Devoir maison n°9

⋆ ⋆ ⋆

Transpilation : fiche d'aide

On découpe le travail en plusieurs parties. On suggère de gérer la lecture et écriture de fichier en dernier (car on peut faire des tests sans ces étapes).

1 Analyse lexicale

On suggère de travailler avec le type suivant :

```
type lexeme =
    | Init
    | Print
    | While
    | End
    | Plus
    | Minus
    | Equal
    | Var of string
    | Nat of string
```

À noter, on n'est pas obligé de garder les `Nat` sous forme d'entiers, car il faudrait de toute façon les reconvertir en chaîne en écrivant le programme C.

L'analyse lexicale lira alors le programme comme une chaîne de caractères et renverra une liste de lexèmes.

Pour le découpage, on pourra utiliser :

- `String.map` pour remplacer chaque caractère blanc par un caractère espace ;
- `String.split_on_char` pour découper les chaînes entre deux espaces consécutives ;
- `List.filter` pour ne garder que les lexèmes non vides.

2 Analyse syntaxique

On voit un programme comme une suite d'instructions. Cela donne l'idée de la grammaire suivante, sur l'ensemble de variables $\{Prog, Instr\}$, et sur l'alphabet de terminaux {init, print, plus, minus, equal, while, end, str} :

$$\begin{array}{lll} Prog & \to & \varepsilon \mid Instr\ Prog \\ Instr & \to & \text{init str} \mid \\ & & \text{print str} \mid \\ & & \text{plus str} \mid \\ & & \text{minus str} \mid \\ & & \text{equal str str} \mid \\ & & \text{while str } Instr\ Prog \text{ end} \end{array}$$

À noter, ici le terminal str représente soit le nom d'une variable, soit un entier naturel. Évidemment, l'analyse syntaxique se chargera de vérifier que seul le deuxième argument de egal peut être un entier naturel.

On suggère de créer un arbre d'analyse syntaxique avec les types suivants, dont les constructeurs correspondent aux terminaux de la grammaire précédente (on distingue print et plus), et les `string` correspondent aux variables ou entiers :

```
type prog =
    | V (* programme vide *)
    | Prog of instr * prog
and instr =
    | I of string
    | Pr of string
    | Pl of string
    | M of string
    | E of string * string
    | T of string * instr * prog
```

On peut alors procéder par analyse syntaxique descendante, en écrivant deux fonctions auxiliaires mutuellement récursives, une pour dériver *Prog*, et l'autre pour dériver *Instr*.

3 Conversion en C

L'avantage est que les instructions du langage OCalme se traduisent plutôt simplement en C. On commencera par ajouter l'entête et écrire la déclaration de la fonction `main`, puis on traduira l'arbre d'analyse syntaxique.

Pour la lisibilité, on peut choisir de marquer les indentations (mais ce n'est pas obligatoire) : on peut garder en mémoire un compteur qui croit à chaque `while` et décroit à chaque `end`.

Ensuite, pour chaque instruction, imbriquées comme il faut avec les `while`, on concatène l'instruction équivalente en C.

4 Lecture et écriture de fichier

On peut se reporter aux consignes dans le TP13 pour la lecture de fichier. On se contentera de concaténer toutes les lignes pour former la chaîne correspondant au programme. Attention, il faudra rajouter des espaces entre les concaténations, car les retours à la ligne auront disparu.

Pour l'écriture de fichier, on pourra utiliser `open_out` pour ouvrir un fichier en écriture et `output_string` pour écrire dedans. On se réferrera à la documentation pour la syntaxe et l'ordre des arguments.

⋆ ⋆ ⋆