# String Builders
# UE IPFL – ENSIIE – S2

Mars 2022

## 1 Informations générales

### 1.1 Travail à rendre

Le projet est à réaliser en OCaml individuellement. Il sera accompagné d'un dossier contenant impérativement la description des choix faits, la description des types et des fonctions. Même si le sujet est décomposé en questions, il est possible qu'une question se résolve par l'écriture d'une ou plusieurs fonctions intermédiaires. Le dossier fournira également des cas de tests accompagnés des résultats attendus et retournés. Pour cela, vous pouvez utiliser des assertions. Attention, le code doit être purement fonctionnel.

### 1.2 Calendrier et procédure de remise

Vous devez rendre votre rapport (en pdf) et le(s) fichier(s) de code rassemblés dans une archive identifiée comme `votre_prenom_votre_nom.tgz`. Une archive qui ne contient pas les fichiers demandés ne sera pas excusable. Votre archive doit être déposée sur `http://exam.ensiie.fr`, dans le dépôt `ipf_projet_2022`, avant 23h59 le 13/05/2021.

## 2 Enoncé du projet

La majorité des langages de programmation fournissent une notion primitive de chaîne de caractères. Si ces chaînes s'avèrent adaptées à la manipulation de mots ou de textes relativement courts, elles deviennent généralement inutilisables sur de très grands textes. L'une des raisons de cette inefficacité est la duplication d'un trop grand nombre de caractères lors des opérations de concaténation ou l'extraction d'une sous-chaîne.

Or il existe des domaines où la manipulation efficace de grandes chaînes de caractères est essentielle (représentation du génome en bio-informatique, éditeurs de texte, ...). Ce projet propose une alternative à la notion usuelle de chaîne de caractères que nous appelons `string_builder`. Un `string_builder` est un arbre binaire, dont les feuilles sont des chaînes de caractères usuelles et dont les noeuds internes représentent des concaténations.

Ainsi le `string_builder` dans Fig.1 représente le mot `GATTACA`, obtenu par concaténation de quatre mots `G`, `ATT`, `A` et `CA`. L'intérêt des `string_builder` est d'offrir une concaténation immédiate et un partage possible de caractères entre plusieurs chaînes, au prix d'un accès aux caractères un peu plus coûteux.

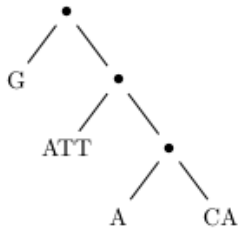


Figure 1: Exemple de `string_builder` (Attention: version simplifié dans laquelle la longueur des caractères stockés dans les noeuds et feuilles n'apparaît pas)

Un `string_builder` est donc soit un mot (feuille), soit une concaténation de deux autres `string_builder` (noeud). **Note :** Pour des raisons d'efficacité, **on conserve dans les feuilles, aussi bien que dans les noeuds, la longueur** $length(c)$ **de la chaîne de caractères** $c$ **correspondante**.

## 2.1 Échauffement

**Question 1.**

- Définir le type `string_builder`.
- Définir la fonction `word` qui prend en argument une chaîne de caractères et qui renvoie le le `string_builder` constitué d'une seule feuille correspondant.
- Définir la fonction `concat` qui prend en argument deux `string_builder` et qui renvoie le nouveau `string_builder` résultant de leur concaténation.

**Question 2.**

- Définir une fonction `char_at` qui prend en argument un entier $i$ et un `string_builder` représentant le mot $[c_0; ...; c_{n-1}]$, et qui renvoie le caractère $c_i$. **Note :** On supposera $0 \leq i < n$, et on utilisera la fonction `String.get` pour obtenir le $i^e$ caractère d'une chaîne de caractères.

**Question 3.**

- Définir une fonction `sub_string` qui prend en arguments un entier $i$, un entier $m$ et un `string_builder` $sb$ représentant le mot $[c_0; ...; c_{n-1}]$ et qui renvoie un `string_builder` représentant le mot $[c_i; ...; c_{i+m-1}]$, c'est-à-dire la sous-chaîne de $c$ débutant au caractère $i$ et de longueur $m$.
  **Note :** On supposera $0 \leq i < i + m \leq n$, et on s'attachera à réutiliser dans le `string_builder` résultant autant de sous-arbres de $sb$ que possible.

## 2.2 Équilibrage

Le hasard des concaténations peut amener un `string_builder` à se retrouver déséquilibré, c'est-à-dire à avoir certaines de ses feuilles très éloignées de la racine et donc d'accès plus coûteux. Le but de cette partie est d'étudier une stratégie de rééquilibrage à posteriori. Considérons un `string_builder` $sb$ composé de $k+1$ feuilles, et donc de $k$ noeuds internes. Notons ces $k+1$ feuilles $m_0, \ldots, m_k$. Lorsqu'on les considère de la gauche vers la droite, si bien que $sb$ représente le mot $m_0 m_1 \ldots m_k$. La profondeur de la feuille $m_i$ dans $sb$ est notée $depth(m_i)$ et est définie comme la distance de $m_i$ à la racine de $sb$. Voici un exemple (Fig. 2) de `string_builder` pour $k = 3$, où la profondeur de chaque feuille est indiquée entre parenthèses.

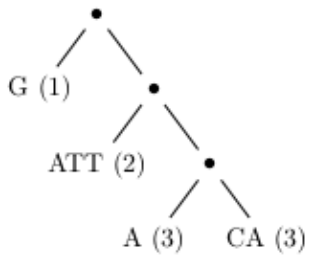


Figure 2: Exemple de `string_builder` ($k = 3$)

Le coût de l'accès à un caractère de la feuille $m_i$ est défini comme la profondeur de cette feuille dans $sb$, soit $depth(m_i)$ (on ne considère pas le coût de l'accès dans le mot $m_i$ lui-même). Le coût total d'un `string_builder` est alors défini comme la somme des coûts d'accès à tous ses caractères et vaut donc:

$$C(sb) = \sum_{i=0}^{k} length(m_i) \times depth(m_i)$$

**Question 4.**

- Définir la fonction `cost` qui prend en argument un `string_builder` et qui renvoie son coût selon la définition précédente.

**Question 5.**

- Définir une fonction `random_string` qui prend en argument un entier $i$ et qui génère un arbre de profondeur $i$. Vous présenterez l'algorithme utilisé pour la génération dans votre rapport, et vous discuterez de la pertinence de vos choix. **Note :** Pensez à consulter la documentation en ligne des modules `Random`, `Char` et `String`.

**Question 6.**

- Définir une fonction `list_of_string` qui prend en argument un `string_builder` et qui renvoie la liste des chaînes de caractères dans le même ordre que dans l'arbre (parcours infixe).

On propose l'algorithme d'équilibrage suivant :

- Transformer l'arbre en une liste de ses feuilles respectant l'ordre.
- Tant qu'il existe au moins 2 éléments dans la liste :
  1. trouver les deux éléments successifs dont la concaténation a le coût le plus faible,
  2. les retirer, les concaténer et insérer le résultat à leur position.
- Concaténer les deux derniers éléments.

**Question 7.**

- Définir une fonction `balance` qui prend en argument un `string_builder` et qui renvoie un nouveau `string_builder` équilibré selon l'algorithme précédent.

**Question 8.**

- Proposer une fonction qui calcule les gains (ou les pertes) en coût de la fonction `balance` sur un grand nombre d'arbres générés aléatoirement. La fonction peut, entre autre, renvoyer le min, le max, la moyenne et la valeur médiane.