Objectif · **Prérequis** · **Travail à faire** · **Rendu**

1 Objectif

ELM est un langage dérivé d'Haskell et compilable en javascript. Il est particulièrement adapté à la création d'une application web monopage. Le but de ce projet est de développer en ELM une application permettant de visualiser le dessin produit par des commandes de tracé données par l'utilisateur. Cet exemple peut être considéré comme un modèle à reproduire, voire à améliorer.

2 Prérequis

Mener à bout ce projet nécessite des prérequis.

2.1 ELM

En suivant les premiers chapitres de ce guide, vous pourrez en quelques heures vous familiariser avec le langage.

La syntaxe de ELM est proche de celle d'Haskell et ne devrait pas trop vous troubler. La principale différence est l'utilisation des quatre points `::` pour l'opérateur d'ajout à une liste, et des deux points `:` pour la signature de type (ce qui est l'exact opposé d'Haskell). Le reste est identique et vous pourrez à tout moment consultez ce récapitulatif.

Au fur et à mesure de votre progression, vous aurez aussi besoin des références suivantes :

- https://package.elm-lang.org/packages/elm/core/latest/Basics
- https://package.elm-lang.org/packages/elm/core/latest/List
- https://package.elm-lang.org/packages/elm/core/latest/String
- https://package.elm-lang.org/packages/elm/html/latest/
- https://package.elm-lang.org/packages/elm/svg/latest/
- https://package.elm-lang.org/packages/elm/parser/latest

2.2 HTML, CSS, SVG

HTML décrit le contenu d'une page web, tandis que CSS décrit son apparence. SVG décrit des dessins vectoriels, pouvant être directement inclus à une page web. Vous n'aurez pas à programmer vous-même directement dans l'un de ces langages. Il vous suffit de reprendre le code source de l'exemple et sa feuille de style. En revanche, vous devrez appeler certaines fonctions ELM qui auront pour effet de créer dynamiquement les éléments correspondant dans la page. La connaissance de HTML et SVG est donc un plus. Au minimum, assurez-vous que vous sachiez utiliser les fonctions suivantes dans de petits programmes :

- `div`, `input`, `button`, `text` du package Html,
- `svg`, `line` du package Svg.

2.3 TcTurtle

Enfin, vous devez comprendre le langage que doit interpréter votre application. C'est un langage inventé au département et inspiré des Turtle graphics. Il s'appelle donc naturellement *TcTurtle*. Il permet d'exprimer le chemin que suit un crayon pour dessiner. Il comporte 4 instructions, toutes avec paramètres : **Forward**, **Left**, **Right** et **Repeat**. L'instruction `Forward x` fait avancer le crayon de `x` unités dans la direction courante. L'instruction `Right x` (respectivement `Left x`) fait tourner la direction courante de `x` degrés à droite (respectivement à gauche). L'instruction `Repeat x [ yyy ]` répète `x` fois la suite d'instructions entre crochets. Les instructions sont séparées par des virgules et le programme entier se trouve entre crochets et sur une seule ligne de texte. Voici quelques exemples de programme :

- `[Repeat 360 [ Right 1, Forward 1]]`
- `[Forward 100, Repeat 4 [Forward 50, Left 90], Forward 100]`
- `[Repeat 36 [Right 10, Repeat 8 [Forward 25, Left 45]]]`
- `[Repeat 8 [Left 45, Repeat 6 [Repeat 90 [Forward 1, Left 2], Left 90]]]`

Utilisez cette application pour visualiser les dessins qu'ils encodent.

3 Travail à faire

Voici quelques indications pour vous permettre de mener à bien ce projet.

3.1 Coeur du projet

Votre programme peut être réparti en trois modules :

- le programme principal qui décrit votre modèle de page et implémente les fonctions `init`, `update` et `view` de l'architecture ELM,
- le module de parsing qui fournit une fonction `read` prenant en entrée une chaîne de caractères et retournant une structure de données que vous aurez définie pour représenter les programmes *TcTurtle*. Vous aurez besoin des fonctions `succeed`, `token`, `int`, `spaces`, `lazy`, `run` du package Parser,
- le module de dessin qui traduit votre stucture de données en un élément svg.

Pour définir une structure de données adéquate et coder le dernier module, vous êtes bien sûr encouragé à reprendre ce programme Haskell.

3.2 Améliorations

Si vous êtes arrivé à une application fonctionnelle sans difficulté, voici quelques pistes d'améliorations possibles :

- offrir à l'utilisateur la possibilité de maîtriser son crayon : couleur, largeur, état (levé ou abaissé).
- traiter les exceptions liées à la position et à la taille du dessin par rapport à la feuille : calculer la position de départ et l'unité en fonction de l'étendue ou donner la possibilité de déplacer, agrandir, rétrécir le dessin par rapport à la feuille.
- fournir des indications détaillées permettant à l'utilisateur de corriger son programme quand il n'est pas syntaxiquement correct.
- ...

4 Rendu

Il doit être réalisé par équipe de 4 à 6 étudiants. Il doit être rendu sous la forme d'une archive au format *zip* contenant l'ensemble du projet. Veillez à inclure les noms et prénoms de tous les membres de l'équipe, ainsi que les explications nécessaires à la bonne compréhension du code et à la compilation. L'archive doit être déposée sur moodle au plus tard le jour de l'examen final, 23h55.