Projet de compilation
Moteur de template *Dumbo*

Alexandre Decan

Année académique 2022-2023, première session

1 Énoncé

Vous devez créer un programme, nommé “dumbo.py”, qui permet de générer facilement un fichier contenant du texte. La génération se fait en fonction de données reçues en paramètres. Pour ce faire, un langage est créé: le *dumbo*.

Le programme “dumbo.py” prend deux paramétres:

- le premier paramètre est le nom d'un fichier contenant du code dumbo déclarant des variables et des données. Ce fichier est appelé *fichier data*;
- le second paramètre est le nom d'un fichier contenant du texte et du code dumbo dans lequel on va injecter les données. Ce fichier est appelé *fichier template*;

La génération du résultat se passe comme suit:

- le programme lit le fichier data et initialise les variables qui s'y trouvent,
- le programme lit le fichier template, et y injecte les variables en respectant les règles décrites plus loin dans ce document,
- le résultat est imprimé sur la sortie standard (*stdout*).

1.1 Exemple de fonctionnement du programme

Voici un exemple de fichier data. Il déclare les variables `nom, prenom` et `cours` et les initialise.

```
{{
nom := 'Brouette';
prenom := 'Quentin';
cours := ('Logique 1', 'Logique 2', 'Algebre 1', 'Math elem.');
}}
```

Remarquez que l'affectation se fait via `:=` et que chaque ligne se termine par `;`.

Voici un exemple de fichier template. Ce fichier mélange du texte au format HTML ainsi que des blocs de code dumbo (entourés par {{...}}).

```
<html>
  <head><title>{{ print nom.' '.prenom; }}</title></head>
  <body>
  <h1>{{ print nom.' '.prenom; }}</h1>
  Cours: {{ for c in cours do
              print c.', ';
            endfor; }}
  </body>
</html>
```

Résultat affiché après exécution de la commande
"python dumbo.py data template" :

```
<html>
  <head><title>Brouette Quentin</title></head>
  <body>
  <h1>Brouette Quentin</h1>
      Cours: Logique 1, Logique 2, Algebre 1, Math elem.,
  </body>
</html>
```

Le programme a généré une sortie en HTML à partir du fichier data et du fichier template. Les données déclarées dans le fichier data ont été injectées dans le fichier template. L'intérêt d'un tel programme est que si on veut générer une page HTML pour une autre personne, il suffit de créer un second fichier data, par exemple:

```
{{
nom := 'De Pril';
prenom := 'Julie';
cours := ('Math discretes');
}}
```

... et d'exécuter la commande "python dumbo.py data2 template" qui produira un nouveau résultat avec les nouvelles données:

```
<html>
  <head><title>De Pril Julie</title></head>
  <body>
  <h1>De Pril Julie</h1>
      Cours: Math discretes,
  </body>
</html>
```

Ce genre de programme est appelé *moteur de templates* et est régulièrement utilisé au sein de framework de développement web tels que Ruby on Rails (en Rubyg), Django (en Python) ou Vue.js (en JavaScript).

1.2 Grammaire de base du dumbo

Voici la grammaire de base du dumbo. Il vous sera demandé de la compléter et de la modifier par la suite afin d'y ajouter d'autres fonctionnalités.

$< programme >$	$\longrightarrow$	$< txt > \mid < txt > < programme >$
$< programme >$	$\longrightarrow$	$< dumbo\_bloc > \mid < dumbo\_bloc > < programme >$
$< txt >$	$\longrightarrow$	...
$< dumbo\_bloc >$	$\longrightarrow$	$\{\{ < expressions\_list > \}\}$
$< expressions\_list >$	$\longrightarrow$	$< expression > ; < expressions\_list >$
$< expressions\_list >$	$\longrightarrow$	$< expression > ;$
$< expression >$	$\longrightarrow$	$print < string\_expression >$
$< expression >$	$\longrightarrow$	$for < variable > in < string\_list >$ $do < expressions\_list > endfor$
$< expression >$	$\longrightarrow$	$for < variable > in < variable >$ $do < expressions\_list > endfor$
$< expression >$	$\longrightarrow$	$< variable > := < string\_expression >$
$< expression >$	$\longrightarrow$	$< variable > := < string\_list >$
$< string\_expression >$	$\longrightarrow$	$< string >$
$< string\_expression >$	$\longrightarrow$	$< variable >$
$< string\_expression >$	$\longrightarrow$	$< string\_expression > . < string\_expression >$
$< string\_list >$	$\longrightarrow$	$(< string\_list\_interior >)$
$< string\_list\_interior >$	$\longrightarrow$	$< string > \mid < string >, < string\_list\_interior >$
$< variable >$	$\longrightarrow$	...
$< string >$	$\longrightarrow$	...

Vous devez compléter cette grammaire afin de respecter les éléments suivants :

1. Le texte ($< txt >$) est composé de n'importe quel caractère, à l'exception du marqueur de début de bloc Dumbo ({{);
2. Les chaînes de caractères ($< string >$) sont composées de n'importe quelle suite de caractères entourées de guillemets simples (');
3. Les noms de variables contiennent uniquement des caractères alphanumériques ou underscore _ et ne peuvent débuter par un chiffre;

Ensuite modifiez la grammaire pour:

- gérer les entiers, les opérations +-*/, et les variables entières;
- gérer les booléens "true" et "false" et les opérations "or" et "and";
- gérer la comparaison des entiers (retournant un booléen):
  - inférieur : " $< integer > < < integer >$"
  - supérieur : "$< integer > > < integer >$"
  - égal : "$< integer > = < integer >$"
  - différent : "$< integer > ! = < integer >$" ;

- gérer le "if" : "*if* < *boolean* > *do* < *expressions_list* > *endif*" ;

Une fois ces modifications faites, votre programme doit pouvoir gérer le fichier template suivant:

```
<html>
  <head><title>{{ print nom; }}</title></head>
  <body>
      <h1>{{ print nom; }}</h1>
      {{
          i := 0
          for nom in liste_photo do
            if i > 0 do print ', '; endif;
            print '<a href="'.nom.'">'.nom.'</a>';

            i := i + 1;
          endfor;
      }}
      <br />
      Il y a {{ print i; }} dans l album {{ print nom; }}.
  </body>
</html>
```

2 Informations pratiques

2.1 Instructions

- Le devoir se fait par groupe de **deux** étudiants. La composition des groupes doit me parvenir avant le **vendredi 1er avril à 12h00** (alexandre.decan@umons.ac.be).
- Le devoir est à rendre sur Moodle pour le **lundi 15 mai à 18h00** au plus tard.
- Vous devez remettre le code python et un rapport dans lequel se trouvent notamment
  - une description de la grammaire complétée et modifiée,
  - une explication de l'analyse sémantique (incluant notamment la structure de données utilisées, la gestion des scopes et l'implémentation des boucles),
  - une présentation des problèmes que vous avez rencontrés et les solutions envisagées/appliquées,
  - la répartition du travail au sein du groupe.
- L'implémentation est à réaliser soit à l'aide de `lark`[1], soit à l'aide de `ply`[2] et doit fonctionner avec Python 3.7 ou supérieur.
- Ne remettez que les fichiers nécessaires à votre projet (rapport, code, etc). N'incluez pas de fichiers facultatifs (exemples, dossiers/fichiers cachés, artefacts générés par PLY).
- Fournissez un fichier `requirements.txt` listant les dépendances (et les versions) nécessaires à l'exécution de votre projet.

[1] `https://github.com/lark-parser/lark`, disponible sur PyPI sous le nom `lark`.
[2] `https://www.dabeaz.com/ply/`, disponible sur PyPI sous le nom `ply`.

2.2 Remarques

Votre grammaire ne peut comporter de conflits, et doit être LALR(1).

La règle "$< string\_expression > \longrightarrow < string\_expression > . < string\_expression >$" est interprétée comme la concaténation de deux chaînes de charactères.

Dans la règle "$< expression > \longrightarrow for < variable > in < variable > do < expressions\_list > endfor$", la seconde $< variable >$ est supposée être une variable contenant une $< string\_list >$. L'utilisateur respecte cette convention.

Dans la règle "$< string\_expression > \longrightarrow < variable >$", la $< variable >$ est supposée contenir une $< string\_expression >$. L'utilisateur respecte cette convention.

Veuillez faire attention à la portée des variables. Les modifications d'une variable dans un <dumbo_bloc> sont transmises aux <dumbo_bloc> suivants. Mais après

```
for nom in liste do
  ...
endfor;
```

la variable "nom" reprend sa valeur précédente si elle avait été initialisée.

2.3 Conseils

Il vous est fortement conseillé de créer le programme étape par étape afin de pouvoir débugger plus facilement.

Des fichiers d'exemple et leur sortie se trouvent sur la plateforme moodle. Vérifiez que votre programme génère la sortie demandée pour ces exemples.

L'évaluation de votre travail portera aussi bien sur le fond et la forme de votre rapport, que sur l'aspect fonctionnel de votre programme. La qualité de l'architecture et la qualité du code sont également des facteurs intervenant dans la note finale. Il existe de nombreux outils en Python (pep8, flake8, black, etc.) pour vous aider à produire du code de qualité. N'hésitez pas à en abuser.

Enfin, pensez à relire attentivement l'ensemble des consignes avant de remettre votre travail.

Vous pouvez, bien entendu, me contacter (alexandre.decan@umons.ac.be) si vous avez des questions.

Bon travail !