h e p i a
Haute école du paysage, d'ingénierie et d'architecture de Genève

hepia
Ingénierie des Technologies de l'information
Technique de compilation - TCP

Labo 3 : Un analyseur SGML avancé

MINELLI Michaël

```
##################################################################
#          Labo3 : JFlex et CUP -- Analyseur SGML                #
##################################################################
# Auteur : Michaël MINELLI                                       #
# Classe : ITI Jour 2ème                                         #
#                                                                #
# Fichier testé : test.txt                                       #
##################################################################
#                          Résultat                              #
##################################################################
<A>
        tata1
        <C/>
        tata2
        <D>
                showTextTEST
                tata3
        </D>
        tata4
        ID12
</A>
##################################################################
```

Table des matières

1 Historique : labo2

1.1 Réalisation

La réalisation s'est déroulée en deux grandes phases. Il s'agit tout simplement des phases JFlex et cup.

1.1.1 Phase JFlex

Lors de la phase JFlex, il a été question de déterminer comment fractionner la détection afin de n'omettre aucune détection. La première chose qui fut visible, est que les balises de fermeture seraient toujours de la même forme ("</X>") sans possibilité d'erreur selon ce qui a été dit lors de la présentation du labo.

Après, il a fallu analyser les balises ouvrantes et auto-fermantes, car elles doivent pouvoir contenir des identificateurs tout en faisant attention qu'elles ne comportent pas d'erreur (guillemet non fermé ou simplement texte ne devant pas être là). Ces balisent ont donc été fractionnées en trois morceaux. Le premier est l'ouverture de balise ("<X") et le dernier est la fermeture (">" ou "/>"). La deuxième partie, quant à elle, est celle pouvant contenir les identificateurs ou les erreurs. La détection d'un identificateur ('a="texte"') a donc été ajoutée.
A ce moment, le dernier problème restant était donc la détection d'erreur. Celle-ci s'est fait part l'intermédiaire de la détection de tous les caractères qui n'ont pas été détecté par une autre règle. Cela voulait dire deux choses. La première est que ce n'était pas forcément une erreur, mais cela pouvait aussi s'avérer être le texte entre les balises il faudrait donc déterminer par la suite (dans le cup) si cela est vraiment une erreur. La deuxième qu'il fallait ignorer la détection des caractères non pris en erreur (espace, retour chariot, etc.) ainsi que les identificateurs.

La partie JFlex a été testée tout d'abord sans le cup avec de simples affichages println.

1.1.2 Phase Java Cup

Une première solution fut envisagée puis écartée, car ne correspondaient pas au cahier des charges. Cette solution mettait en place des variables bouléaines et une pile afin de tester les fermetures de balises. Celle-ci était tout à fait réalisable, mais ne nécessitait pas de cup simplement du JFlex au contraire du but de ce labo.

Dans la solution retenue, il a fallu premièrement concaténer les erreurs potentielles, car dans le JFlex tout caractère était pris comme tel donc il était possible d'avoir plusieurs (sans savoir exactement combien à l'avance) erreurs potentielles à la suite.
Tout était presque mis en place, il ne restait plus qu'à détecter les balises ouvrantes et autofermantes en entier, car elles ont été fractionnées par le JFlex. Pour le besoin de la détection d'erreur pour chacune des balises, deux variables non-terminales ont été créées. La première, si aucune erreur n'est présente et que l'arrêt de l'affichage n'est pas à vrai, la balise sera affichée et pour les balises ouvrantes la valeur de celle-ci mise en mémoire. La deuxième, s'il y a une erreur, cela affiche la lettre avec la mention "(erreur)" et met la variable indiquant la fin de l'affichage à vrai.
Enfin, la grammaire rassemble le tout en vérifiant que la lettre des balises ouvrantes correspond à la lettre des balises fermantes.

2 Réalisation du Labo 3

2.1 Description de la réalisation

Tout d'abord, ce labo a été construit sur les bases du labo précédent. La première étape a donc été de copier ce labo et de l'adapter afin qu'il ne rende non pas des "+A", "-A", etc. mais bien les balises avec une indentation correct.
Par la suite, les mêmes deux étapes que pour le labo 2 ont été suivies. Donc, en tant que deuxième étape de ce labo, il a fallu aller dans le JFlex afin d'y ajouter la détection des balises de définition et celles de tests.
Enfin, en tant que troisième et dernière étape (hormis les tests bien entendu), il a fallu éditer le fichier cup. Le labo 2 ayant été fait de façon à permettre l'évolutivité, il a été simple de rajouter ce qui manquait. Il a fallu rajouter les symboles ainsi que mettre ceux-ci en forme dans le symbole non terminal "sgml". Il a été décidé de faire la gestion des variables par l'intermédiaire d'un HashMap qui aurait comme clé le nom de la variable avec comme valeur la donnée de celle-ci.

2.2 Fichier JFlex

```
/* Michaël Minelli
 * Technique de compilation
 * ITI Jour 2ème
 * Labo 3 : Analyseur SGML
 */
import java_cup.runtime.*;
import java.util.Vector;
%%
%class Labo3
%line
%column
%cup
%{
%}

LETTRE=[A-Z]
LETTRE_MIN=[a-z]
ALPHABET=({LETTRE_MIN}|{LETTRE})

FERMANTE=\<\/{LETTRE}\>
DEBUT=\<{LETTRE}
ID={LETTRE_MIN}\=\".*\"
FIN_AUTOFERMANTE=\/\>
FIN=\>
IGNORE=(\ |\t|\n|{ID})

VAR=\<#set\ {ALPHABET}+\=\".*\"\/\>
IF=\<#if\ {ALPHABET}+\=\=\".*\"\>
FI=\<\#if\/\>

%%

{FERMANTE} {return new Symbol(sym.FERMANTE, String.valueOf(yytext().toCharArray()[2]));}
{DEBUT}   {return new Symbol(sym.DEBUT, String.valueOf(yytext().toCharArray()[1]));}
{FIN_AUTOFERMANTE} {return new Symbol(sym.FIN_AUTOFERMANTE);}
{FIN} {return new Symbol(sym.FIN);}

{VAR} {return new Symbol(sym.VAR, yytext());}
{IF} {return new Symbol(sym.IF, yytext());}
{FI} {return new Symbol(sym.FI);}

{IGNORE} {;}
. {return new Symbol(sym.TEXT_OR_ERROR, yytext());}
```

Listing 1 – `labo3.flex`

2.3 Fichier cup

```
/* Michaël Minelli
* Technique de compilation
* ITI Jour 2ème
* Labo 3 : Analyseur SGML
*/
import java_cup.runtime.*;
import java.util.*;

action code {:
    //Stop l'affichage si une erreur a été trouvée
   public Boolean stopExec = false;

    public Integer countTab = 0;

    //Stockage des variables
    public HashMap<String, String> vars = new HashMap<String, String>();

    //Affiche le nombre de tabulation voulue
    public void showTab() {
        for (int i = 0; i < countTab; i++)
            System.out.printf("\t");
    }

    //Test que le balise se soit fermée avec le meme type de balise
   public void testBaliseFermante(String l, String m) {
      if (!stopExec) {
            countTab--; //La fermeture d'une balise signifie que nous decendons d'un étage dans les tabs
            showTab();
         System.out.printf("</" + m + ">");
         if (!l.equals(m)) {
            System.out.printf("(erreur) ");
            stopExec = true;
         }
            System.out.printf("\n");
      }
   }
:};

terminal String DEBUT, FERMANTE, VAR, IF;
terminal FIN_AUTOFERMANTE, FIN, TEXT_OR_ERROR, FI;

non terminal String baliseAutoFermante, baliseAutoFermanteWithError, balise, baliseWithError;
non terminal textOrError, sgml, sgml_withoutError;

//Tout le code
sgml ::= balise:l sgml FERMANTE:m {: testBaliseFermante(l, m); :} sgml
            | VAR:n {: vars.put(n.substring(6, n.length()-2).split("=")[0], n.substring(6, n.length()-2).split("\"")[1]); :} sgml
            | IF:n textOrError:t FI {:  String valVar = vars.get(n.substring(5, n.length()-2).split("=")[0]); //Valeur de la variable, null si n'éxiste pas
                                        if (valVar != null && valVar.equals(n.substring(5, n.length()-2).split("\"")[1])) {
                                            showTab();
                                            System.out.printf(t + "\n"); //Si c'est correct, affiche le texte entre les if
                                        }
                                    :} sgml
        | baliseWithError sgml FERMANTE sgml
        | baliseAutoFermante sgml
        | baliseAutoFermanteWithError sgml
            | textOrError:t {: showTab(); System.out.printf(t + "\n"); :} sgml_withoutError
        | ;

//Permet de contourné une érreur avec les textOrError qui fesait de la grammaire une grammaire ambigue
sgml_withoutError ::= balise:l sgml FERMANTE:m {: testBaliseFermante(l, m); :} sgml
            | VAR:n {: vars.put(n.substring(6, n.length()-2).split("=")[0], n.substring(6, n.length()-2).split("\"")[1]); :} sgml
            | IF:n textOrError:t FI {:  String valVar = vars.get(n.substring(5, n.length()-2).split("=")[0]);
                                        if (valVar != null && valVar.equals(n.substring(5, n.length()-2).split("\"")[1])) {
                                            showTab();
                                            System.out.printf(t + "\n");
                                        }
                                    :} sgml
            | baliseWithError sgml FERMANTE sgml
        | baliseAutoFermante sgml
        | baliseAutoFermanteWithError sgml
        | ;

//Balises auto-fermante
baliseAutoFermante ::= DEBUT:n FIN_AUTOFERMANTE {: if (!stopExec) { showTab(); System.out.printf("<" + n + "/>\n"); RESULT = n; } :};
baliseAutoFermanteWithError ::= DEBUT:n textOrError FIN_AUTOFERMANTE {: if (!stopExec) { showTab(); System.out.printf("<" + n + "/> (erreur)"); stopExec = true; } :};

//Balises ouvrantes
balise ::= DEBUT:n FIN {: if (!stopExec) { showTab(); System.out.printf("<" + n + ">\n"); countTab++; } RESULT = n; :};
baliseWithError ::= DEBUT:n textOrError FIN {: if (!stopExec) { showTab(); System.out.printf("<" + n + "> (erreur)"); stopExec = true; } :};

//Fais en sort de réduite plusieurs TEXT_OR_ERROR à  la suite par un seul textOrError
textOrError ::= textOrError:o TEXT_OR_ERROR:n {: RESULT = new String((String)o + (String)n); :} | TEXT_OR_ERROR:n {: RESULT = n; :};
```

Listing 2 – `labo3.cup`

3 Compilation

Le makefile a été conçu de sorte qu'il n'y ai rien d'autre à faire qu'un "make". Voici le contenu du fichier "LISEZMOI" :

1. Ouvrez la console dans le dossier contenant les fichiers du labo
2. Effectuez un « make » avec l'option non obligatoire « TESTFILE=... » (ou les ... est le nom du fichier à analyser). Si cette option n'est pas indiquée cela prendra comme nom de fichier par défaut : test.txt

Exemples :
a. make
b. make TESTFILE=test_faux.txt

3.1 Code du makefile

```
1  JAVA=java
2  JAVAC=javac
3  JFLEX=jflex
4  JAVACUP=java-cup-11a.jar
5  CLASSPATH=$(JAVACUP):.
6
7  FILE_FLEX=Labo2.flex
8  FILE_CUP=Labo2.cup
9  FILE_JAVA_NAME=Labo2
10 FILE_TEST_PRG_NAME=test
11
12 TEST_CLASS=test
13
14 FILE=test.txt
15 ifdef TESTFILE
16    FILE=$(TESTFILE)
17 endif
18
19 all : $(FILE) sym.class parser.class $(FILE_JAVA_NAME).class $(FILE_TEST_PRG_NAME).class
20    $(JAVA) -classpath $(CLASSPATH) $(TEST_CLASS) $(FILE)
21
22 $(FILE_JAVA_NAME).java : $(FILE_FLEX)
23    $(JFLEX) $(FILE_FLEX)
24
25 sym.java parser.java : $(FILE_CUP)
26    $(JAVA) -jar $(JAVACUP) $(FILE_CUP)
27
28 %.class : %.java
29    $(JAVAC) -classpath $(CLASSPATH) $<
30
31 clean :
32    rm -rf *class *~ parser.java sym.java $(FILE_JAVA_NAME).java
```

Listing 3 – `makefile`