les fonctions (ou procédures).

4 Tout le monde : en quoi consiste le rendu

Vous devez téléverser avant la date limite une archive qui compile.

Important : votre programme devra *respecter scrupuleusement* les formats en entrée et en sortie exactement comme ils sont donnés dans le programme qui est disponible sur la page www du cours. N'inventez pas des variantes, il est important pour nous que les tests puissent être menés de manière uniforme au moment d'évaluer vos programmes.

Incluez dans votre archive un fichier README, contenant des commentaires sur votre code : indiquez notamment s'il y a des parties que vous n'avez pas traitées, s'il y a des bugs/imperfections dont vous êtes conscients — c'est bien mieux de lire dans le README ce qui ne marche pas que de le découvrir en interagissant avec votre programme.

On ne le répètera jamais assez : il est très préférable de rendre quelque chose de propre, qui ne traite pas toutes les questions, plutôt que quelque chose qui essaie de tout traiter, mais est en vrac et ne marche pas. En particulier, il n'est pas question d'envoyer quelque chose qui ne compile pas[3]. Cela peut signifier dans certains cas que vous devez "couper une branche" qui marche presque presque presque au moment de

[3] évitez d'utiliser des versions ultra sophistiquées de Caml, qui ne marchent que sur la machine des membres d'une secte de votre choix.

la date limite, et où il reste juste un petit bug dont vous êtes sûrs qu'il est minuscule, mais vous n'avez pas eu le temps de le trouver.

"Propre" signifie en particulier :

- du code bien structuré et bien commenté.

  Il ne faut pas commenter chaque ligne ni toutes les fonctions, mais il n'est pas envisageable d'avoir un fichier avec uniquement du code. Les parties (les plus) pertinentes doivent être commentées, et le commentaire s'insère *avant* le code auquel il s'applique — à l'extrême, un fichier de 8 lignes avec juste des fonctions utilitaires pour manipuler des couples d'entiers comporte un commentaire en tête de fichier du style `(* fonctions utilitaires pour manipuler des couples d'entiers *)`

- des programmes testés, avec les fichiers de test que vous avez utilisés (il faut qu'il y en ait bien plus que ce qui est fourni dans l'archive initiale) ;

- un fichier README.