NFP119 – nov 2018
Projet de traducteur `markdown` → `HTML`

Résumé

Il s'agit d'implanter un traducteur de la syntaxe `markdown` vers le langage `HTML`. La lecture et l'écriture de ces deux formats dans des fichiers sont déjà implantées dans le squelette du projet. Le travail consiste donc à écrire la fonction de traduction du type représentant un contenu `markdown` (`Md.tree`) vers le type représentant un contenu `HTML` (`Html.tree`) de sorte que l'affichage dans un navigateur du résultat `HTML` corresponde au contenu initial du fichier `markdown`. Un bonus sera donné si la page `HTML` générée contient une table des matières permettant de naviguer dans le document (liens cliquables).

Modalité d'évaluation du projet :

— Soutenance : pendant l'un des 3 derniers TP de l'année : présentation des fonctionnalités sur des exemples préparés + questions (10-15 minutes au total).

— Rendu final : avant l'examen final. Modalité à préciser sur le site du cours.

Remarque importante : Les séances de TP seront consacrées majoritairement à vous aider pour le projet.

1 Introduction

1.1 `markdown`

`markdown` [1] est un langage de *balisage léger* de mise en page ayant une syntaxe facile à lire et à écrire (contrairement à `HTML`). Un document balisé par Markdown peut être lu en l'état sans donner l'impression d'avoir été balisé ou formaté par des instructions particulières. Les possibilités de mise en page sont assez limitées. Voici un exemple de fichier `markdown`, dont l'affichage devrait ressembler à la figure 1 :

```
# Titre
## Sous-titre
ce **mot** est en gras. *Celui-ci* en italique.

Nouveau **paragraphe** introduit grâce à 2 sauts de ligne
consécutifs.

L'*imbrication **des balises** est*  possible.

* Les listes sont possible, y compris *imbriquées* et numérotées
    1. Les numéros ne sont pas retenu, ils sont recalculés par l'affichage
    3. Ainsi il n'est pas nécessaire de les numéroter correctement
* L'imbrication de liste est spécifiée par les **4 espaces** d'indentation
```

1. https://fr.wikipedia.org/wiki/Markdown

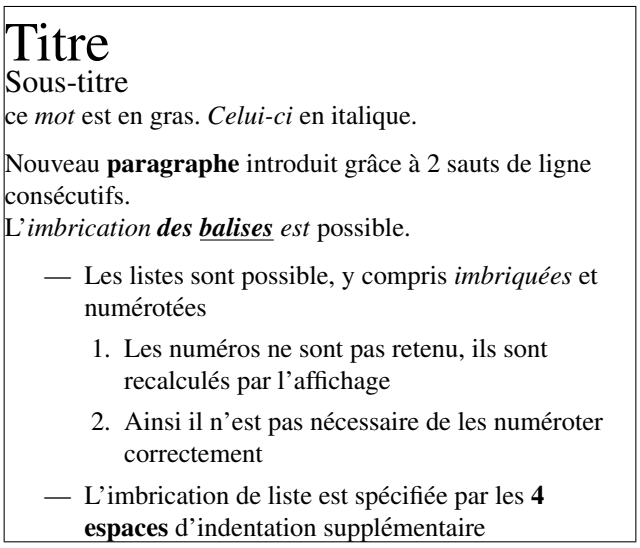


FIGURE 1 – Exemple d'affichage d'un fichier `markdown`

Dans le squelette du projet la lecture de ce format est assurée par la fonction `Md.lit_fichier` déjà implantée. Cette fonction retourne une valeur de type `Md.tree list`, où `Md.tree` est défini comme ceci :

```
type tree =
  | Texte of string
  | ForcedNL (* Un retour à la ligne forcé *)
  | Titre of int * element list (* Le int es le niveau du tire (1 à 5)*)
  | HLigne (* une ligne horizontale *)
  | Gras of element list
  | TresGras of element list
  | Code of string (* du code dans le texte (le texte doit apparaître tel quel *)
  | ParagCode of string (* un paragraphe de code *)
  | Lien of (element list*string) (* t=le texte à afficher, string=url *)
  | Paragraphe of element list
  | List of (element list) list (* Chaque puce contient une liste d'éléments *)
  | Enum of (element list) list
```

Il s'agit d'un type d'*arbre* dans lequel un *noeud* peut posséder une liste de fils, contenant zéro, un ou plusieurs fils. L'utilisation de ce genre de structure de données est la difficulté principale de ce projet et fera l'objet de cours et de plusieurs TP.

1.2 HTML

Une page Web est un texte écrit dans le format informatique *Hypertext Markup Language* (`HTML`) conçu pour fournir au navigateur le texte à afficher ainsi que la structure générale de sa mise en page : titres et paragraphes, listes, tableaux. La manière dont la structure influe sur l'affichage dépend du navigateur et des options choisies par l'utilisateur. Ainsi le fichier `HTML`[2] suivant sera affiché approximativement de la même façon que le `markdown` de la section précédente (figure 1). Notez que les sauts de ligne sont non significatifs.

2. Notez que nous utilisons volontairement la version 1.0 du format `HTML`, obsolète mais très simple.

```
<h1>Titre</h1>
<h2>Sous-titre</h2>
<p>ce <b>mot</b> est en gras. <i>Celui-ci<i> en italique.</p>
<p>Nouveau <b>paragraphe</b> introduit grâce à 2 sauts de ligne
consécutifs.</p>
<p>L'<i>imbrication <b>des balises</b> est<i>  possible.</p>
<ul>
<li> Les listes sont possible, y compris <i>imbriquées<i> et numérotées
  <ol>
    <li> Les numéros ne sont pas retenu, ils sont recalculés par l'affichage</li>
    <li> Ainsi il n'est pas nécessaire de les numéroter correctement</li>
  </ol>
</li>
<li> L'imbrication de liste est spécifiée par les <b>4 espaces</b> d'indentation
</li>
</ul>
```

Dans le projet le type `Html.tree` représente un arbre `HTML`. Certaines balises acceptent des *attributs*, comme la balise `<h1>` qui accepte l'attribut `name` (utile pour la table des matières). Consultez le web pour plus de précisions.

2 Ce qui est fourni

Il s'agit de travailler sur un programme partiellement écrit, disponible sur le site du cours, dans lequel des fonctions sont à implanter. Le programme est composé de nombreux fichiers appelés modules (voir l'explication sur les *modules* en annexe A).

Vous travaillerez uniquement dans le fichier `traduc.ml`. Vous devrez en revanche consulter les fichier `md.mli` et `html.mli` qui contiennent la définition des types d'arbres représentant le contenu `markdown` et `HTML` ainsi que les fonctions disponibles sur ces types.

2.1 Les type d'arbres

Comme les balises peuvent être *imbriquées*, la structure de donnée la plus pratique et la plus naturelle pour représenter un contenu `markdown` ou `HTML` est l'*arbre*. Dans ce projet les arbres de type `Html.tree` et `Md.element` représentent respectivement un contenu `HTML` et un contenu `markdown`. Il s'agit d'arbres à branchement quelconque : chaque noeud a un nombre arbitraire de sous-arbres stockés dans une *liste*. La manipulation de ces arbres est la principale difficulté du projet. N'hésitez pas à poser beaucoup de questions lors des séances de TP afin de bien comprendre comment procéder.

Les noeuds des arbre `HTML` ont également une liste d'attributs. Dans ce projet cette liste est à ignorer sauf lorsque vous implanterez les liens (attribut `href`) et la table des matières (`href` et `name`).

En `HTML` les noeuds `Html.Word` n'ont pas de sous-arbre et contiennent du texte pur (un ou plusieurs mots). Même chose pour les noeuds `Md.Text` pour `markdown`.

2.2 Le type des arbres `markdown`

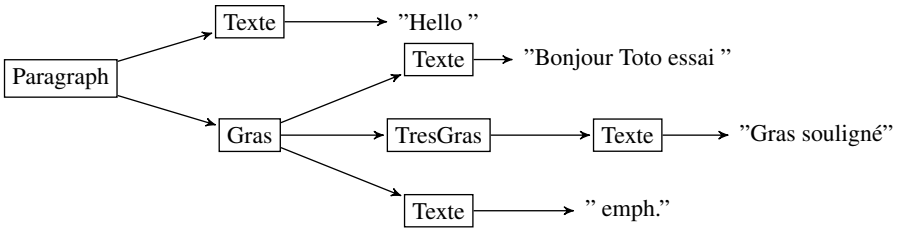
est fourni dans le module `Md`[3]. Il y a un constructeur par balise.

Exemples d'arbres Le contenu `markdown` ci-dessous :

```
Hello *Bonjour Toto essai **Gras souligné** emph.*
```

schématisé par l'arbre suivant :

3. voir le type `tree` à la section 1.1



sera représenté par la valeur caml suivante (notez que les espaces sont gardés dans les balises `Text`, inutile donc de les ajouter dans l'`HTML`) :

```
Md.Paragraphe
  [Md.Texte "Hello␣";
   Md.Gras
    [Md.Texte "Bonjour␣Toto␣essai␣"; Md.TresGras [Md.Texte "Gras␣souligné"];
     Md.Texte "␣emph."]]
```

Vous pouvez voir la représentation en utilisant les fonctions `Md.lit_fichier` et `Md.lit_string` dans la boucle interactive comme expliqué à la section 4.3.

2.3 Le type des arbres `HTML`

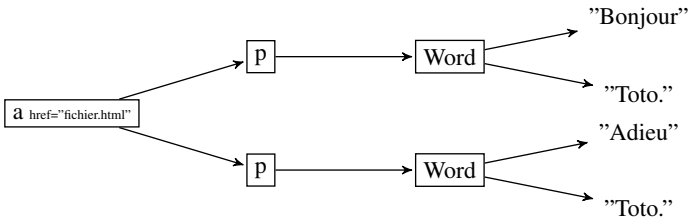
Le type des arbres `HTML` est fourni dans le module `Html`[4]. Il y a un constructeur par balise, et certaines balises prennent une liste d'attributs en plus de la liste des sous-arbres.

```
type attribut =
    | Href of string
    | Size of int
    | Value of int
    | Name of string

type tree = (* Un arbre HTML. *)
  | Empty
  | Html of attribut list * tree list
  | P of attribut list * tree list
  | B of attribut list * tree list
  | I of attribut list * tree list
  | A of attribut list * tree list
  | H1 of attribut list * tree list
  | ...
  | Word of string
```

Exemples d'arbres Par exemple le code `HTML` ci-dessous à gauche, schématisé par l'arbre ci-dessous à droite :

```
<A href="fichier.html">
 <p>Bonjour Toto.</p>
 <p>Aurevoir Toto.</p>
</A>
```



sera représenté par la valeur caml suivante (notez qu'ici chaque mot est dans une balise séparée, ce n'est pas nécessaire) :

4. notez que en dehors de ce fichier on fait référence à ce type par `Html.tree`, ce qui le différencie de `Md.tree` le type réprésentant les arbres `markdown`

```
A([Href("fichier.html")] ,
  [ P([], [Word "Bonjour"; Word "Toto." ]);
    P([], [Word "Adieu"; Word "Toto." ])])
```

Le schéma d'un arbre plus complexe est donné quant à lui en annexe C.

3 Ce qui vous est demandé

3.1 La fonction de traduction `markdown` → `HTML`

3.1.1 Description

Vous devez implanter la fonction `traduit` (dans le fichier `traduc.ml`) de traduction d'un arbre `markdown` vers un arbre `HTML`. Comme les arbres ont des *listes* de sous-arbres cette fonction est *mutuellement récursive* avec une autre fonctions qui traduit une liste d'arbre. Nous travaillerons sur les fonctions mutuellement récursives en TP.

```
let rec traduit (e:Md.tree): Html.tree =
  ...
and traduit_list (l: Md.tree list) : Html.tree list =
  ...
```

3.1.2 Fonctionnalités progressives

Il est conseillé d'implanter les fonctionnalités de manière progressive. Lorsqu'une version semble fonctionner correctement on peut passer à la version suivante (sans oublier de garder une copie de la version qui marche).

Première version Une traduction correcte des balises simples (gras, italique, paragraphe, ...). L'imbrication de ces balises doit fonctionner.

Deuxième version (pour avoir plus que la moyenne) Traduction des listes numérotées ou pas (balises `<ul>` ou `<ol>` en `HTML`). L'imbrication de toutes les balises traitées doit fonctionner.

Amélioration (> 13) Ajout d'une table des matières `HTML` cliquable. Pour implanter cette fonctionnalité il faudra complexifier la fonction de traduction (ajout de paramètres et changement du type de retour) pour lui permettre d'insérer des références uniques dans les balises de titres (`<h1 name="section1">...</h1>`) et collecter les contenus de ces balises. Il faudra aussi modifier la fonction `Traduc.build_html_page_body` pour l'insertion proprement dite de la table des matières : une liste de titres cliquables (`<a href="#section1">...</a>`) en tête de page `HTML`.

3.2 Difficultés

La fonction de traduction doit parcourir récursivement l'arbre `markdown` et construire un arbre `HTML` ayant une structure très similaire. La principale difficulté consiste à maîtriser la récursion sur un arbre où les noeuds ont des *listes* de fils.

4 Compilation et exécution

Remarque pour Windows Dans cette section, toutes les command `xxx.sh` sont exécutables sous linux seulement. Sous windows utilisez la commande `xxx.bat` à la place.

4.1 Commandes de compilation

Compilation du programme Pour compiler le projet on doit ouvrir un terminal (ligne de commande) et se positionner dans le répertoire du projet (command `cd`), puis lancer la commande `compile.sh` (`.bat` sous windows)[5]. La compilation crée l'exécutable `md_to_html.byte`.

4.2 Exécution du programme `md_to_html.byte`

Une fois compilé, le programme se lance de la manière suivante (remplacer les « / » par des « \ » sous windows) :

```
./md_to_html.byte fichier.md
```

où `fichier.md` est le fichier à traduire. Par défaut la traduction `HTML` s'affichera dans le terminal. Si on précise un fichier de sortie :

```
./md_to_html.byte fichier.md -o fichier.html
```

le code `HTML` sera écrit dans le fichier déclaré et on pourra l'ouvrir dans un navigateur.

4.3 Utiliser une boucle interactive (toplevel) pour le test

Vous pouvez utiliser la boucle interactive de `OCaml` pour tester vos fonctions. Pour cela

1. *compilez le programme* comme décrit ci-dessus (à chaque fois que vous avez changé qqchose dans le programme)
2. puis lancez la commande : `toplevel.sh`.

Vous pouvez ensuite utiliser les fonctions de vos modules. Par exemple :

```
$ ./toplevel.sh
        OCaml version 4.07.0

# Md.lit_fichier "tests/test2.md";;
- : Md.tree list =
[Md.Paragraphe
  [Md.Gras [Md.Texte "Gras souligné"]; Md.Texte " ";
   Md.Gras [Md.Texte "emph"]; Md.Texte "."; Md.ForcedNL;
   Md.TresGras [Md.Texte "Gras "; Md.Gras [Md.Texte "souligné"]];
   Md.Texte " "; Md.Gras [Md.Texte "emph"]; Md.Texte "."]]
# let md = Md.lit_string "Hello *Bonjour Toto essai **Gras souligné** emph.*";;
val md : Md.tree list =
[Md.Paragraphe
  [Md.Texte " Hello ";
   Md.Gras
    [Md.Texte "Bonjour Toto essai "; Md.TresGras [Md.Texte "Gras souligné"];
     Md.Texte " emph."]]]
# Traduc.traduit_list md ;;
- : Html.tree list = [Html.Word "Traduction Non implantée"]
```

A Qu'est-ce qu'un module ?

En `ocaml` les modules sont un outil très puissant de structuration du code mais nous n'en verrons dans ce projet qu'une versions très simple. On appelle module un fichier dont le nom commence par une minuscule et se termine par l'extension `.ml` (par exemple : `toto.ml`), éventuellement accompagné d'un fichier de même nom avec l'extension `.mli` (donc `toto.mli` dans notre exemple). Le premier contient des fonctions et types ocaml. Le second contient

5. Si vous changez les dépendance entre fichiers vous devrez regénérer la commande compile.sh, voir TP.

uniquement les signatures des fonctions visibles depuis les autres modules. Le fichier `.mli` joue donc le rôle d'interface pour le fichier `.ml`. Le nom du module est engendré automatiquement à partir du nom du fichier en mettant une majuscule et en enlevant l'extension `.ml` (ce qui donne `Toto` dans notre exemple). Pour désigner une fonction (ou un type) `f` du module `Toto` dans un autre module on écrit `Toto.f`, ou bien simplement `f` si on a préalablement écrit `open Toto`.

Remarque : Dans le fichier `traduc.ml` il est déconseiller de faire `open` sur les module `Md` et `Html` car il contiennent des types et fonctions de même nom. Il vaut mieux utiliser la notation `Md.tree` et `Html.tree` pour plus de clarté.

B Liste des balises `markdown` à traiter

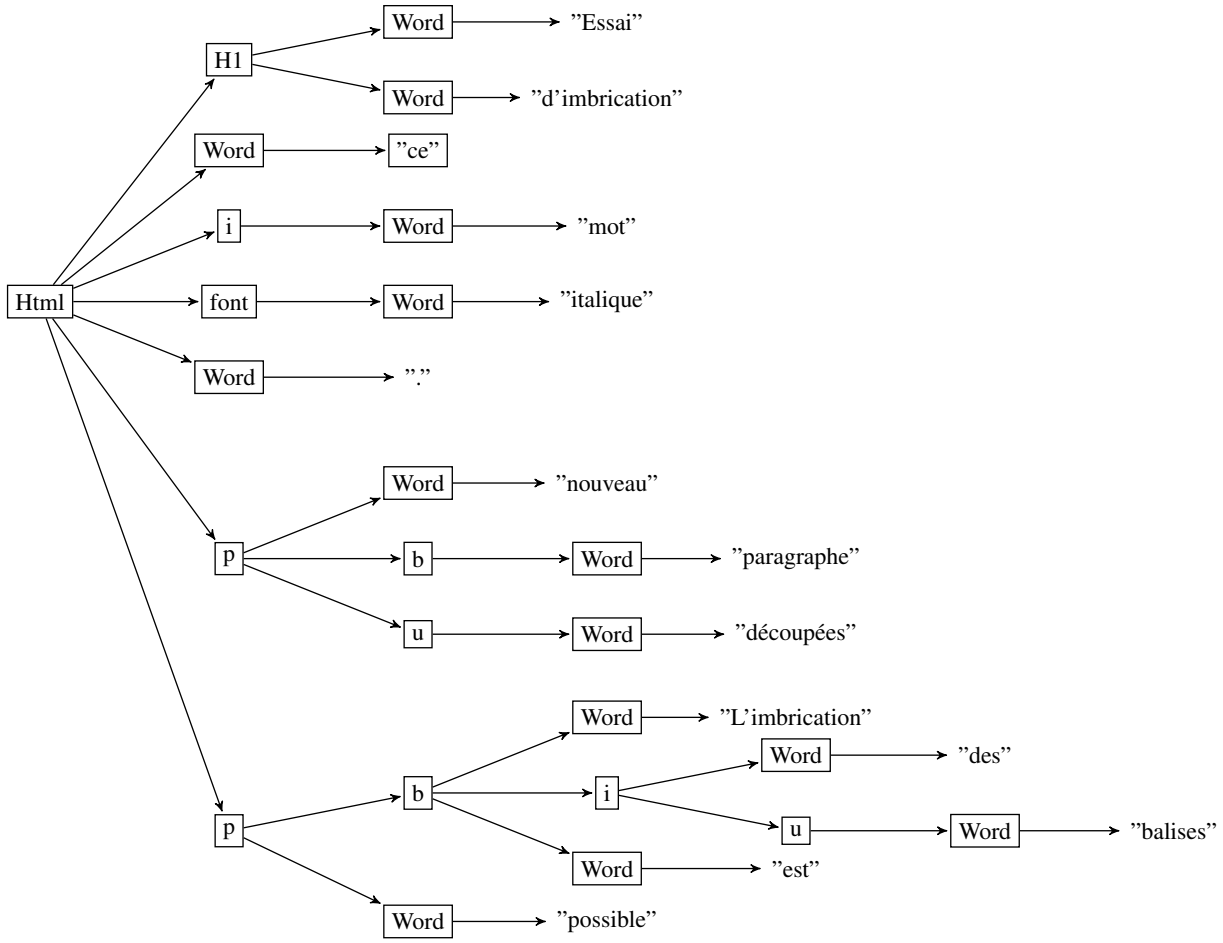
B.1 Balises à traiter

— `*` `**`
— les liens de la forme `[text](url)`
— les titres jusqu'au cinquième niveau : `#`, `##`, `###`, `####`, `#####`
— Les listes numérotées et non numérotées

C Exemple d'arbre `HTML`

Dans cet exemple, on voit que la structure reflète l'imbrication des balise. Par exemple les balises `<b>`, `<i>` et `<u>` sont imbriquées dans le dernier paragraphe. Vous pouvez mettre ce texte dans un fichier xxx.html et l'ouvrir dans un navigateur pour voir le résultat.

```
<html>
<h1>Essai d'imbrication:</h1> ce <i>mot</i>
<font color="red">italique</font>. <p> Nouveau
<b> paragraphe </b> <u>découpées</u>.</p>
<p><b>L'imbrication <i>des <u>balises</u></i> est</b> possible.</p>
</html>
```


Html
H1
Word
"Essai"
Word
"d'imbrication"
Word
"ce"
i
Word
"mot"
font
Word
"italique"
Word
"."
p
Word
"nouveau"
b
Word
"paragraphe"
u
Word
"découpées"
p
b
Word
"L'imbrication"
i
Word
"des"
u
Word
"balises"
Word
"est"
Word
"possible"