Text to Web Compiler

TP de Automates, Langages et Compilation

Nicolas Baudru - Polytech Marseille - Département Informatique

Introduction

Objectif

Ecrire un programme permettant de traduire un fichier texte faiblement formaté en du code HTML avec

- numérotation des sections,
- génération automatique d'une table des matières avec liens hypertextes,
- génération automatique d'une table d'index avec liens hypertextes.

Le programme sera écrit en langage C.

Comment ?

Nous allons procéder par étape :

1. A partir d'une description des entités élémentaires utilisables dans le fichier texte (aussi appellés tokens ou lexèmes), nous modéliserons un automate fini déterministe les reconnaissant.
2. A partir de l'automate fini, nous implémenterons un analyseur lexical, ainsi qu'un programme de test qui affichera dans un terminal la suite des lexèmes trouvés.
3. Nous vérifierons que la grammaire décrivant la structure des fichiers textes acceptés en entrée est bien LL(1) et nous décrirons la table prédictive.
4. A partir de la table prédictive, nous implémenterons l'analyseur syntaxique comme une suite de fonctions mutuellement

récursives, ainsi qu'un programme de test qui affichera dans un terminal l'arbre de dérivation correspondant au fichier texte lu en entrée.

5. Nous ajouterons sans les fonctions utilisées pour l'analyse syntaxique du code permettant de traduire le fichier texte dans un format HTML.

(Si le temps le permet :)

6. Nous ajouterons un mécanisme de numérotation automatique des sections et sous sections
7. Nous génèrerons automatiquement une table des matières en début de page avec liens hypertextes
8. Nous génèrerons automatiquement une table d'index avec liens hypertextes construite à partir des mots #tagués dans le fichier texte.

Voici un exemple de fichier texte accepté en entrée :

```
>Titre Exemple de fichier texte
>Auteur Nicolas

Ce fichier vous montre a quoi ressemble un fichier texte
lu en entree. Pour eviter les problemes de format, les
accents ne sont pas autorises. Un tel fichier commence
toujours par le mot cle Titre, precede d'un chevron, et
suivi du titre du texte; puis du mot cle Auteur precede
d'un chevron suivi du nom de l'auteur.
Vient ensuite un ou plusieurs paragraphes optionnels
comme celui que vous lisez.

Les paragraphes sont separes par une ou plusieurs lignes
blanches. Vous etes donc entrain de lire un deuxieme
paragraphe. Les sections sont introduites par le symbole
plus suivi du titre, comme ci-dessous.

= Ma premiere section

Ici vous pouvez ecrire vos paragraphes de section. Au
besoin vous pouvez definir des sous sections, en les
```

```
introduisant par deux symboles plus suivi du titre de
sous section, comme ci-dessous.

== Ma premiere sous Section

Ici vous pouvez ecrire vos paragraphes de section
toujours separes par une ou plusieurs lignes blanches.
Il peut bien sur avoir plusieurs sous sections dans une
meme section. En voici une deuxieme.

== Au sujet des indexes

Au besoin vous pouvez aussi ajouter des mots en #index en
lui accolant devant un #symbole diese. Dans la phrase
precedente les mots index et symbole seront donc mis en
index lors du processus de compilation.

= Au sujet de la table des matieres et des numeros de
section

La numerotation des sections et sous sections se fera
automatiquement lors du processus de compilation. On
utilisera une numerotation du style 1.3. qui signifiera
sous section 3 de la section 1. Actuellement vous etes en
train de lire la section 2.

Une table des matieres sera aussi generee lors de la
compilation et inseree en tout debut de page HTML.
```

Et voici le résultat attendu après compilation :

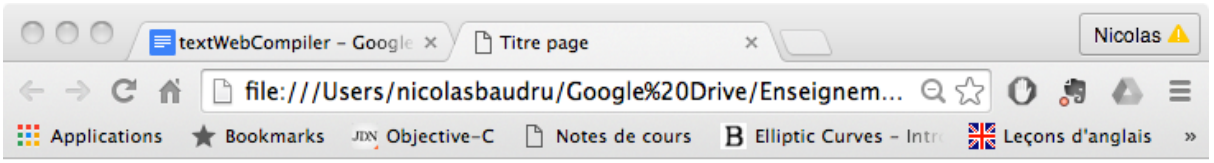


Table des matieres

Exemple de fichier texte

Nicolas

Ce fichier vous montre a quoi ressemble un fichier texte lu en entree. Pour eviter les problemes de format, les accents ne sont pas autorises. Un tel fichier commence toujours par le mot cle Titre, precede d'un chevron, et suivi du titre du texte; puis du mot cle Auteur precede d'un chevron suivi du nom de l'auteur. Vient ensuite un ou plusieurs paragraphes optionnels comme celui que vous lisez.

Les paragraphes sont separes par une ou plusieurs lignes blanches. Vous etes donc entrain de lire un deuxieme paragraphe. Les sections sont introduites par le symbole plus suivi du titre, comme ci-dessous.

1 Ma premiere section

Ici vous pouvez ecrire vos paragraphes de section. Au besoin vous pouvez definir des sous sections, en les introduisant par deux symboles plus suivi du titre de sous section, comme ci-dessous.

1.1 Ma premiere sous Section

Ici vous pouvez ecrire vos paragraphes de section toujours separes par une ou plusieurs lignes blanches. Il peut bien sur avoir plusieurs sous sections dans une meme section. En voici une deuxieme.

1.2 Au sujet des indexes

Au besoin vous pouvez aussi ajouter des mots en index en lui accolant devant un symbole diese. Dans la phrase precedente les mots index et symbole seront donc mis en index lors du processus de compilation.

2 Au sujet de la table des matieres et des numeros de section

La numerotation des sections et sous sections se fera automatiquement lors du processus de compilation. On utilisera une numerotation du style 1.3. qui signifiera sous section 3 de la section 1. Actuellement vous etes en train de lire la section 2.

Une table des matieres sera aussi generee lors de la compilation et inseree en tout debut de page HTML.

Index

Et le code source associé :

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//ENhttp://www.w3.org/TR/html4/strict.dtd">
<html><head><title>Titre page</title></head><body><h3>Table des matieres</h3>
<ul>
<li><a href=#h1>1. Ma premiere section </a></li>
<li>      <a href=#h1.1>1.1. Ma premiere sous Section </a>
</li>
<li>      <a href=#h1.2>1.2. Au sujet des indexes </a></li>
<li><a href=#h2>2. Au sujet de la table des matieres et des numeros de section </a></li>

</ul>
<hr />
<h1 align=center>Exemple de fichier texte </h1>
<h3 align=center>Nicolas </h3>
<p>
Ce fichier vous montre a quoi ressemble un fichier texte lu en entree. Pour eviter les
problemes de format, les accents ne sont pas autorises. Un tel fichier commence toujours
par le mot cle Titre, precede d'un chevron, et suivi du titre du texte; puis du mot cle
Auteur precede d'un chevron suivi du nom de l'auteur. Vient ensuite un ou plusieurs
paragraphes optionnels comme celui que vous lisez. </p>
<p>
Les paragraphes sont separes par une ou plusieurs lignes blanches. Vous etes donc entrain
de lire un deuxieme paragraphe. Les sections sont introduites par le symbole plus suivi du
titre, comme ci-dessous. </p>
<h1 id="h1">
1 Ma premiere section </h1>
<p>
Ici vous pouvez ecrire vos paragraphes de section. Au besoin vous pouvez definir des sous
sections, en les introduisant par deux symboles plus suivi du titre de sous section, comme
ci-dessous. </p>
<h2 id="h1.1">
1.1 Ma premiere sous Section </h2>
<p>
Ici vous pouvez ecrire vos paragraphes de section toujours separes par une ou plusieurs
lignes blanches. Il peut bien sur avoir plusieurs sous sections dans une meme section. En
voici une deuxieme. </p>
<h2 id="h1.2">
1.2 Au sujet des indexes </h2>
<p>
Au besoin vous pouvez aussi ajouter des mots en <span id=index>index</span> en lui
accolant devant un <span id=symbole>symbole</span> diese. Dans la phrase precedente les
mots index et symbole seront donc mis en index lors du processus de compilation. </p>
<h1 id="h2">
2 Au sujet de la table des matieres et des numeros de section </h1>
<p>
La numerotation des sections et sous sections se fera automatiquement lors du processus de
compilation. On utilisera une numerotation du style 1.3. qui signifiera sous section 3 de
la section 1. Actuellement vous etes en train de lire la section 2. </p>
<p>
Une table des matieres sera aussi generee lors de la compilation et inseree en tout debut
de page HTML. </p>

<hr />
<h1>Index</h1>
<a href=#symbole>symbole<a></br><a href=#index>index<a></br></body></html>
```

Travail préliminaire

Le programme suivant copie un fichier source dans un fichier target. Recopiez-le, comprenez-le et testez-le. Il servira de point de départ.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

FILE *source, *target = NULL;
char c;

int main(int argc, char const *argv[]) {
        // Ouvre le fichier test.txt en lecture seulement
        // (le fichier doit exister) :
        source = fopen("test.txt", "r");
        // Cree et ouvre un fichier tmp.html en lecture/ecriture,
        // avec suppression du contenu au prealable :
        target = fopen("target.html", "w+");

        if (source == NULL) {
                printf("Impossible d'ouvrir le fichier source\n");
                return -1;
        }

        if (target == NULL) {
                printf("Impossible d'ouvrir le fichier target\n");
                return -1;
        }

        c = fgetc(source); // lecture du caractere suivant du fichier source
        while(c!=EOF) {         // tant que la fin du fichier n'est pas atteinte
                fputc(c,target);          // ecrire c dans le fichier target
                c = fgetc(source);  // lecture du caractere suivant
        }

        if (source != NULL) fclose(source);   // fermeture du fichier source
        if (target != NULL) fclose(target);     // fermeture du fichier target
        return 0;
}
```

D'autres modes d'ouverture d'un fichier sont possibles. Par exemple :

- "w" : ouverture en écriture seul. Le fichier est créé s'il n'existe pas.
- "r+" : lecture et écriture. Le fichier doit exister.
- "a" : mode d'ajout, i.e. écriture dans le fichier à partir de la fin. Le fichier est créé s'il n'existe pas.

Les deux méthodes

fprintf(**fichier**, chaineDeFormat, [var1, var2, …])

fscanf(**fichier**, chaineDeFormat, [&var1, &var2, …])

permettent aussi de lire et écrire dans un fichier **fichier**. Leurs utilisations sont semblables à celles de printf() et scanf(). Seul le nom du fichier où écrire est ajouté en 1er argument.

Analyseur lexical

Description formelle des lexèmes (tokens)

Les lexèmes à reconnaître sont MOTCLE_T (mot-clé pour le titre), MOTCLE_A (mot-clé pour l'auteur), SECTION, SSECTION (sous section), NOUV_PARA (nouveau paragraphe), MOT et FIN décrits par les expressions rationnelles suivantes :

Caractere = [a-z] + [A-Z] + [0-9] + '.' + '?' + '!' + ',' + ';' + ':' + '-'

MOTCLE_T = ('\t' + ' ' + '\n')* . > . "Titre". ('\t' + ' ' + '\n' + EOF)

MOTCLE_A = ('\t' + ' ' + '\n')* . > . "Auteur" . ('\t' + ' ' + '\n' + EOF)

SECTION = ('\t' + ' ' + '\n')* .'=' . ('\t' + ' ' + '\n' + EOF)

SSECTION = ('\t' + ' ' + '\n')* .'='.'=' . ('\t' + ' ' + '\n' + EOF)

NOUV_PARA = ('\t' + ' ' + '\n')* .'\n' . ('\t' + ' ' + '\n')* .
\n . ('\t' + ' ' + '\n')*. Caractere

MOT = ('\t' + ' ' + '\n')* . Caractere^{+} . (\t' + ' ' + '\n' + EOF)

FIN = ('\t' + ' ' + '\n')* . EOF

On peut voir qu'un lexème est précédé d'un nombre quelconque d'espaces, de tabulations ou de sauts de ligne, et est suivi ou d'un espace, ou d'une tabulation ou d'un saut de ligne. Cela explique que toutes les expressions rationnelles précédentes commencent par ('\t' + ' ' + '\n')* et terminent par ('\t' + ' ' + '\n') (sauf pour FIN, mais c'est normal car EOF représente le dernier caractère du fichier source).

Travail à effectuer

1. Pour chaque expression rationnelle, déterminez un automate équivalent. Remarquez que tous ces automates ont le même état initial.
2. Fusionnez les états initiaux pour obtenir un nouvel automate équivalent à l'union des expressions rationnelles. **Veillez à garder tous les états finaux. Ils vous permettront de déterminer le token trouvé dans la suite du TP.**
3. Donnez l'automate **déterministe** correspondant à l'automate de la question 2.
4. Il est intéressant de noter que le mot '\n'.'\n'.toto par exemple est reconnu de deux façons : en une seul fois, par l'expression rationnelle **MOT**; ou en deux fois, par l'expression rationnelle **NOUV_PARA** suivie de **MOT**. Le bon choix est évidemment le second. Mais cet exemple illustre le fait que les expressions rationnelles doivent être priorisées. Voici comment : **FIN** est prioritaire sur **NOUV_PARA** qui est prioritaire sur le reste. Les expressions rationnelles autres que **FIN** et **NOUV_PARA** peuvent être de même priorité car elles ne sont pas en conflit. Noter qu'un tel mécanisme de priorité existe aussi dans les outils de compilation tel que Lex/Flex.
   Modifier votre automate pour tenir compte de ces priorités.
5. Ecrivez une fonction scanner() qui cherche le **prochain lexème** d'un fichier texte. La fonction renvoie 1 si un lexème est trouvé, 0 sinon (ce qui correspond à une erreur). Elle sera écrite à l'aide d'instructions **goto**. Chaque étiquette doit correspondre à un état de votre automate. Ainsi le code du scanner "collera" à la

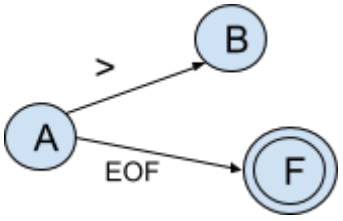
conception. Lorsqu'un nouveau lexème est trouvé, la fonction enregistre son type et sa valeur dans deux **variables globales** token et tokenValue (ces variables seront donc disponibles partout dans le programme). La variable token peut valoir **MOTCLE_T**, **MOTCLE_A**, **SECTION**, **SSECTION**, **NOUV_PARA**, **MOT** ou **FIN**. La variable tokenValue sera utilisée pour enregistrer le mot correspondant au token **MOT**. La lecture d'un caractère dans le fichier en entrée se fera par l'instruction

c = fgetc(source)

où la variable c est aussi globale. Nous avons donc besoin des structures de données suivantes (ces variables sont globales) :

```
char c;
enum TokenType {MOTCLE_T, MOTCLE_A, SECTION, SSECTION,
                NOUV_PARA, MOT, FIN} token;
char tokenValue[50];
```

Si une partie de votre automate ressemble à



alors le morceau de code correspondant dans la fonction scanner() sera :

```
scanner() {
  ...
  A : // cas d'un état non final
  if (c == '>') {c = fgetc(source); goto B; }
                  // fgetc() avant le goto car B n'est pas final

  If (c == EOF) goto F;
                  // pas de fgetc() car F est final
  goto Error;

  F : // cas d'un état final
  token = FIN;
  return 1;

  Erreur : // Etat  supplémentaire pour gérer les erreurs
  printf("aucun token trouve\n");
  exit(-1); // le programme s'arrête dès qu'une erreur est détectée.
} \\ end scanner()
```

Notez dans cet exemple que nous lisons le caractère suivant avant un goto X uniquement si l'état X n'est pas final.

6. Ecrivez un programme test qui
   - ouvre un fichier texte,
   - lit un premier caractère puis,
   - tant que le lexème FIN n'est pas trouvé, appelle la fonction scanner() et affiche le type et la valeur du token trouvé dans le terminal.
   - lorsque le token FIN est trouvé, ferme le fichier texte.

   Essayez de bien gérer les erreurs, i.e. quand scanner() renvoie 0.

Analyseur syntaxique

Description formelle de la structure du fichier texte

Elle est donnée par la grammaire suivante où :

- AXIOME est le l'axiome de la grammaire
- les symboles terminaux sont les lexèmes :

MOTCLE_T, MOTCLE_A, **SECTION**, **SSECTION**,
NOUV_PARA, **MOT** et **FIN**.

- les règles de production sont

AXIOME	→	HEAD BODY **FIN**
HEAD	→	**MOTCLE_T** TEXT **MOTCLE_A** TEXT
TEXT	→	**MOT** TEXT | epsilon
BODY	→	P S1
P	→	**NOUV_PARA** TEXT P | epsilon
S1	→	H1 P S2 S1 | epsilon
H1	→	**SECTION** TEXT
S2	→	H2 P S2 | epsilon
H2	→	**SSECTION** TEXT

Travail à effectuer

1. Calculez les first et les follow de cette grammaire.
2. Vérifiez que cette grammaire est LL(1).
3. Implémentez l'analyseur syntaxique à l'aide de fonctions mutuellement récursives. Les différents tokens seront obtenus par un appel à la fonction scanner(). Par exemple, à la règle

TEXT → **MOT** TEXT | epsilon

correspondra à la fonction Analyse_TEXT() suivante (la liste des règles effectuées n'apparait pas dans ce code…) :

```
Analyse_TEXT() {

  if ( token == MOT ) {
       scanner();      // trouve le prochain token dans
       Analyse_TEXT();
  }
  else if ( token ∈ folow(TEXT) )
            // alors il n'y a rien à faire.
       else {
            printf("erreur dans l'analyse de TEXT\n");
            exit(-1);  // on s'arrête dès qu'une erreur est trouvée.
            }
}
```

4. Ecrivez un programme test qui lit un fichier texte en entrée et qui vérifie sa syntaxe.
5. Donnez la table prédictive.

Génération de code HTML

Nous allons maintenant produire le fichier HTML à partir du fichier texte donnée en entrée. L'idée principale est de générer le code HTML durant la phase d'analyse syntaxique.

Travail à effectuer

1. Dans le **programme principal**, avant le lancement de l'analyseur syntaxique, ajoutez le code nécessaire pour créer un fichier target.html et écrire dans ce fichier le texte de l'exemple (code

source html) allant du DOCTYPE jusqu'à la balise ouvrante <body> incluse. Après l'analyse syntaxique, ajoutez le code nécessaire pour écrire dans le fichier target.html les balises fermantes </body></html>. N'oubliez pas de refermer votre fichier. Testez votre programme et vérifiez dans un navigateur que le fichier target.html est bien créé et contient ce qu'il faut.

2. Dans la **fonction Analyse_TEXT()** correspondant à l'analyse du non-terminal TEXT de la grammaire, ajoutez les instructions permettant de recopier le texte source dans un fichier target.html. Plus précisément, il s'agit ici d'écrire dans le fichier target.html la variable globale tokenValue **au bon moment**.Testez votre programme. Vérifiez dans un navigateur que le contenu de votre fichier texte source est bien présent. Notez que pour l'instant aucun formatage HTML n'est présent.

3. Dans la **fonction Analyse_H1()** correspondant à l'analyse du non-terminal H1 de la grammaire, juste avant l'instruction Analyse_TEXT(), ajoutez le code nécessaire pour écrire dans le fichier target.html une balise <h1>. De même, juste après l'instruction Analyse_TEXT(), ajoutez le code nécessaire pour écrire dans le fichier target.html une balise </h1>. Testez votre programme. Normalement, si vous ouvrez votre fichier target.html dans un navigateur, les titres de section seront formatés.

4. En procédant de manière similaire, ajoutez dans les bonnes fonctions le code nécessaire pour reproduire l'exemple donné au début du TP. Vous aurez donc à formater le titre, les auteurs, les sous sections et les paragraphes.

Pour aller plus loin

Numéro de sections automatiques

Nous allons ici ajouter la numérotation automatique des sections et sous sections. Il suffit pour cela de maintenir deux variables entières num_section et num_ssousection au cours de l'analyse syntaxique et de les ajouter dans le fichier target.html au moment de l'écriture des titres de section et sous section.

...

Génération de la table des matières

...

Génération de la table d'index

...