

TP de Compilation: JFlex & CUP

Franck Cassez

CNRS/IRCCyN

BP 92101

1 rue de la Noë

44321 Nantes Cedex 3

France

Février 2006

ENSAI

Campus de Ker Lann, Bruz

Contenu

- ▶ Analyse lexicale avec JFlex
 - Installer et tester JFlex
 - Exemples de programmes jflex

- ▶ Analyse syntaxique avec CUP
 - Installer et tester CUP
 - Exemple de programme CUP
 - Utiliser CUP avec JFlex

- ▶ Compilation: traduction dirigée par la syntaxe
 - Calcul d'un attribut synthétisé
 - Actions sémantiques
 - Mini calculateur

- ▶ Bibliographie et liens

- ▶ Analyse lexicale avec JFlex
 - Installer et tester JFlex
 - Exemples de programmes jflex

- ▶ Analyse syntaxique avec CUP
 - Installer et tester CUP
 - Exemple de programme CUP
 - Utiliser CUP avec JFlex

- ▶ Compilation: traduction dirigée par la syntaxe
 - Calcul d'un attribut synthétisé
 - Actions sémantiques
 - Mini calculateur

- ▶ Bibliographie et liens

Installer JFlex pour un système Unix

① télécharger `jflex-1.x.x.tar.gz` à l'adresse <http://jflex.de/>

② dans un répertoire *dir*:

```
$ tar zxvf jflex-1.x.x.tar.gz
```

③ dans votre répertoire *bin* ajouter un lien `jflex`:

```
$ cd bin
```

```
$ ln -s dir/jflex-1.x.x/bin/jflex jflex
```

④ ajouter *bin* aux chemins de `PATH`

▶ bash

▶ csh

⑤ essai:

```
$ cd jflex-1.x.x/examples/standalone
```

```
$ jflex standalone.flex
```

```
$ javac Subst.java
```

```
$ java Subst sample.inp
```

Transformation et filtrage de texte

(calcul-v1.flex)

```

1  /* calcul-v1.flex */
2  %%
3  /* jflex options */
4  %class Lexi
5  %unicode
6  %line
7  %column
8  %standalone
9
10 /* models */
11 integer=[0-9]+
12 %%
13 /* rules */
14 {integer}      {System.out.print ("Number("+yytext()+")" ) ; }
15 \n            { ; }
16 .             { ; }

```

► Makefile

Transformation et filtrage de texte

(calcul-v2.flex)

```

1  /* calcul-v2.flex */
2  %%
3  /* options */
4  %class Lexi
5  %unicode
6  %line
7  %column
8  %standalone
9  %{eof
10 System.out.println ("\nEnd_of_the_program");
11 %eof}
12 /* models */
13 integer=[0-9]+
14 %%
15 /* rules */
16 {integer}      {System.out.print ("Number("+yytext()+")" ); }
17 \n            { ; }
18 .            { ; }

```

Transformation et filtrage de texte (calcul-v3.flex)

```

1  /* calcul-v3.flex */
2  %%
3  %class Lexi
4  %unicode
5  %line
6  %column
7  %standalone
8  %{
9  public int n,sum=0;
10 %}
11 %{eof
12 if (n>0)
13 System.out.println ("\n└─Mean└─value└─
    of└─the└─sum└─of└─the└─integers└─is└─"
    +((sum+0.0)/n));
14 %eof}

15 /* models */
16 integer=[0-9]+
17 %%
18 /* rules */
19 {integer} {
20     System.out.print ("Number(
        "+yytext()+")" );
21     n++;
22     sum+= Integer.valueOf(
        yytext()).intValue() ;
23     }
24 \n    { ; }
25 .    { ; }

```

Analyse Lexicale (calcul-v4.flex)

```

1  /* calcul-v4.flex */
2  %%
3  %class Lexi
4  %unicode
5  %line
6  %column
7  %standalone
8  %{eof
9  System.out. println ("\nFini");
10 %eof}
11 /* models */
12 integer=[0-9]+
13 ope=[\+|-|\*\|/]
14 %%
15 /* rules */
16 {integer}      {System.out. print ("Int("+yytext()+")") ;}
17 {ope}         {System.out. print ("Ope("+yytext()+")") ;}
18 \n           { ; }
19 .           { ; }

```

Évaluation d'expressions en notation post fixée

- ▶ $n \in \mathbb{N}$ est une **expression post fixée**
- ▶ si s_1 et s_2 sont des **expressions post fixées**, $s_1 s_2 \bowtie$ avec $\bowtie \in \{+, -, *, /\}$ est une **expression post fixée**.

Exercice (Évaluation d'expressions post fixées)

Écrire un programme jflex permettant d'évaluer des expressions post fixées.

Exemple

```
<7 2 * 3 1 - 7 * +>
```

```
1: 7 2 * 3 1 - 7 * + = 28.0
```

```
<9 + 2 9 >
```

~>

```
2: 9 +'Error on "+" line 2 column 4' 2 9 = novalue
```

```
<11 9 - 5 6 / * 2 +>
```

```
3: 11 9 - 5 6 / * 2 + = 3.6666665
```

- ▶ Analyse lexicale avec JFlex
 - Installer et tester JFlex
 - Exemples de programmes jflex

- ▶ Analyse syntaxique avec CUP
 - Installer et tester CUP
 - Exemple de programme CUP
 - Utiliser CUP avec JFlex

- ▶ Compilation: traduction dirigée par la syntaxe
 - Calcul d'un attribut synthétisé
 - Actions sémantiques
 - Mini calculateur

- ▶ Bibliographie et liens

Installer CUP (système Unix)

- 1 télécharger `java_cup_vxxx.tar.gz` à l'adresse
`http://www2.cs.tum.edu/projects/cup/`
- 2 dans un répertoire *dir*:
`$ tar zxvf java_cup_vxxx.tar.gz`
- 3 ajouter le répertoire `dir/java_cup_vxxx/` au CLASSPATH (version bash)
`$ export CLASSPATH=$CLASSPATH:./dir/java_cup_vxxx/`
- 4 essai:
`$ cd java_cup_vxxx`
`$./INSTALL`

Installer CUP (système Unix) (2)

- créer un fichier `jcup` dans les répertoire `bin` avec:

```
#!/bin/bash

# path to the java interpreter
JAVA=java

CLASSPATH=$CLASSPATH:./dir/java_cup_vxxx/java_cup
export CLASSPATH

$JAVA java_cup.Main $@
```

Grammaire des expressions arithmétiques

```

1  /* calcul .cup */
2  import java_cup.runtime.*;
3
4  parser code {:
5  public void report_fatal_error(String
        message, Object info)
6      throws Exception {
7      report_error(message, info);
8      throw new Exception("Syntax_ Error");
9  }
10 :}
11 ;
12
13 terminal INT,PLUS,MOINS,
14     FOIS,DIV,PARENG,PAREND;
15 non terminal expr;
16 precedence left PLUS, MOINS;
17 precedence left FOIS, DIV;
18
19 expr ::= expr PLUS expr
20       |
21       expr MOINS expr
22       |
23       expr FOIS expr
24       |
25       expr DIV expr
26       |
27       PARENG expr PAREND
28       |
29       INT
30 ;

```

Communication CUP/JFlex

- ▶ l'analyseur **syntaxique** (obtenu avec CUP) reçoit les “tokens” de l'analyseur lexical
- ▶ l'analyseur **lexical** (obtenu avec JFlex) doit envoyer les bons “tokens”
- ▶ à partir de `calcul.cup` **deux classes** générées:
 - \$ `jcup calcul.cup`
 - ▶ `sym.java` [le fichier](#)
 - ▶ `parser.java`
- ▶ l'analyseur syntaxique attend des valeurs de type `Symbol` (`java_cup.runtime.Symbol`) [la classe Symbol](#)

Fichier JFlex pour CUP

```

1  /* calcul.flex */
2  import java_cup.runtime.*; // import Symbol class etc
3  %%
4  %class Lexi
5  %unicode
6  %line
7  %column
8  %cup
9  %{ /* a function to create tokens along with line,col. numbers */
10 private Symbol symbol(int type) {
11     return new Symbol(type, yylne, yycolumn);
12 }
13 %}
14 /* models */
15 integer=[0-9]+
16 %%
17 /* rules */
18 {integer}      {System.out.print(yytext()) ; return symbol(sym.INT) ; }
19 +             {System.out.print(yytext()) ; return symbol(sym.PLUS) ; }
20 -             {System.out.print(yytext()) ; return symbol(sym.MOINS) ; }
21 \*            {System.out.print(yytext()) ; return symbol(sym.FOIS) ; }
22 \/           {System.out.print(yytext()) ; return symbol(sym.DIV) ; }
23 \(           {System.out.print(yytext()) ; return symbol(sym.PARENG) ; }
24\)           {System.out.print(yytext()) ; return symbol(sym.PAREND) ; }
25 \n           {System.out.print(yytext()) ; }
26 .            {System.out.print(yytext()) ; }

```

L'analyseur syntaxique complet

► un Main utilisant le parser

► le fichier Main.java

► compilation:

► Makefile

```
$jcup calcul.cup
```

```
$jflex calcul.flex
```

```
$javac Lexi.java
```

```
$javac parser.java
```

```
$javac Main.java
```

► utilisation

```
$java Main exemple.txt
```

Exercice (Liste d'expressions)

Ajouter à la grammaire précédente des règles permettant de décrire des listes d'expressions arithmétiques séparées par des «;».

L'analyseur syntaxique complet

► un Main utilisant le parser

► le fichier Main.java

► compilation:

► Makefile

```
$jcup calcul.cup
```

```
$jflex calcul.flex
```

```
$javac Lexi.java
```

```
$javac parser.java
```

```
$javac Main.java
```

► utilisation

```
$java Main exemple.txt
```

Exercice (Liste d'expressions)

Ajouter à la grammaire précédente des règles permettant de décrire des listes d'expressions arithmétiques séparées par des «;».

- ▶ Analyse lexicale avec JFlex
 - Installer et tester JFlex
 - Exemples de programmes jflex

- ▶ Analyse syntaxique avec CUP
 - Installer et tester CUP
 - Exemple de programme CUP
 - Utiliser CUP avec JFlex

- ▶ **Compilation: traduction dirigée par la syntaxe**
 - Calcul d'un attribut synthétisé
 - Actions sémantiques
 - Mini calculateur

- ▶ Bibliographie et liens

Attribut synthétisé

- ▶ **but**: calculer la valeur de chaque expression
- ▶ calcul d'un **attribut synthétisé** *val* sur la grammaire
- ▶ définition de *val*:
 - ▶ règle $\text{expr} ::= \text{INT}$
 $\text{val}(\text{expr}) = \text{val}(\text{INT})$
 - ▶ règle $\text{expr} ::= \text{expr}(1) \text{ PLUS } \text{expr}(2)$
 $\text{val}(\text{expr}) = \text{val}(\text{expr}_1) + \text{val}(\text{expr}_2)$
- ▶ *val* est **synthétisé**: la valeur sur le membre gauche d'une règle est fonction des valeurs sur les membres de droite

Implémentation en JFlex et CUP

```

1  /* calcul.flex */
2  ...
3  %{
4  // a function to create tokens along with line,col. numbers
5  private Symbol symbol(int type) { ...
6  private Symbol symbol(int type, Object value) {
7      return new Symbol(type, yyline, yycolumn, value);
8  }
9  %} ...
10 %%
11 /* rules */
12 {integer} {System.out.print (yytext()); return symbol(sym.INT,new Integer(yytext()));}
13 ...

```

```

1  /* calcul.cup */ ...
2  terminal Integer INT;
3  terminal PLUS,MOINS,FOIS,DIV,PARENG,PAREND;
4  non terminal list_expr;
5  non terminal Integer expr;
6  ...
7  expr ::= expr:e1 PLUS expr:e2
8      {: RESULT = new Integer(e1.intValue() + e2.intValue()); :}
9  ...
10      INT:n
11      {: RESULT = new Integer(n.intValue()); :}

```

Insertion d'actions sémantiques

```

1  /* calcul.cup */ ...
2  terminal Integer INT;
3  terminal PLUS,MOINS,FOIS,DIV,PARENG,PAREND;
4  non terminal list_expr;
5  non terminal Integer expr;
6  ...
7  expr ::= expr:e1 PLUS {: Action :} expr:e2
8         {: RESULT = new Integer(e1.intValue() + e2.intValue()); :}
9  ...
10 ...
11      INT:n
12      {: RESULT = new Integer(n.intValue()); :}

```

Exercice (Résultats des expressions)

Ajouter à la grammaire précédente des actions affichant le résultat de chaque expression.

Exemple

 $2 + 3 * 5 ;$
 $34 - (5 - 1)$

devient

 $2 + 3 * 5 = 17$
 $34 - (5 - 1) = 30$

Formattage du résultat

Exercice (Filtrage des résultats)

Modifier les programmes précédents pour obtenir une présentation où les caractères superflus sont supprimés.

Exemple

2 + 4 * (4

\$ % nb,nb,w + 8)

donne

2+4*(4+8) = 50

;

3*(2+6/(2+1)) = 12

3 * (2 + 6 / (2 + 1))

Exercice (Compilation en expression post fixée)

Remplacer l'évaluation par la traduction en expressions post fixées.

Calculette

Exercice (Equations avec variables)

Ecrire les programmes traitant les expressions avec variables suivant les spécifications suivantes:

- 1 en supposant que les variables ont été déclarées avant d'être utilisées,
- 2 en imposant qu'une variable soit déclarée avant son utilisation,
- 3 en permettant la réutilisation des variables.

Exemple

```
a=2; b=5;
```

```
2+a+2 ; 5*b + 18; (b+a)*a; a=2; 12*a
```

- ▶ Analyse lexicale avec JFlex
 - Installer et tester JFlex
 - Exemples de programmes jflex

- ▶ Analyse syntaxique avec CUP
 - Installer et tester CUP
 - Exemple de programme CUP
 - Utiliser CUP avec JFlex

- ▶ Compilation: traduction dirigée par la syntaxe
 - Calcul d'un attribut synthétisé
 - Actions sémantiques
 - Mini calculateur

- ▶ **Bibliographie et liens**

Bibliographie et URLs utiles

- [Aho et al.]** A.Aho, R.Sethi et J.Ullmann
Compilateurs, principes, techniques et outils,
Addison Wesley, 1991
- [Linux Gaz.]** Un article sur l'utilisation de JFlex et CUP.
<http://www.linuxgazette.com/issue41/lopes/lopes.html>
- [CUP]** Site de CUP
<http://www2.cs.tum.edu/projects/cup/>
- [JFLEX]** Site de JFlex
<http://jflex.de/>
- [Java]** Java Sun
<http://java.sun.com/j2se/1.4.2/docs/api/>

Variable d'environnement PATH

- ▶ pour bash:
dans le fichier `.bashrc` ajouter:

```
# PATH settings  
PATH=$PATH:~/bin
```

```
export PATH
```

- ▶ pour csh:
dans le fichier `.cshrc` ajouter:

```
# PATH settings
```

```
setenv PATH "$PATH":~/bin
```

Makefile

- But = *automatiser* le processus de compilation

```
# commentaire: un Makefile pour le standalone jlex
SRC=calcul-v1.flex
JSRC=Lexi

all: $(JSRC).class

$(JSRC).class : $(JSRC).java
    javac $<

$(JSRC).java : $(SRC)
    jflex $(SRC)

clean:
    rm -rf *.class *~ *.java
```

- Usage:

```
$ make
$ make clean
$ make SRC=calcul-v2.flex
```

La classe sym.java

```
//-----  
// The following code was generated by CUP v0.10k  
// Mon Nov 29 15:18:09 CET 2004  
//-----  
  
/** CUP generated class containing symbol constants. */  
public class sym {  
    /* terminals */  
    public static final int INT = 2;  
    public static final int PARENG = 7;  
    public static final int PAREND = 8;  
    public static final int EOF = 0;  
    public static final int PLUS = 3;  
    public static final int DIV = 6;  
    public static final int error = 1;  
    public static final int MOINS = 4;  
    public static final int FOIS = 5;  
}
```

La classe Symbol.java

```

package java_cup.runtime;

/**
 * Defines the Symbol class, which is used to represent all terminals
 * and nonterminals while parsing. The lexer should pass CUP Symbols
 * and CUP returns a Symbol.
 *
 * @version last updated: 7/3/96
 * @author Frank Flannery
 */

/* *****
 * Class Symbol
 * what the parser expects to receive from the lexer.
 * the token is identified as follows:
 * sym: the symbol type
 * parse_state: the parse state.
 * value: is the lexical value of type Object
 * left : is the left position in the original input file
 * right: is the right position in the original input file
 * *****/

```

La classe Symbol.java (cont.)

```
public class Symbol {

    /**
     * Constructor for l,r values
     */
    public Symbol(int id, int l, int r, Object o) {
        this(id);
        left = l;
        right = r;
        value = o;
    }

    /**
     * Constructor for no l,r values
     */
    public Symbol(int id, Object o) {
        this(id, -1, -1, o);
    }

    /**
     * Constructor for no value
     */
    public Symbol(int id, int l, int r) {
        this(id, l, r, null);
    }
}
```

La classe Symbol.java (cont.)

```

/*****
  Constructor for no value or l,r
*****/

public Symbol(int sym_num) {
    this(sym_num, -1);
    left = -1;
    right = -1;
    value = null;
}

/*****
  Constructor to give a start state
*****/
Symbol(int sym_num, int state)
{
    sym = sym_num;
    parse_state = state;
}

/** The symbol number of the terminal or non terminal being represented */
public int sym;

/** The parse state to be recorded on the parse stack with this symbol.
 * This field is for the convenience of the parser and shouldn't be
 * modified except by the parser.
 */
public int parse_state;
/** This allows us to catch some errors caused by scanners recycling

```

La classe Symbol.java (cont.)

```
    * symbols. For the use of the parser only. [CSA, 23-Jul-1999] */
    boolean used_by_parser = false;

/*****
The data passed to parser
*****/

    public int left, right;
    public Object value;

/*****
Printing this token out. (Override for pretty-print).
*****/
    public String toString() { return "#" + sym; }
}
```

La classe Main.java

```
import java.io.*;

public class Main {
    static public void main(String argv[]) {
        /* Start the parser */
        try {
            parser p = new parser(new Lexi(new FileReader(argv[0])));
            Object result = p.parse().value;
            System.out.println("\nfile OK");
        } catch (Exception e) {
            /* do cleanup here — possibly rethrow e */
            System.out.println("\nSyntax Error");
            e.printStackTrace();
        }
    }
}
```

Un Makefile plus complet

```

# commentaire: un Makefile pour jflex et cup
SRCFLEX=calcul.flex
SRCCUP=calcul.cup
JSRC=Lexi
CSRC=parser

all: Main.class

Main.class : $(JSRC).class sym.class $(CSRC).class

$(JSRC).java : $(SRCFLEX) sym.java
             jflex $(SRCFLEX)

sym.java parser.java : $(SRCCUP)
             jcup $(SRCCUP)

clean:
    rm -rf *.class *~ $(CSRC).java $(SRCFLEX).java sym.java

.SUFFIXES: .class .java

.java.class:
    javac $<

```