Thème 1 - TME 2 - Un petit projet Stack complet

Sorbonne Université - Master Informatique M1 - STL

MU4IN510 Programmation Avancée en Fonctionnel - 2021

Dans ce deuxième TME nous développons un petit projet complet dans l'environnement de développement Stack.

Il s'agit d'un petit utilitaire nommé `textstat`, en ligne de commande, qui prend en entrée un fichier texte encodé en *Unicode* (utf-8) et qui retourne quelques statistiques sur le fichier.

Par exemple, si on prend en entrée le roman *Quatrevingt-Treize* de Victor Hugo, disponible grâce au projet Gutemberg à l'URL suivante: https://www.gutenberg.org/cache/epub/9645/pg9645.txt On obtiendra la sortie suivante :

```
$ stack run
Lecture du fichier: pg9645.txt
Nombre de lignes de texte = XXXX
Nombre de mots = YYYYY
Les 10 lettres les plus fréquentes :
 - E = ZZ
 - A = TT
 ...
```

Préliminaires

Nous entrons, avec ce TME, dans le monde complexe du *devops*. Pour que tout se passe bien, nous devons effectuer quelques vérifications. Le premier prérequis est d'être suffisamment à l'aise avec le *shell* (Bash ou Powershell) dans le terminal.

La première chose à vérifier est que l'on dispose de *Stack* dans une version récente :

```
$ stack --version
Version 2.5.1 ... blabla ...
```

Création du projet

La première étape de ce TME consiste à générer le squelette des sources du projet. Pour cela nous allons utiliser l'outil `stack` avec la commande suivante :

```
$ stack new textstat --resolver lts-17.1
Downloading template "new-template" to create project "textstat" in textstat/ ...

... etc ...

Writing configuration to file: textstat/stack.yaml
All done.
```

Remarque : j'ai utilisé un *resolver* explicite, ici `lts-17.1` qui est la dernière version stable (et compatible avec `17.0` que nous avons utilisé précédemment, ce sont juste des correctifs de bugs.

Stack émet un certain nombre d'avertissements, notamment parce que certaines méta-données, comme les coordonnées du mainteneur du projet (vous!) ne sont pas saisies par défaut. Pour l'instant nous allons ignorer ces problèmes mais il faudra combler cela pour la version finale. Les deux effets principaux de la commande ci-dessus sont les suivants :

1. création du répertoire de configuration globale de Stack dans `$HOME/.stack` (uniquement au premier lancement de `stack`)
2. création de l'arborescence du projet dans le répertoire `textstat/`

Le répertoire `$HOME/.stack` contient tous les fichiers nécessaires au bon fonctionnement de l'outil Stack. On y trouve en particulier le fichier de configuration globale `config.yaml` dans lequel on peut éditer les méta-données qui concerne l'ensemble des projets. Dans ce répertoire Stack installe également toutes les dépendances nécessaires à la construction des projets, et il peut donc devenir de taille très importante (plusieurs gigaoctets même pour des projets avec quelques dépendances!).

Arborescence du projet

Pour chaque projet, la commande `stack new <nom-du-projet>` créer un répertoire du `<nom-du-projet>` avec une arborescence complète. Dans notre cas, un répertoire `textstat/` a été créé avec :

— à la racine `textstat\` les fichiers de configuration spécifiques au projet, notamment :
  — le fichier `README.md` de description du projet (au format *markdown*), la `LICENSE` (BSD3 par défaut, mais les TME de PAF sont *tous droits réservés* car ne nous voulons pas de diffusion des corrections), etc.
  — le fichier `stack.yaml` qui contient la configuration locale de l'outil Stack. Nous n'aurons pas à modifier ce fichier, en particulier si on a bien indiqué le *resolver*.
  — le fichier `package.yaml` qui contient la configuration du projet avec notamment les dépendances et les cibles de *build*
— dans le répertoire `src/` les sources principales du projet, avec un fichier d'exemple `Lib.hs`
— dans le répertoire `app/` les sources du point d'entrée pour les projets de programme, avec un fichier d'exemple `Main.hs`
— dans le répertoire `test/` les sources des tests unitaires ou de propriétés, avec un fichier d'exemple (non-informatif) `Spec.hs` (nous reviendrons dans les prochains TME sur la problématique des tests)

Le projet ainsi créé est vide mais tout de même fonctionnel, ce qui nous permet de découvrir les commandes principales de Stack.

Commandes principales de Stack

— **Construction du projet**

La commande pour construire le projet est la suivante :

```
$ stack build
```

Si tout se passe bien, vous aurez des affichages ressemblant à ceci :

```
$ stack build
Building all executables for `textstat' once. (...)
textstat> configure (lib + exe)
Configuring textstat-0.1.0.0...
textstat> build (lib + exe)
Preprocessing library for textstat-0.1.0.0..
Building library for textstat-0.1.0.0..
[1 of 2] Compiling Lib
[2 of 2] Compiling Paths_textstat
Preprocessing executable 'textstat-exe' for textstat-0.1.0.0..
Building executable 'textstat-exe' for textstat-0.1.0.0..
[1 of 2] Compiling Main
[2 of 2] Compiling Paths_textstat
Linking stack-work/dist/<...blabla...>
textstat> copy/register
Installing library in (...)
Installing executable textstat-exe in (...)
Registering library for textstat-0.1.0.0..
```

Remarque : tout ce qui est compilé spécifiquement pour ce projet le sera dans le répertoire `stack-work/`, qui peut grossir dans d'importantes proportions.

— **Lancement du programme principal**

Vous pouvez lancer le programme principal par la commande suivante :

```
$ stack run
someFunc
```

Il est aussi possible de lancer les commandes par leur nom:

```
$ stack exec textstat-exe
someFunc
```

(ce sera utile lorsque nous installerons des outils d'aide au développement)

— **Lancement des tests**

Pour lancer les tests (pour l'instant aucun), on utilise la commande suivante :

```
$ stack test
textstat> test (suite: textstat-test)
(...)
Test suite not yet implemented

textstat> Test suite textstat-test passed
```

— **Lancement de l'interprète de commandes (REPL)**

Pour lancer GHCI dans le cadre du projet, on utilise la commande suivante :

```
$ stack ghci
Using main module: 1. (...) main-is file: (...)/app/Main.hs
The following GHC options are incompatible with GHCi (...)
Configuring GHCi with the following packages: textstat
GHCi, version 8.4.4: http://www.haskell.org/ghc/  :? for help
[1 of 2] Compiling Lib              ( (...)/textstat/src/Lib.hs, interpreted )
[2 of 2] Compiling Main             ( (...)/textstat/app/Main.hs, interpreted )
Ok, two modules loaded.
Loaded GHCi configuration from /tmp/haskell-stack-ghci/(...)/ghci-script
*Main Lib Paths_textstat>>
```

On peut utiliser les commandes comme `:load` de GHCI sans se soucier de l'organisation des sources du projet, des modules disponibles, etc.

— **Nettoyage des sources**

L'arborescence du projet, notamment les fichiers générés dans `./.stack-work/`, peut être nettoyée par la commande suivante :

```
$ stack clean --full
```

Important : n'oubliez pas l'option `--full` sinon le volumineux répertoire `.stack-work/` n'est pas nettoyé

— **Génération d'une archive des sources du projet**

Pour générer une archive des sources du projet, il existe une commande `stack sdist` mais qui n'intègre pas le projet de développement en entier. Il faudra donc plutôt créer une archive manuellement en commençant par nettoyer le projet avec `stack clean --full` puis en archivant le dossier.

Installation d'une dépendance : la bibliothèque `text`

Les chaînes de caractères Haskell, par défaut, sont de simple listes de caractères, soit le type `[Char]` (et son alias `String`). C'est un choix historique qui conserve un intérêt pédagogique mais qui n'est pas réaliste en pratique. La

bibliothèque `text` (cf. https://hackage.haskell.org/package/text) permet de manipuler du texte au standard Unicode, et est aujourd'hui un standard de fait.

Pour installer la dépendance, nous devons ajouter au fichier `package.yaml` la ligne suivante dans la section `dependencies` :

```
dependencies:
- base >= 4.7 && < 5
- text
```

Remarque : il est recommandé de borner les numéros de version, ce qui pourra peut-être indiquer quelques problèmes potentiels lors d'un changement de *resolver* (lorsque le projet évolue...). Ici, nous n'indiquons pas de contraintes car nous sommes dans le cadre d'un projet pédagogique.

Pour vérifier que la dépendance fonctionne, modifiez le fichier `Main.hs` de la façon suivante :

```haskell
-- Option de compilation pour que les chaînes litérales
-- soient surchargées plutôt que fixées à `[Char]`
{-# LANGUAGE OverloadedStrings #-}

module Main where

-- Le type `Text` est disponible
import Data.Text (Text)

-- Les fonctions de manipulation de textes
-- sont préfixées par `T` plutôt que `Data.Text`
import qualified Data.Text as T

-- Les fonctions d'entrées sorties pour les textes
import qualified Data.Text.IO as TIO

import Lib

main :: IO ()
main = do
  TIO.putStrLn $ T.reverse "Hello world"
```

et relancez le *build*...

```
$ stack build
...
$ stack run
dlrow olleH
```

Pour pouvoir coder le projet, vous aurez besoin de consulter la documentation de la bibliothèque. Voici le lien correspondant : https://www.stackage.org/haddock/lts-17.1/text-1.2.4.1/Data-Text.html

(j'utilise un lien vers la version correspondant au *resolver* `lts-17.1`)

Le projet textstat

Vous pouvez maintenant commencer à coder le projet, qui consiste à afficher des statistiques sur un fichier lu sur le disque (cf. exemple en début de sujet).

Pour cela, on suivra les recommandations suivantes :

1. le coeur des analyses sera codé en *fonctionnel pur* dans un module `Stats` dans le fichier `src/Stats.hs`
2. on modifiera le `Main.hs` pour coder tout ce qui concerne les entrées/sorties (lecture du fichier, affichages). Aucun calcul ne deva être effectué dans le module `Main`, ce qui ne veut pas dire qu'il faut tout écrire dans la fonction `main` (on peut bien sûr créer des fonctions auxiliaires).

Remarque: on détruira le fichier `src/Lib.hs` ainsi que toutes les références au module `Lib`.

Défi : tests unitaires avec HSpec

Pour finaliser le projet il faudrait ajouter des tests unitaires pour les fonctions pures de votre projet (tester la partie IO nous intéresse moins). Pour cela, consultez la documentation sur l'intégration des tests dans Stack avec HSpec, ainsi que la document de HSpec proprement dit.

Important : les défis sont des exercices *facultatifs* prévus pour les étudiant.e.s qui veulent en fait un peu plus, en autonomie. Nous aborderons les tests unitaires avec HSpec dans le prochain TME donc il s'agit ici de prendre un peu d'avance.