Horia Mut, Lucas Bitter, Steve Visinand

HE-ARC

Générateur de pages HTML

Projet compilateur

# Introduction

Ce projet s’inscrit dans le cadre du cours de compilateur de troisième année de la branche Développent Logiciel et Multimédia (DLM) de l’HE-Arc.

**Langage utilisé** : Python avec PLY

## Etat de l’art

Ce projet fait partie des langages de description visant à améliorer la productivité du programmeur et à lui éviter des tâches redondantes.

Dans le même type que ce projet nous pouvant citer les langages de « pre-processing » devenus très utilisés telle que :

* **SASS**  : <http://sass-lang.com>

Syntactically Awesome Style Sheets est un langage qui vise à donner des « superpouvoirs » aux feuilles de style CSS.

* **LESS** : <http://lesscss.org>

Même utilité que SASS.

* **HAML** : <http://haml.info>

HAML sert à accélérer l’écriture d’HTML.

Fondamentalement notre projet aborde une syntaxe plus courte et produisant de plus grandes portions de code, il a plutôt comme but d’initier un projet web rapidement avec une structure initiale de page auto-générée.

## Suivi de projet

**Wiki** : <https://github.com/stevevisinand/CompilerToHTML/wiki>

**Git** : <https://github.com/stevevisinand/CompilerToHTML>

Ce projet étant réalisé à trois, nous avons pris la peine de créer un Git, un suivi de projet ainsi qu’une spécification dans un wiki commun en plus de ce document.

# But

Le but de notre programme est de générer des pages HTML ainsi que leur style avec un langage simplifié et beaucoup plus court que de l’HTML.

Ainsi, un utilisateur sans connaissance de l’HTML pourra facilement générer plusieurs pages avec une structure et des liens entre elles.

# Spécifications du langage

## Eléments

Un **élément** représente une partie de la page HTML. L’utilisateur paramètre cet élément sous forme de variable avec des attributs spécifiés pour le personnaliser comme il le souhaite.

**"element"** est un type abstrait et ne peut pas être créé seul, il lui faut un type spécifique.

On peut passer des variables à un "element" entre les parenthèses. On pourra ensuite utiliser ces variables pour les concaténer à des "attributs" de notre element.

**Déclaration :**

**element** **TYPE\_SPECIFIQUE NOM\_ELEMENT**{

ATTRIBUTES

}

« TYPE\_SPECIFIQUE » est le type réel de l’élément : « header », « footer », « nav », …

« NOM\_ELEMENT » est le nom de la variable contenant l’élément.

« ATTRIBUTES » sont les paramètres possibles de l’élément.

### Header

Entête du site.

|  |  |
| --- | --- |
| **TYPE\_SPECIFIQUE** | header |
| **ATTRIBUTES**  **(possibles)** | title, color, text\_color |

**Exemple :**

**element** **header** **monHead**{

title : "Mon site";

color : "blue";

text\_color : "red";

};

**Résultat (print dans une page) :**



### Nav

Menu du site web.

|  |  |
| --- | --- |
| **TYPE\_SPECIFIQUE** | nav |
| **ATTRIBUTES**  **(possibles)** | color, text\_color, menu |

"menu" décrit l'apparence et le contenu du menu du site internet, le texte des cases éléments du menu et la page cible.

**Exemple :**

**element nav monNav**{

color : "#FFFFF";

text\_color : "#FAFAFA";

menu : [

"home" : index, //index contient la page "index"

"gallerie" : gallery,

];

};

**Résultat (print dans une page) :**



Dans ce cas gallery contient deux sous pages, gal1 et gal2

Un menu peut être créé avant que tout les pages ne soit « compilées », pour ce faire on obtient les valeurs « nom » et « adresse » de chaque page dans le parseur puis le générateur de code HTML « generatorHTML » réutilise ces dernières.

### footer

Pied du site web.

|  |  |
| --- | --- |
| **TYPE\_SPECIFIQUE** | footer |
| **ATTRIBUTES**  **(possibles)** | title, paragraph (text), copyright (text), color, text\_color |

**Exemple :**

**element footer myFooter**{  
 title: "Title";

paragraph: "paragraphe";

copyright: "blue";  
 color: "red";

text\_color: "white";  
};

**Résultat (print dans une page) :**



## Pages

La page est un type défini par l’utilisateur qui génèrera une page HTML, dans cette page on ajoute nos éléments HTML déjà crées.

Une page est décrite par un nom de variable et des attributs obligatoire.

**Elle doit définir les attributs :**

* Name : Nom de la page
* Address : Adresse relative de la page

On peut appeler la méthode **"print"** dans une page afin d'imprimer des **"element"** dans la page.

On peut ajouter des sous-pages à une page pour créer une hiérarchie dans la navigation avec l'opérateur "+=".

**exemple :**

page gallery{

//paramètres obligatoires

name : "home";

address : "/index.htm";

print monHead();

print monNav();

print "contenu de la page";

print monFoot(1);

};

gallery += galleryPaysages;

## Boucles

Il est possible de crée une boucle simple avec une seule instruction interne. Cette boucle est utilisée pour généré des éléments de page.

for(i from 1 to 10){  
 //instruction  
};

## Variables

Un utilisateur peut définir une variable contenant des éléments, des pages mais également des variables au contenu plus classique telle que des nombres ou des valeurs texte.

Les variables sont déclarées ainsi :

VAR\_NAME = VALUE;

# Implémentation

## Mots réservés

Voici la liste des mots réservés à l’exécution du programme. L’utilisateur ne peut pas les utiliser hors de leur fonction prévue.

|  |  |
| --- | --- |
| **Mot clé** | **Type** |
| page | "VARIABLE" |
|  |  |
| name | PAGE ATTRIBUT |
| adress | PAGE ATTRIBUT |
| content | PAGE ATTRIBUT |
|  |  |
| element | "ABSTRACT" |
|  |  |
| nav | "VARIABLE" |
| header | "VARIABLE" |
| footer | "VARIABLE" |
|  |  |
| titile | HEADER ATTRIBUT |
| color | ELEMENT ATTRIBUT |
| text | HEADER, FOOTER ATTRIBUT |
| text\_color | HEADER, FOOTER ATTRIBUT |
| paragraph | FOOTER ATTRIBUT |
| copyright | FOOTER ATTRIBUT |

## Opérateurs

Voici la liste des symboles réservés au programme avec leur utilisation.

|  |  |
| --- | --- |
| **Opérateur** | **Fonction** |
| ; | Fin de ligne |
| = | Affectation de variable |
| += | Ajout d'une sous-page à une page |
| [] | Définition d'un tableau associatif pour l'élément menu |
| {} | Block de code, délimitation de la portée de variables |
| () | Paramètres élément, for |
| "" | Délimitation chaine de caractère |
| . | Concaténation de deux chaines de caractères |
| : | Affectation d'attributs "réservés" des éléments |

## Structure

* **compilerHTML.py**  : Compilateur
  + **generetorHTML.py** : Ensemble de fonctions utiles à la génération de code HTML
  + **AST.py**  : Classe de description d’un noeud
* **syntaxeHTML.py** : Analyse syntaxique
* **lexHTML.py**  : Analyse lexicale

Les fichiers HTML sont générés dans le dossier « generatedSite », attention le fichier « styles.css » doit obligatoirement être présent à l’intérieur.

# Exemple d’application

Les fichiers d’exemple sont présents dans le dossier « exemples » à la racine du projet. Ils ont l’extension « .cmphtml » (CoMPiler to HTML).

Le programme prend comme premier argument un fichier « .cmphtml » et va générer les fichiers « .htm » dans le dossier « generatedSite » présent à la racine du projet.

Ce dossier contient en tout temps un sous-dossier « styles » contenant un fichier de styles CSS. Il est impératif de copier ce dossier avec les fichiers générés pour utiliser le site hors du dossier « generatedSite ».

Le programme à lancer s’appelle : compilerHTML.py

## Exemples

### Page simple

Des styles de bases sont appliqués si vous ne spécifiez pas de styles avec « color » ou « text\_color ».

***Fichier source : exemples/simplepages.cmphtml***

**Code « .cmphtml » :**

titleSite = "Mon Site simple"; //une variables  
  
element header monHead{  
 title : titleSite; //le titre du site prend une variable ou pas :)  
};  
  
element nav monNav{  
 menu : [  
 "home" : index  
 ];  
};  
  
element footer monFoot{  
 title : "mon pied de page";  
 paragraph : "Un petit mot";  
 copyright : "(c) mon copyright";  
};  
  
//voila ma page :D  
page index{  
 //parametres obligatoires (name et address)  
 name : "ma page avec style";  
 address : "/index.htm";  
  
 print monHead;  
 print monNav;  
  
 //on peut utiliser de l'HTML directement  
 print "<div class='center'>";  
 print "<h1>Bienvenu sur mon site tout neuf :)</h1>";  
 print "<p>N'est il pas beau ?</p>";  
 print "</div>";  
  
 print monFoot;  
}

**Fichier généré :**



### Styles et boucles

On peut utiliser des boucles pour mettre en page du contenu dans le site. Et on peut styliser le site en spécifiant des couleurs.

***Fichier source : exemples/bouclestyles.cmphtml***

**Code :**

element header monHead{  
 title : "Ma superbe gallerie";  
 color : "red";  
};  
  
page gallery{  
 //parametres obligatoires  
 name : "Des paysages";  
 address : "./paysages.htm";  
  
 print myHead;  
 print myNav;  
 print "<div class='center'>";  
  
 print "<h1>Voici des paysages</h1>";  
 print "<table>";  
 for(i from 1 to 10){  
 print "<tr><td><img src='http://fakeimg.pl/290/'></td><td><img src='http://fakeimg.pl/290/'></td><td><img src='http://fakeimg.pl/290/'></td><tr>"  
 };  
 print "</table>";  
  
 print "</div>";  
}

**Fichier généré :**

****

### Multi-pages

En additionnant 2 pages, elles seront affichées en accordéon dans le menu et l’utilisateur pensera qu’il s’agira de deux pages imbriqués.

***Fichier source : exemples/site1.cmphtm***

**Code :**

mySite = 3;  
titlee = "Helo";  
  
element header myHead{  
 title : titlee;  
 color : "red";  
};  
  
element footer myFooter{  
 title: "MyFooter";  
 color : "red"; // We can add comments that are ignored here  
 paragraph : "This is a paragraph without special characters";  
};  
  
element nav myNav{  
 menu : [  
 "home" : maison,  
 "gallery" : galleryHello  
 ];  
};  
  
page maison{  
 name : "home";  
 address : "./index.htm";  
  
 print myHead;  
 print myNav;  
 print "<div class='center'>";  
 print "<h1>Bienvenu sur mon site tout neuf :)</h1>";  
 print "<p>Il n'est pas beau ?</p>";  
 print "</div>";  
 print myFooter;  
};  
  
page galleryHello{  
 //parametres obligatoires  
 name : "Mes photos";  
 address : "./hello.htm";  
  
 print myHead;  
 print myNav;  
  
 print "<div class='center'>";  
 print "<h1>Mes galeries</h1>";  
 print "<p>dans cette page vous trouverez toutes mes galleries</p>";  
 print "</div>";  
  
 print myFooter;  
  
};  
  
page galleryPaysage{  
 //parametres obligatoires  
 name : "Des paysages";  
 address : "./paysages.htm";  
  
 print myHead;  
 print myNav;  
 print "<div class='center'>";  
 print "<h1>Voici des paysages</h1>";  
 print "<table>";  
  
  
 for(i from 1 to 10){  
 print "<tr><td><img src='http://fakeimg.pl/290/'></td><td><img src='http://fakeimg.pl/290/'></td><td><img src='http://fakeimg.pl/290/'></td><tr>"  
 };  
  
 print "</table>";  
 print "</div>";  
};  
  
page galleryFamille{  
 //parametres obligatoires  
 name : "Ma famille";  
 address : "./famille1.htm";  
  
 print myHead;  
 print myNav;  
  
 print "<div class='center'>";  
 for(i from 1 to 20){  
 print "<p>Hello ma famille</p>"  
 };  
 print "</div>";  
};  
  
galleryHello += galleryPaysage;  
galleryHello += galleryFamille

**Fichier généré :**



# Exceptions gérées

* **« ; » manquant :**

Syntax error in line 35 at 'page'

* **Attributs obligatoires manquant pour les pages :**

Page must specify an address and a name. Error in page : galleryHello

Error, first args of a page must contain a 'address'

Error, first args of a page must contain a 'name'

Ces erreurs proviennent du script « generatorHTML » qui peut lever une exception si une page ne contient pas d’adresse au moment de la génération du menu.

* **Addition de pages invalide:**

Syntax Error: Addition requires two existing pages. Page galleryPaysage1 unknown. Line 87. Position 1739.

* **Erreur de syntaxe « / » au lieu de « // » par exemple :**

Syntax error in line 11 at '/'

* **Page inconnue dans le menu :**

Missing page adresse for : gallery

# Bugs connus

Le programme suit une récursivité qui nécessite qu’une instruction soit séparée d’une autre par un « ; », mais un « ; » de fin introduit un nouveau statement et donc un nœud vide sans type. Cela créé une erreur car une instruction nulle.

Nous n’avons malheureusement pas trouvé une solution adéquate.

# Remarques et conclusion

Lors de l’implémentation nous avons rapidement rencontrer les limitations de PLY, ce qui nous a compliqué la tâche.

En effet, notre langage particulier oblige l’utilisateur à respecter une syntaxe composée d’une suite de mots réservés dans un ordre précis et pouvant être utilisés que dans certaines situations (dans une page il faut au minimum les deux attributs, etc…).

Avec du recule, la syntaxe de notre langage s’approche plus d’un langage de description que d’un réel langage de programmation, s’il fallait refaire le projet nous choisirions une syntaxe plus proche de l’utilisation de PLY.

Nous avons donc dû adapter notre langage lors du déroulement du projet et faire avec les limitations de cet utilitaire.

Dans notre cas, l’utilisation d’un arbre syntaxique était alors moins évident. Le langage étant plutôt descriptif, nous avions directement la possibilité de générer notre propre structure de donnée à partir de l’analyse syntaxique et de la parser plus simplement par la suite.

Nous avons donc commencé par implémenter cette structure mais après discutions et vu du fait que la génération d’un AST est une contrainte de ce cours, nous avons en conséquence revu notre code.

Vous pouvez retrouver cette version sur notre Git.

Autre difficulté : nous générons directement du code HTML, il a donc été nécessaire de créer une structure HTML et CSS qui se prête à la génération de code. Notre langage est donc restreint par ces contraintes et le temps nécessaire à prévoir le code que l’utilisateur pourra générer et les types d’éléments adéquats pour la page sont trop limités.