Parsing et interprétation

Objectifs

— Utilisation des parsers fonctionnels.
— Application à l'écriture d'un interprète LOGO.

On utilisera le fichier source `main.ml` fourni, contenant des éléments permettant la réalisation des exercices de la séance.

1 Le langage LOGO

On souhaite définir un parser pour une variation du langage LOGO, qui permet de réaliser des dessins, à travers l'exécution de commandes destinées à un curseur graphique (la "tortue" LOGO). On propose la grammaire suivante :

$P \rightarrow$ `begin` I `end`	$C \rightarrow$ `move entier`
$I \rightarrow \Lambda$	$C \rightarrow$ `turn entier`
$I \rightarrow C\,;I$	$C \rightarrow$ `on`
$C \rightarrow$ `repeat entier` P	$C \rightarrow$ `off`

Les non-terminaux P et I désignent respectivement les programmes et les suites d'instructions séparées par des points-virgules. C désigne une commande, soit : la répétition d'un programme P un certain nombre de fois ; le déplacement de la tortue d'une distance donnée dans la direction courante ; le changement de direction courante d'un certain angle (en degrés), l'écriture sur la feuille (au fur et à mesure des déplacements) ou non. Le terminal `entier` désigne les constantes entières non-signées, les autres terminaux étant des mots-clés du langage. Enfin, derrière tout programme se trouve la fin de fichier.

Voici les types utilisés (fournis) pour décrire les programmes LOGO :

```
(* Le type des programmes LOGO *)
type prog = inst
and inst = cmd list
(* Le type des commandes LOGO  *)
and cmd =
| Repeat of int * prog      (* on repete n fois un programme                    *)
| Move of int               (* on se deplace de d pas dans la direction courante *)
| Turn of int               (* on tourne d'un angle a                           *)
| On                        (* on pose le stylo sur la feuille                  *)
| Off                       (* on leve le stylo                                 *)
```

▷ **Exercice 1 (Parsers)**
Voici les types attendus des parsers pour les différents non-terminaux de la grammaire ci-dessus.

```
(* parser associe a P *)
val parse_P : (char, prog) parser
(* parser associe a I *)
val parse_I : (char, inst) parser
(* parser associe a C *)
val parse_C : (char, cmd) parser
```

1. *Définir les parsers. On utilisera les fonctions auxiliaires fournies, notamment celles du module* `Parser`*, pour composer les parsers. Le parsing des terminaux est fourni.*
2. *Tester vos parsers à l'aide de la fonction* `test_parser_logo` *fournie.*

Maintenant, il est temps d'exécuter les programmes LOGO. L'état de la tortue LOGO, modifié par les commandes exécutées, est défini par un quadruplet `(on, x, y, a) : bool * float * float * float` tel que `on` détermine si le stylo est posé ou non, `x` et `y` représentent la position (exprimée en nombres flottants) et enfin `a` est l'angle de la direction courante.

▷ **Exercice 2 (Application)**
En utilisant la fonction `run_logo : prog -> unit` *fournie et d'autres fonctions, définir un programme principal qui lit un programme LOGO depuis un fichier et l'exécute.*

2 Extension du langage LOGO

On souhaite étendre la syntaxe du langage LOGO en y ajoutant la déclaration de sous-programmes (sans paramètres), qui peuvent se trouver au début de n'importe quel bloc `begin` ... `end`. La grammaire est modifiée de la façon suivante, où `ident` représente les identificateurs tandis que `proc` et `call` sont des nouveaux mots-clés.

$P \to$ `begin` $D\ I$ `end`	$C \to$ `repeat entier` P
$I \to \Lambda$	$C \to$ `move entier`
$I \to C\,;I$	$C \to$ `turn entier`
$D \to \Lambda$	$C \to$ `on`
$D \to S\,;D$	$C \to$ `off`
$S \to$ `proc ident` P	$C \to$ `call ident`

Le type des programmes LOGO sera modifié comme suit :

```
(* un programme est une séquence de déclarations suivi d'une séquence d'instructions *)
type prog = decls * inst
and decls = decl list
and decl =
  Decl of string * prog      (* declaration d'un sous-programme nomme     *)
and inst =
⋮
and cmd =
| Call of string             (* appel du sous-programme par son nom       *)
⋮
```

▷ **Exercice 3 (Parsers pour langage LOGO étendu)**
Définir de nouvelles fonctions de parsing *pour prendre en compte les nouvelles constructions.*

▷ **Exercice 4 (Interprète pour langage LOGO étendu)**
Modifier les fonctions d'évaluation pour prendre en compte les nouvelles constructions. On pourra par exemple ajouter à l'état de la tortue l'ensemble des sous-programmes connus. Cet ensemble est en fait manipulé comme une pile.