Devoir - compilation

Année académique 2016-2017

Intructions

- Le devoir se fait par groupes de 2 étudiants (veuillez indiquer la composition des groupes par email (alexandre.decan@umons.ac.be) avant le 26 avril).
- Le devoir est à rendre pour le mercredi 17 mai à 12h00.
- La remise du devoir se fait via moodle.
- Vous devez remettre le code python et un rapport dans lequel se trouvent
  - une description de la grammaire,
  - une explication de la manière dont vous avez géré le “for” et le “if”,
  - une présentation des problèmes que vous avez rencontrés.

Énoncé

Vous devez créer un programme, nommé “dumbo_interpreter.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_interpreter.py” prend trois paramétres:

- le premier paramètre est le nom d’un fichier contenant du code dumbo, ce fichier est appelé *fichier data*;
- le second paramètre est le nom d’un fichier contenant du texte dans lequel du code dumbo est injecté, ce fichier est appelé *fichier template*;
- le dernier paramètre désigne le nom du *fichier output*.

La génération du code se passe comme suit:

- le programme lit le fichier data et initialise les variables qui s’y trouvent,
- il crée ensuite le fichier output dont le contenu est le contenu du fichier template dont le code dumbo a été évalué.

Grammaire de base du dumbo.

Voici la grammaire de base du dumbo. Il vous sera demandé de la modifier afin d'y ajouter des fonctionnalités.

Pour simplifier, nous limitons le texte à $[a-zA-Z0-9;\&<>"\_-./\backslash\ \backslash n\backslash p:,]+$. Le charactère $'$ n'est donc pas compris.

$<programme>$	$\longrightarrow$	$<txt> \mid <txt><programme>$
$<programme>$	$\longrightarrow$	$<dumbo\_bloc> \mid <dumbo\_bloc><programme>$
$<txt>$	$\longrightarrow$	$[a-zA-Z0-9;\&<>"\_-.\backslash/\ \backslash n\backslash p:,]+$
$<dumbo\_bloc>$	$\longrightarrow$	$\{\{ <expressions\_list> \}\}$
$<expressions\_list>$	$\longrightarrow$	$<expression> ; <expressions\_list>$
$<expressions\_list>$	$\longrightarrow$	$<expression> ;$
$<expression>$	$\longrightarrow$	$print <string\_expression>$
$<expression>$	$\longrightarrow$	$for <variable> in <string\_list>$ $do <expressions\_list> endfor$
$<expression>$	$\longrightarrow$	$for <variable> in <variable>$ $do <expressions\_list> endfor$
$<expression>$	$\longrightarrow$	$<variable> := <string\_expression>$
$<expression>$	$\longrightarrow$	$<variable> := <string\_list>$
$<string\_expression>$	$\longrightarrow$	$<string>$
$<string\_expression>$	$\longrightarrow$	$<variable>$
$<string\_expression>$	$\longrightarrow$	$<string\_expression> . <string\_expression>$
$<string\_list>$	$\longrightarrow$	$(<string\_list\_interior>)$
$<string\_list\_interior>$	$\longrightarrow$	$<string> \mid <string>, <string\_list\_interior>$
$<variable>$	$\longrightarrow$	$[a-zA-Z0-9\_]+$
$<string>$	$\longrightarrow$	$'\ [a-zA-Z0-9;\&<>"\_-.\backslash/\ \backslash n\backslash p:,]+\ '$

Exemple de fonctionnement du programme.

Fichier data1.dumbo :

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

Fichier template.dumbo :

```
<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>
```

Fichier output.html après exécution de la commande
"dumbo_interpreter.py data1.dumbo template.dumbo output.html" :

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

Remarques.

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.

Le programme a généré un fichier HTML à partir du fichier data et du fichier template.

L'intérêt du programme est que si on veut générer une page HTML pour une autre personne, il suffit de créer un fichier data2.dumbo:

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

Et d'exécuter la commande "dumbo_interpreter.py data2.dumbo template.dumbo output2.html" qui créera un fichier output2.html contenant:

```
<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 template* et est régulièrement utilisé au sein de framework web tels que Ruby on Rails ou Django.

Conseils.

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

La première étape est de créer l'analyseur lexical. Définissez les lexèmes en utilisant les fonctions (**def** t_NomLexeme(t): ....) ; de cette manière la priorité est donnée aux lexèmes déclarés en premier.

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

Cahier de charge.

Dans un premier temps, veuillez créer le programme associé à la grammaire décrite ci-dessus et ayant le comportement décrit par l'exemple de l'énoncé.

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 initiale.

Ensuite modifiez la grammaire et le programme 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 les fichiers template suivants:

```
<html>
  <head><title>{{ print nom; }}</title></head>
  <body>
      <h1>{{ print nom; }}</h1>
      {{
          i := 0
          for nom in liste_photo do
```

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

```
<html>
  <head><title>{{ print nom.' '.prenom; }}</title></head>
  <body>
  <h1>{{ print nom.' '.prenom; }}</h1>
  Cours: {{
      i := 0;
      for c in cours do
        print c
        if i != dernier_elem_list do print ', '; endif;
        i := i + 1;
      endfor; }}
  </body>
</html>
```

Pour le second exemple, la variable " dernier_elem_list " contient un entier qui est la taille de la liste "cours".

La gestion du "if" et du "for" est la partie la plus importante du devoir. Si votre programme ne gère pas cette fonctionnalité, peu de points seront accordés à votre travail. De plus, un programme qui ne marche pas sera pénalisé.

Si vous avez le temps, vous pouvez ajouter aux variables des types afin de détecter lorsqu'une variable est mal utilisée (par ex. concaténer une variable contenant une $< string\_list >$).

Bon travail! Si vous avez des questions our si vous êtes bloqué, contactez-moi par mail.