Projet de navigateur HTML – Partie I

Résumé

Il s'agit d'implanter la fonction d'affichage d'un navigateur HTML (le reste du programme est fourni dans un squelette disponible sur le site du cours).

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 : 7 jours après 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. Si vous ne faites pas de soutenance vous risquez d'avoir une moins bonne note.

1 Introduction

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 [1] suivant sera affiché approximativement comme à la figure 1.

```
<html>
 <h1>Essai:</h1> ce <i>mot</i> est en <font color="red">italique</font>.
 <p> Nouveau <b> paragraphe </b>, les lignes sont <u>découpées</u>.</p>
 <p><b>L'imbrication <i>des <u>balises</u></i>est</b>  possible.</p>
 <p> les <a href="autrefichier.html">ancres</a> sont permises.</p>
</html>
```

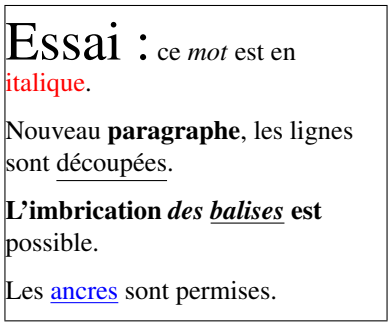


FIGURE 1 – Exemple d'affichage d'un fichier HTML

Le but de ce projet est de programmer un petit navigateur HTML capable d'afficher correctement un texte comportant des balises HTML (1.0) (décrites plus bas). Il sera également possible (optionel) de *cliquer* sur un lien pour afficher un autre fichier ou une autre partie du même fichier.

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

Certaines balises acceptent des *attributs*, comme la balise `<font>` qui accepte l'attribut `color`. Ces attributs sont en général optionnels et dans un premier temps vous pouvez les ignorer. Une page web commence nécessairement par la balise `<html>` et se termine par `</html>`. Consultez l'annexe A et surtout le web pour plus de précisions.

2 Ce qui vous est demandé

La fonction d'affichage HTML

Vous devez implanter la fonction `display_html` d'affichage d'un arbre HTML dans le fichier `draw_html.ml`. Comme les arbres ont des listes de sous-arbres cette fonction est *mutuellement récursive* avec celle qui affiche une liste d'arbre. Nous travaillerons sur les fonctions mutuellement récursives en TP.

```
let rec display_html (state:display_state) (htree:html_tree) : return_state = ...
and display_html_list (state:display_state) (htree:html_tree list) : return_state = ...
```

Avec les définitions de types suivantes (types enregistrements (records)) :

```
type display_state =
  { start_pos: position;                        (** Position courante de l'afficheur. *)
    style: text_style; }                            (** Style courant de l'afficheur. *)

type return_state =
  { list_actions: drawing_action list;                (** Liste des drawing actions. *)
    last_position: position;                                (** Position finale. *)
    (* ancres... *) }
```

Ces fonctions prennent en argument un `display_state` et un arbre HTML (ou une liste d'arbres HTML) et retournent un objet de type `return_state`. Donc elles n'affichent pas à proprement parler : elles retournent les informations d'affichage (`drawing_actions list`) ainsi que d'autres informations nécessaires pour continuer à calculer une éventuelle suite de l'affichage (en particulier la position de la fin de l'affichage précédent, c'est-à-dire la position à partir de laquelle il faut continuer à afficher).

C'est une autre fonction déjà écrite (`draw_list_actions`) qui se chargera de l'affichage graphique. Des exemples de `drawing_action` sont présents dans le fichier principal `navig.ml`.

Fonctionnalités

Voici par ordre croissant de difficulté (et de bonne note) les fonctionnalités à implanter.

Première version Interprétation des balises HTML de style (gras italique etc). Les ancres sont affichées en bleu mais ne sont pas cliquables. Les textes trop longs sont affichés sur plusieurs lignes (on ne coupe pas les mots en deux) et la balise `<p>` est interprétée correctement. La hauteur des lignes n'est PAS ajustée à la taille des caractères.

Si cette partie fonctionne correctement et que votre code est à peu près propre vous aurez la moyenne au projet. Les améliorations suivantes sont possibles (sans ordre de priorité) :

Amélioration 1 Les ancres sont cliquables (affichage d'un nouveau fichier). Pour cela il faut collecter les coordonnees cliquables dans le `return_state`. La réaction au clic se fera facilement en TP avec l'aide de l'enseignant.

Amélioration 2 La balise `<center>` est interprétée correctement.

Amélioration 3 On peut faire défiler le texte page par page.

Amélioration 4 La hauteur des lignes est gérée.

Difficultés

L'imbrication des balises. Par exemple le mot `balises` dans l'exemple de l'annexe est affiché en gras, en italique et en souligné car il se trouve à l'intérieur de ces trois balises simultanément. Le mot suivant (`est`) est encore en gras car les balises `<u>` et `<i>` ont été fermées mais pas la balise `<b>`. On voit que l'imbrication des balises doit être considérée pour un affichage correct.

Pour résoudre cette difficulté on représente le contenu HTML sous la forme d'un *arbre*, qui reflète les imbrications de balises. Le type de cet arbre (`html_tree`) est fourni dans le squelette du programme, voir section 2.1 pour plus d'explications. Le mécanisme de lecture du fichier et de construction de l'arbre *est déjà fourni* (fonction `Html_tree.lit_fichier`).

La fonction d'affichage `display_html` doit parcourir récursivement l'arbre HTML. À chaque appel récursif l'argument de type `display_state`, qui permet de savoir où et comment afficher les prochains mots, est remplacé pour refléter l'effet de la balise traversée.

La mise en page. Lorsqu'on affiche les différents sous-arbres d'un noeud il faut les afficher les uns à la suite des autres dans la fenêtre graphique.

Donc la fonction d'affichage prend non seulement en argument la position à partir de laquelle elle doit afficher son arbre HTML, mais elle doit également *retourner* la nouvelle position d'affichage après affichage de son arbre afin de permettre les affichage futurs. Elle doit donc retourner plusieurs chose, elle retourne donc un type record `return_state` qui contient plusieurs champs : `last_position` pour la nouvelle position, `list_actions` pour la liste `drawing_actions`, etc.

Le programm `navig.byte`

Le programme `navig.byte` prend en argument un nom de fichier (local, il n'est bien entendu pas demandé d'accéder au réseau). Une fois l'affichage effectué, le programme attendra que l'utilisateur effectue une action et réagira en fonction de cette action. Dans un premier temps la seul action (sortir du programme) est déjà implantée. À la troisième version d'autres actions apparaîtront.

Le programme peut également prendre 2 options supplémentaires :

— `-parse`, affiche dans le terminal l'arbre HTML reconnu, sans démarrer la fenêtre graphisme. Ceci permet de savoir ce que le parseur du programme a reconnu. Cela peut être utile pour comprendre et pour débugger vos fonctions programme.

— `-exemple`, ne lit pas le fichier (qui peut être absent) et affiche dans la fenêtre graphique des exemples de `drawing_actions`.

2.1 Les type d'arbres du projet

Comme les balises peuvent être *imbriquées*, la structure de donnée la plus pratique et la plus naturelle pour représenter un contenu HTML est l'*arbre*. Dans ce projet les arbres de type (`Html_tree.html_tree`) représentent une page web. Il s'agit d'arbres à branchement quelconque : chaque noeud à un nombre arbitraire de sous-arbres stockés dans une *liste*. La manipulation des 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 de l'arbre ont également une liste d'attributs. Dans ce projet cette liste est à ignorer sauf pour la balise `font` et `href` (deuxième version).

Les noeuds `Word` n'ont pas de sous-arbre et contiennent un seul mot (suite de symboles ne contenant pas d'espace).

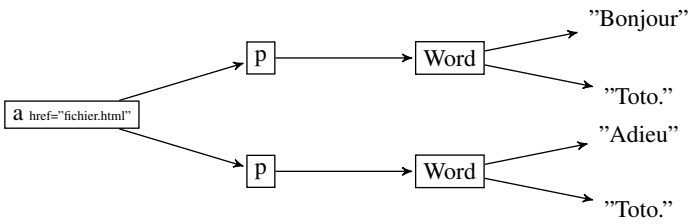
Le type des arbres HTML est fourni dans le module `Html_tree`. Il y a un constructeur par balise.

```
type attribut =
    | Href of string
    | Hcolor of string
    | Size of int
```

```
(* Type des arbres. *)
type html_tree =
  | Empty
  | Html of attribut list * html_tree list
  | P of attribut list * html_tree list
  | B of attribut list * html_tree list
  | I of attribut list * html_tree list
  | A of attribut list * html_tree list
  | H1 of attribut list * html_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 :

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

Le schéma de l'arbre HTML correspondant à l'exemple de l'introduction est donné quant à lui en annexe B.

3 Ce qui est fourni

Il s'agit de travailler sur le programme partiellement écrit, disponible sur le site du cours, dans lequel des fonctions sont à implanter. Le programme est composé de plusieurs fichiers (voir l'explication sur les *modules* plus bas) qu'il s'agira de compiler de manière classique (comme en Java ou en C).

3.1 Les modules du projet

Compilation Pour compiler sans eclipse vous pouvez utiliser la commande suivante : `ocamlbuild navig.byte` (ou `navig.native` pour un exécutable plus rapide mais un temps de compilation plus lent). Si vous avez configuré eclipse correctement (vor site du projet) et qu'aucune erreur n'est visible dans les fichiers cela signifie que l'exécutable `navig.byte` est créé dans le répertoire du projet.

Sous Windows il arrive que la compilation automatique via `ocamlbuild` ne marche pas, vous pouvez utiliser le script `compile.bat` qui recompile tous les fichiers.

Exécution L'exécutable produit est `navig.byte` et pour le lancer il faut lui donner en argument le nom d'un fichier HTML. Donc en ligne de commande tapez par exemple sous linux/macos :
`navig.native tests/test1.html`.
et sous windows :
`navig.native tests\test1.html`.
Vous ne pouvez pas lancer l'exécution directement depuis eclipse car il manque l'argument.

3.2 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 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`.

3.3 Les modules du projet

Les fonctions déjà écrites sont documentées (ouvrir le fichier `doc.docdir/index.html`). Une documentation de la librairie standard de OCaml se trouve à l'adresse suivante (section `The standard library`) : `http://caml.inria.fr/pub/docs/manual-ocaml/index.html`.

— Le module `Xmlm` (fichiers `xmlm.ml(i)`) est une modification d'un parseur de fichier XML disponible sur internet (`http://erratique.ch/software/xmlm`). Vous n'avez pas à utiliser ce module directement, ni à le modifier.

— `Html_tree` (fichier `html_tree.ml(i)`), dans lequel le type des arbres HTML (`html_tree`, voir plus bas) est défini. Il contient notamment la fonction `lit_fichier` qui permet de lire un fichier HTML et retourne un arbre HTML.

— `Graph_utils` (fichier `graph_utils.ml(i)`) qui contient des fonctionnalités d'affichages basées sur le module `Graphics` de la librairie de OCaml. Si nécessaire (troisième version) ajoutez des fonctionnalité à ce module (en n'oubliant pas de mettre dans le `.mli` la signature des fonctions à exporter).

— `Draw_html` (fichier `draw_html.ml`) qui sera votre fichier travail principal, vous implanterez vos fonctions d'affichage dans ce fichier.

— `Navig` (fichier `navig.ml`) est le module principal, celui qui se lance à l'exécution et qui lancera votre fonction d'affichage sur le fichier passé en argument du programme. Il contient des exemples d'utilisation des autres modules.
le programme obtenu en compilant s'appelle `navig.byte` et prend en argument un nom de fichier et/ou des options.
Dans un premier temps vous n'avez pas à modifier ce fichier. Vous serez amené à modifier la fonction `main` seulement pour la partie interactive du programme (lorsque vous devrez réagir à des événements clavier ou souris).

A Liste des balises HTML à traiter

Attention, sous windows certaines typographies ne sont pas disponibles (différentes tailles de fontes, italique). Il est important de tester votre programme sous linux en salle TP.

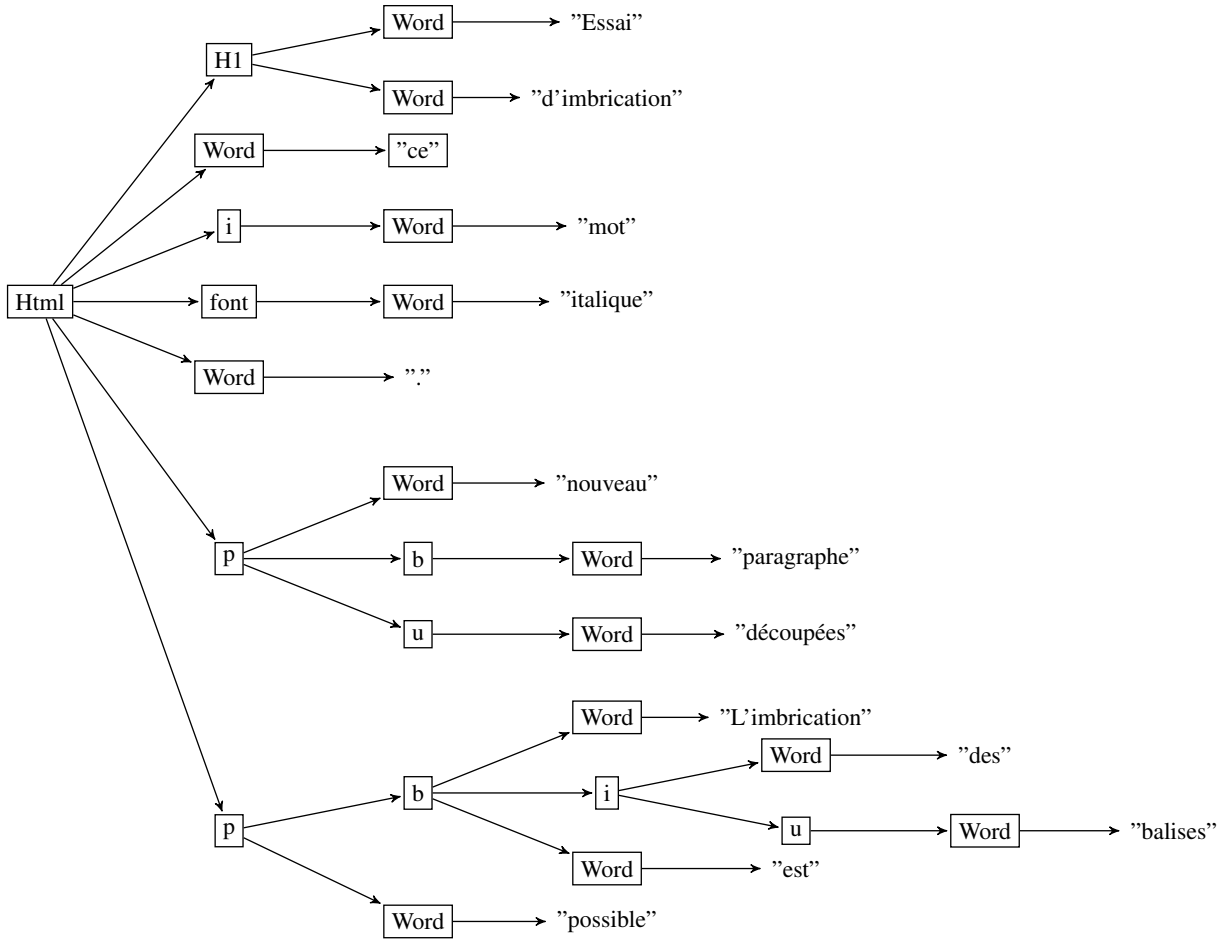
A.1 Balises à traiter

`<p>texte</p>`	Texte dans un paragraphe
`<b>texte</b>`	**Texte en gras**
`<i>texte</i>`	*Texte en italique*
`<u>texte</u>`	Texte souligné
`<font color="#cc0000"> texte</font>`	Affiche le texte dans la couleur choisie
`<body> texte</body>`	Annonce la partie principale de la page, afficher les sous-arbres.
` `	Retour à la ligne forcé
`<em>texte</em>`	mise en avant du texte (en général italique)
`<h1>texte</h1>`	Taille de texte 1 (Grande), hauteur de la ligne courante à adapter (version 3)
`<h2>texte</h2>`	Taille de texte 2, idem
`<h3>texte</h3>`	Taille de texte 3, idem
`<h4>texte</h4>`	Taille de texte 4, idem
`<h5>texte</h5>`	Taille de texte 5 (Petite), idem
`<head>texte</head>`	Annonce l'en-tête du fichier, ne pas afficher
`<html>texte</html>`	Annonce le début d'une page web
`<title>texte</title>`	Texte est le titre de la page, ne pas afficher
`<a href="lien.html"> texte </a>`	Affiche en bleu souligné (version 1), cliquable (version 2)
`<center>texte</center>`	Centrage du texte (version 3)
	Gestion des textes longs (next page/previous page, version 3)

B 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"