HiComp

Compilateur HTML

Rapport de projet

Milán Cerviño & William Bikuta

Enseignant : François Tièche

Assistant : Edouard Goffinet

Classe : INF3dlm-a

Table des matières

1 Introduction

1.1 Contexte

Dans le cadre du cours de Compilateur de la filière Développement Logiciel et Multimédia à la HE-Arc Ingénierie, plusieurs travaux pratiques ont été réalisés pour créer un premier pseudo-compilateur en utilisant la librairie PLY (Python Lex-Yacc) de Python.

Ces travaux pratiques permettaient de comprendre dans les grandes lignes, les différents aspects d'un compilateur.

Afin de tester les compétences, il est demandé de réaliser un projet qui permet de compiler un langage au choix tout en mettant en pratique les points vus durant les travaux pratiques. Pour réaliser ce projet, le choix s'est porté sur un compilateur HTML qui permet d'interpréter un langage de description de document qui génère une page HTML. Le projet doit gérer quelques éléments de base et pouvoir gérer des conditions ainsi que des boucles.

1.2 Objectifs et fonctionnalités

Objectifs du compilateur

Le compilateur doit comporter les notions suivantes :

- Analyse lexicale du langage ;
- Analyse syntaxique du langage ;
- Analyse sémantique du langage.

La problématique avec les documents HTML est qu'il n'y a pas beaucoup de contraintes au niveau de l'analyse sémantique. Le langage HTML ne produit pas d'erreur particulière lorsque les attributs des éléments n'ont pas de sens. Afin de pouvoir implémenter l'analyse sémantique, une règle vérifie l'utilisation des couleurs en s'assurant qu'un texte de couleur ne puisse être sur un fond de même couleur, ou presque, puisque dans ce cas le texte ne sera pas lisible. Cette idée a été suggérée par l'enseignant. L'analyse sémantique diffère du cahier des charges qui propose la vérification de l'imbrication des éléments, mais il paraissait plus intéressant de contrôler les couleurs.

Fonctionnalités du langage

Le langage doit gérer au minimum les éléments suivants :

- Head et Title ;
- Body ;
- Header → h1, h2 ;
- Paragraphe de texte (<p>) ;
- Liste à puce (<ul> +) ;
- Conteneur ( ) ;
- Condition et boucle.

Le langage doit également permettre la gestion des attributs de type background-color et color, ainsi que l'affichage conditionnel avec des if … else en lien avec les attributs de couleur des éléments parents et la gestion de boucle for pour afficher plusieurs éléments similaires.

Le compilateur et les différentes fonctionnalités à implémenter doivent gérer le code de base suivant :

```
head { title "Titre de la page"; }
body {
    header1 "Chapitre 1";
    header2 "Chapitre 1.1";

    div backgroundcolor = "black" {
      if (backgroundcolor is "black") {
            text color = "white" "Mon texte de description";
            text color = "green" "Mon deuxieme texte";
      }
      else {
            text color = "black" "Mon texte de description";
            text color = "red" "Mon deuxieme texte";
      }
    }

    list {
        for (i = 1 to 6) {
            element "element %i de ma liste";
        }
    }
}
```

2 Description du langage

2.1 Déclaration d'un élément

Pour déclarer un nouvel élément, il faut tout d'abord écrire son nom tel que header1, body, … La liste des éléments disponible est gérée par le langage figurant dans le tableau du chapitre 4.

Ce tableau peut augmenter en ajoutant de nouvelles balises dans le dictionnaire qui se trouve dans le fichier *parser2.py*.

Après avoir nommé la balise, il est possible d'ajouter un attribut de couleur (impossible de chaîner plusieurs attributs) sous la forme suivante :

`backgroundcolor = "black" ou color = "red".`

Finalement, il faut spécifier le contenu de l'élément soit par un texte entre guillemets suivi d'un point-virgule pour les éléments de type simple, soit avec les accolades `"{ … }"` pour éléments de type bloc.

2.2 Condition et boucle

2.2.1 Condition if … else

Les conditions s'utilisent pour afficher un contenu si l'attribut spécifié de l'élément parent a une certaine valeur. Les conditions s'appliquent donc sur la couleur de fond ou de texte, de l'élément parent. Ces conditions permettent d'éviter des erreurs qui peuvent être soulevées dans l'analyse sémantique. De plus, elles permettent la mise en forme conditionnelle et d'implémenter plusieurs affichages en fonction de la couleur de fond. Cela permet d'éviter de changer tous les attributs des enfants si l'élément parent change de couleur de fond par exemple.

La syntaxe est la suivante :

```
if (backgroundcolor is not "black") {
      text color = "white" "Mon texte de description";
      text color = "green" "Mon deuxieme texte";
}
else {
      text color = "black" "Mon texte de description";
      text color = "red" "Mon deuxieme texte";
}
```

Les éléments optionnels sont :

- Le « not » qui permet d'inverser le test (backgroundcolor is not black) ;
- Le bloc else qui permet d'afficher un contenu différent si le test est faux.

2.2.2 Boucle for

La boucle for permet de répéter un contenu similaire plusieurs fois de suite. Il est possible d'utiliser la variable i pour faire varier le contenu des éléments de la boucle.

La syntaxe est la suivante :

```
for (i = 1 to 6) {
      element "element %i de ma liste";
}
```

La boucle reçoit une variable (ici « i ») qui reçoit une valeur de départ, 1 dans ce cas-ci. La variable s'incrémente tant qu'elle est inférieure ou égale à la valeur finale (ici 6). Le contenu de la boucle sera donc répété 6 fois ici.

3 Développement

3.1 Analyse lexicale

L'analyseur lexicale du compilateur se trouve dans le fichier *lexer.py* du projet. Celui-ci permet convertir les chaines de caractères du code en différents types :

- *NUMBER* : Nombre qui permet de définir l'intervalle de la boucle for ;
- *STRING* : Chaîne de caractère utile pour le contenu des différents éléments simples, ainsi que la valeur des attributs de type couleur ;
- *IDENTIFIER* : Permet d'identifier les différents types de balise, le nom des variables pour la boucle for ou encore le nom des attributs.

Pour le reste, il y a des mots réservés tels que : if, else, for, to, etc…

Ils sont utilisés pour les conditions et les boucles. Finalement, certains caractères spéciaux sont acceptés et permettent, tout comme les accolades pour la délimitation des éléments de type bloc, les conditions if ou les boucles.

Les différents mots et caractères qui sont acceptés dans l'analyse lexicale, peuvent être utilisé ensuite dans l'analyse syntaxique pour définir les différentes expressions régulières.

3.2 Analyse syntaxique

3.2.1 Eléments

Les règles de la partie analyse syntaxique imposent que les éléments du programme respectent la forme suivante :

- ***assignation*** *:* ***IDENTIFIER attribut content*** ou ***IDENTIFIER content***
  - ***attribut*** *:* ***parameter '=' STRING***
    - ***parameter*** *:* ***IDENTIFIER***
  - ***content*** *:* ***bloc*** ou ***element***
    - ***bloc*** *:* ***'{' programme '}'***
    - ***element*** *:* ***expression ' ;'***

Afin de vérifier les IDENTIFIER, le parser contient deux dictionnaires, un pour les éléments et l'autre pour les noms d'attributs. Ainsi, si l'élément ou l'attribut ne figure pas dans le dictionnaire correspondant, alors le compilateur affiche un message d'erreur et ignore l'élément ou l'attribut (voir exemple *error1.txt* et *error2.txt*). Cette « erreur » n'est pas réellement un problème de syntaxe, puisque la syntaxe est respectée, mais les éléments ou attributs peuvent ne pas être traités dans le parser. Lorsque le parser va rechercher dans les dictionnaires, le programme va planter. C'est pourquoi le compilateur ne va pas s'arrêter mais simplement ignorer ce qui n'est pas accepté.

L'utilisation des dictionnaires permet au langage Hicomp de s'éloigner du langage HTML en se laissant libre choix du nom des éléments qui sont acceptés et en leurs donnant une

équivalence HTML. De plus, il est plus facile de répertorier les valeurs par défauts et pourquoi pas d'autres spécificités sur les éléments et attributs.

3.2.2 Conditions

Les règles syntaxiques pour les conditions sont les suivantes :

- **condition** : **if '(' parameter IS** ou **IS NOT ')' bloc** et éventuellement **ELSE bloc**
  - ***parameter** : **IDENTIFIER***
  - ***bloc** : **'{' programme '}'***
  - ***if : IF***

L'affichage conditionnel est géré dans la fonction show() de la classe IfNode. Le paramètre fait référence à l'attribut du parent s'il existe. S'il n'existe pas, la valeur par défaut est spécifiée dans le dictionnaire des attributs selon les valeurs par défaut utilisées en CSS (par exemple la valeur par défaut de background-color et white).

L'utilisation du if et parameter peut paraitre inutile puisqu'il est possible de les remplacer par IF et IDENTIFIER, mais la fonction p_if(p) permet de modifier une variable globale en incrémentant le nombre de if actifs. Cette variable globale permet d'ignorer l'analyse sémantique des couleurs sur les éléments qui sont « protégées » par un if. Sans cette variable globale, le compilateur peut afficher des warnings de certains éléments qui sont dans le bloc if, ce qui ne fait pas de sens. Pour le parameter, la fonction p_parameter(p) permet de vérifier l'existence de l'attribut dans le dictionnaire.

3.2.3 Boucles

Les règles syntaxiques pour les boucles sont les suivantes :

- ***loop** : **FOR '(' IDENTIFIER '=' NUMBER TO NUMBER ')' bloc***
  - ***bloc** : **'{' programme '}'***

L'affichage à répétition du bloc spécifié se fait dans la fonction show() de la classe ForNode. La boucle se fait dans l'intervalle [start, end], le paramètre end est donc inclus dans l'intervalle de répétition. Exemple for (1 to 6) est l'équivalent de for (int i = 1 ; i < 7 ; i++) dans un langage courant.

Si le 1er nombre est plus grand que le second, alors le compilateur affiche un message d'erreur et la boucle est ignorée.

3.2.4 Classes Node

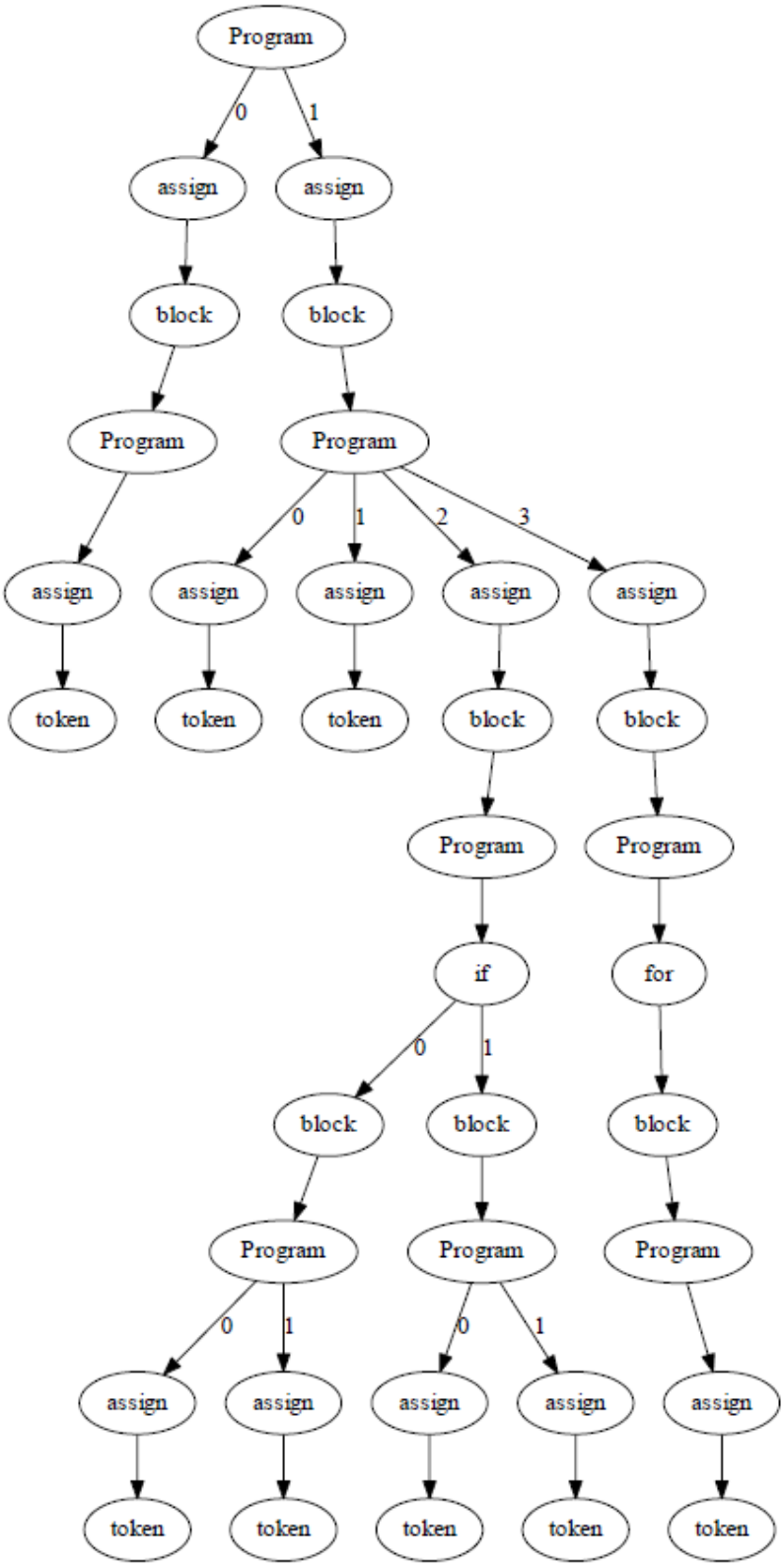
Dans le fichier *Node.py*, se trouve la déclaration de plusieurs classes de type Node qui représente les éléments de l'analyse syntaxique. La classe Node est la classe parente à tous les autres nœuds. Elle permet de gérer les variables communes des nœuds et éviter la répétition de code pour la fonction show() de chaque nœud.

En effet, la fonction show() de la classe parente contient une partie commune de l'affichage aux autres types de nœud qui est l'affichage des éléments enfants du nœud. De plus, la classe parente contient les fonctions permettant la génération de l'arbre syntaxique du programme.

Comme vu dans le cours l'utilisation d'une structure arborescente pour l'analyse syntaxique est fortement conseillée, car cela rend la tâche plus simple.

3.2.5 Arbre syntaxique

Ci-dessous, se trouve l'arbre syntaxique de l'exemple qui figure dans la partie d'introduction et que l'on retrouve également dans le fichier *base.txt* des exemples.



Pour cet arbre syntaxique on remarque que le premier nœud Program, la racine de l'arbre, a deux enfants qui sont les blocs head et body du code.

Le premier bloc (head) ne contient qu'un seul élément simple qui est représenté par la feuille Token de la branche gauche de la racine.

Le second bloc (body) contient 4 autres éléments (2 éléments simples et 2 blocs). Parmi les deux blocs, on retrouve un div qui contient if. Ce bloc if contient deux blocs, un pour la condition if et l'autre pour le else. L'autre bloc est la boucle for qui contient un autre bloc avec un seul élément simple. On remarque qu'un bloc est composé d'un programme.

3.3 Analyse sémantique

Comme cité dans l'introduction, l'analyse sémantique n'est pas très utile pour le langage HTML puisqu'un fichier HTML ne retourne aucune erreur. En effet le langage ne tient pas compte des attributs CSS de ces éléments d'un point de vue sémantique. Pour ce projet, quelques règles sémantiques ont été implémentées afin d'améliorer le compilateur.

Ces règles sont basées majoritairement sur les attributs couleurs des différents éléments. Le parser utilise une pile qui permet d'empiler les couleurs des éléments parents afin de tester ces couleurs sur les éléments enfants.

L'analyse sémantique s'applique sur les cas suivants :

- Test de l'attribut couleur d'un élément avec l'attribut couleur du parent si l'élément n'est pas protégé par un if.
    - Si l'élément n'a pas d'attribut, le compilateur teste si le background-color de l'élément parent n'est pas noir pour éviter un texte noir sur fond noir. Noir étant la couleur par défaut des textes en HTML.
    - Si le parent n'a pas d'attribut, la pile remonte à l'attribut parent le plus proche et si la pile est vide alors le compilateur teste avec un fond blanc. Blanc étant la couleur de fond par défaut des blocs en HTML.
- Test si la couleur existe dans la liste des couleurs supportées par CSS. Cette liste de couleur provient de matplotlib.colors.
- Test si un attribut ou une balise figure dans les dictionnaires. Le cas échéant le compilateur renvoie une erreur et ignore la balise.
- Pour une boucle test que la valeur de départ ne soit pas plus grande que la valeur d'arrivée. Le cas échant le compilateur renvoie une erreur et ignore la boucle.

Pour le test des attributs couleur, il y a deux types d'analyse :

Le compilateur test dans un premier temps que deux couleurs ne soit pas exactement les mêmes. Le cas échéant, un warning apparait et prévient l'utilisateur que le texte ne sera pas lisible.

Dans un second temps le compilateur test la distance entre deux couleurs. Cette distance n'est autre que la somme des distances de chaque composante Rouge, Vert, Bleu des deux couleurs.

$$\text{distance} = \sqrt{(R_2 - R_1)^2 + (G_2 - G_1)^2 + (B_2 - B_1)^2}$$

Source : https://en.wikipedia.org/wiki/Color_difference

Les composantes RGB sont récupérées par matplotlib.colors. Si la distance est inférieure à un seuil fixé (dans le projet 0.15), alors le compilateur considère les couleurs comme trop proches et renvoie un warning en conséquence.

4 Guide utilisateur

4.1 Prérequis

Il faut avoir installé Python (minimum version 3.5) sur votre ordinateur.

4.2 Installation

Pour installer le compilateur, il faut tout d'abord cloner le repository en veillant à bien être dans le réseau de l'école.

```
git clone git@gitlab-etu.ing.he-arc.ch:milan.cervino/compilateur-hicomp.git
```

Ensuite, il faut installer les différents packages comme **PLY** :

Sous Debian:

```
apt install python-ply
```

Sous Windows :

Il faut récupérer le zip d'installation à cette adresse : http://www.dabeaz.com/ply/

Ensuite il faut le décompresser et l'installer.

```
python setup.py install
```

Maintenant, il faut installer **Graphviz :**

Sous Debian :

```
apt install graphviz
```

Sous Windows :

Pour ce faire, il faut se rendre à cette adresse : http://www.graphviz.org/download

Pour finir, il faut encore installer **PyDot** :

Il y a deux manières d'installer PyDot

Via ligne de commande :

```
pip install pydot
```

Via le site officiel :

PyDot est disponible à cette adresse : https://pypi.org/project/pydot/#files

4.3 Exécution

Pour exécuter le compilateur, il faut tout d'abord ouvrir un invité de commande dans le dossier du projet à la racine.

Pour afficher le lexème, entrez la commande suivante :

```
python lexer.py examples/[NameFile].txt
```

Pour générer l'arbre syntaxique, entrez la commande suivante :

```
python parserHiComp.py examples/[NameFile].txt
```

Pour générer la page HTML résultante, entrez la commande suivante :

```
python interpréter.py examples/[NameFile].txt
```

Note : *NameFile* représente le nom d'un fichier qui figure dans le dossier examples.

4.4 Créer son propre programme

4.4.1 Balises

Le code s'écrit de la façon suivante → NomBalise Attribut Contenu :

- NomBalise → Nom de la balise (se référer au tableau ci-dessous)
- Attribut **(Optionnel)** → NomAttribut = ''couleur''
  - NomAttribut → Nom de l'attribut (se référer au tableau ci-dessous)
- Contenu → Texte entre guillemets ou { Programme }.
  - Programme → Suite du programme

4.4.2 Conditions

Dans le cas où l'on souhaiterait utiliser une condition, cette dernière s'écrit :

Pour tester si un attribut vaut une valeur :

if NomAttribut is ''valeur'' { Programme }

Pour tester si un attribut est différent de la valeur :

if NomAttribut is not ''valeur'' { Programme }

On peut également ajouter une alternative au test si celui-ci est refusée avec :

else { Programme }

Dans les cas ci-dessus :

- NomAttribut → Nom de l'attribut
- Programme → Suite du programme si condition acceptée.

4.4.3 Boucles

Dans le cas où l'on souhaiterait répéter un morceau du programme, il est possible d'utiliser une boucle for selon la syntaxe suivante :

for (NomVariable = StartValeur to EndValeur) { Programme }

- StartValeur → Valeur de depart
- EndValeur → Valeur d'arrivée
- Programme → Suite du programme à répéter

Note : La boucle s'exécute dans l'intervalle [StartValeur, EndValeur]

4.4.4 Éléments employables

Liste des balises employables :

Balise HiComp	Équivalent HTML	Description
header1	h1	Titre de niveau 1
header 2	h2	Titre de niveau 2
body	body	Corps de la page
head	head	En-tête
title	title	Titre
text	p	Paragraphe
list	ul	Liste à puce
element	li	Elément de la liste
div	div	Section

Liste des attributs employables :

Attribut HiComp	Attribut CSS	Valeur par défaut	Description
backgroundcolor	background-color	white	Couleur de fond
color	color	black	Couleur du texte

Un exemple de code est disponible dans l'introduction de ce document.

5 Résultat obtenu

Exemple de résultat que l'on obtient avec le fichier **base.txt**.

Ce fichier est disponible dans le dossier **examples** à la racine du projet.

Code Hicomp

```
head { title "Titre de la page"; }
body {
    header1 "Chapitre 1";
    header2 "Chapitre 1.1";

    div backgroundcolor = "black" {
      if (backgroundcolor is "black") {
            text color = "white" "Mon texte de description";
            text color = "green" "Mon deuxieme texte";
      }
      else {
            text color = "black" "Mon texte de description";
            text color = "red" "Mon deuxieme texte";
      }
    }

    list {
        for (i = 1 to 6) {
            element "element %i de ma liste";
        }
    }
}
```

Résultat

```html
<!DOCTYPE html>
<html>
    <head>
        <title>
            Titre de la page
        </title>
    </head>
    <body>
        <h1>
            Chapitre 1
        </h1>
        <h2>
            Chapitre 1.1
        </h2>
        <div style="background-color:black;">
            <p style="color:white;">
                Mon texte de description
            </p>
            <p style="color:green;">
                Mon deuxieme texte
            </p>
        </div>
        <ul>
            <li>
                element 1 de ma liste
            </li>
            <li>
                element 2 de ma liste
            </li>
            <li>
                element 3 de ma liste
            </li>
            <li>
                element 4 de ma liste
            </li>
            <li>
                element 5 de ma liste
            </li>
            <li>
                element 6 de ma liste
            </li>
        </ul>
    </body>

</html>
```

6 Conclusion

Pour conclure, le langage HiComp permet de traduire du code simple à rédiger en un code HTML/CSS qui peut parfois poser problème.

Les choix d'analyse sémantique et d'autres règles étaient assez libres puisque le langage HTML n'a pas beaucoup de restriction. Ces règles ont été choisies dans le but de découvrir et améliorer les compétences dans le cours de compilateur.

Les fonctionnalités citées dans le cahier des charges ont été implémentées et d'autres idées sont venues au cours du développement.

Il existe plusieurs améliorations possibles pour ce compilateur. Notamment :

- L'ajout de tableaux
- Plus de balises HTML
- Gestion de plus d'attributs CSS
- Complexification du langage avec d'autres boucles (WHILE)
- Complexification du langage avec la condition SWITCH.

Milán Cerviño & William Bikuta