

# Introduction to the 1st Obligatory Exercise

## Scanning, parsing, constructing the abstract syntax tree

Department of Informatics  
University of Oslo

Compiler technique, Friday 22 2013

# Scanner and Parser Generators

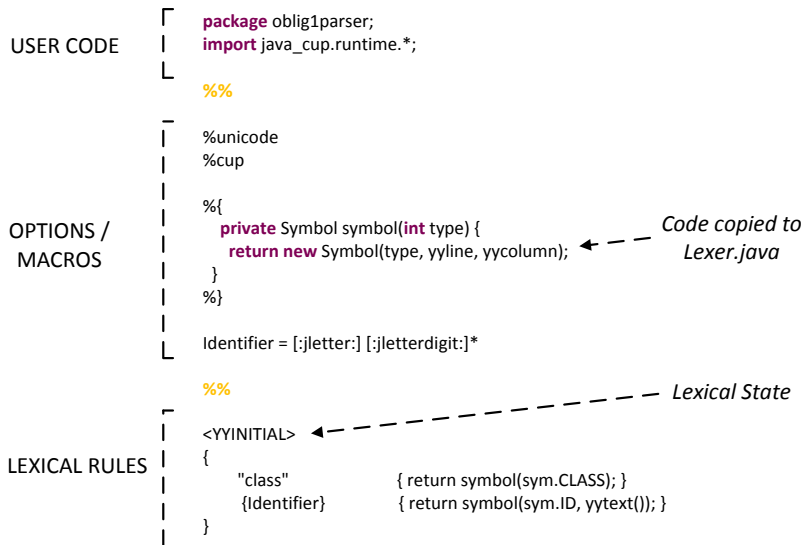
## JFlex

### JFlex

- Is specified in `.lex` file
- Specification consists of three parts:
  - ▶ User code
  - ▶ Options and macros
  - ▶ Lexical rules

# Scanner and Parser Generators

JFlex



# Scanner and Parser Generators

## CUP

### Java Based Constructor of Useful Parsers (CUP)

- Is specified in `.cup` file
- Specification consists of five parts:
  - ▶ Package and import specifications
  - ▶ User code components
  - ▶ Symbol list
  - ▶ Precedence declarations
  - ▶ Grammar

# Scanner and Parser Generators

## CUP

PACKAGE / IMPORTS	<pre>package oblig1parser; import java_cup.runtime.*; import syntaxtree.*;</pre>	
USER CODE	<pre>parser code { : };</pre>	<i>Code copied to parser.java</i>
SYMBOL LIST	<pre>terminal CLASS; terminal String ID; non terminal ClassDecl class_decl, decl;</pre>	
PRECEDENCE DECLARATIONS	<pre>precedence left OR; precedence left AND;</pre>	
GRAMMAR	<pre>program ::= decl_list:dl { RESULT = new Program(dl); } ; decl_list ::= decl:d { List&lt;ClassDecl&gt; l = new LinkedList&lt;ClassDecl&gt;(); l.add(d); RESULT = l; }   decl_list:dl decl:d { dl.add(d); RESULT = dl; } ; decl ::= class_decl:sd { RESULT = sd; } ; class_decl ::= CLASS ID:name LBRACK RBRACK { RESULT = new ClassDecl(name); }</pre>	

# Scanner and Parser Generators

## CUP

PACKAGE / IMPORTS	<pre>package oblig1parser; import java_cup.runtime.*; import syntaxtree.*;</pre>
USER CODE	<pre>parser code { : :};</pre>
SYMBOL LIST	<pre>terminal CLASS; terminal String ID; non terminal ClassDecl class_decl, decl;</pre>
PRECEDENCE DECLARATIONS	<pre>precedence left OR; precedence left AND;</pre>
GRAMMAR	<pre>program ::= decl_list:dl { RESULT = new Program(dl); :} ; 5. 3. 4. decl_list ::= decl:d { List&lt;ClassDecl&gt; l = new LinkedList&lt;ClassDecl&gt;(); l.add(d); RESULT = l; :}   decl_list:dl decl:d { dl.add(d); RESULT = dl; :} ; decl ::= class_decl:sd { RESULT = sd; :} ; class_decl ::= CLASS ID:name LBRACK RBRACK { RESULT = new ClassDecl(name); :};</pre> <p>The diagram shows five numbered arrows indicating the flow of execution:</p> <ul style="list-style-type: none"><li>1. From the <code>class_decl</code> rule to the <code>PRECEDENCE DECLARATIONS</code> section.</li><li>2. From the <code>decl</code> rule to the <code>PRECEDENCE DECLARATIONS</code> section.</li><li>3. From the <code>decl_list</code> rule to the <code>PRECEDENCE DECLARATIONS</code> section.</li><li>4. From the <code>program</code> rule to the <code>PRECEDENCE DECLARATIONS</code> section.</li><li>5. From the <code>program</code> rule to the <code>USER CODE</code> section.</li></ul>

# Scanner and Parser Generators

CUP

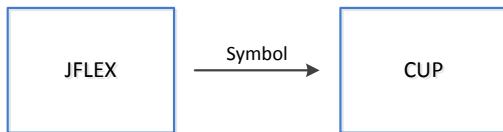
## Conflicts

Two types of conflicts may occur:

- Shift-Reduce in which CUP chooses to **reduce**
- Reduce-Reduce in which CUP chooses **first rule**

# Scanner and Parser Generators

## Integration



```
"class" { return symbol(sym.CLASS); }
```

```
public interface sym  
{  
    public static final int NOT = 36;  
    public static final int AND = 21;  
    public static final int RPAR = 42;  
    public static final int COMMA = 37;  
    public static final int CLASS = 2;  
    public static final int REFERENCE = 5;  
    ...  
}
```



# Abstract syntax tree model

## Using Java

Each **required** node in the tree model is represented by a Java class

```
package syntaxtree.nonterminals;
import java.util.List;
import utils.Utils;

public class ClassDecl extends Decl {
    private String name;
    private List<VarDecl> varDecls;

    public ClassDecl( String name, List<VarDecl> varDecls ) {
        this.name = name;
        this.varDecls = varDecls;
    }

    public void extractAST( StringBuilder sb, String indent ) {
        sb.append( "\n" + indent + "(CLASS_DECL (NAME " + name + ")");
        ...
    }
}
```

# Overview of provided code



Class files for compiler, lexer, parser, syntaxtree, etc.

build



**Oblila source code**

oblila-code



**Three pairs of .lex/.cup files**

grammars



**Java source code for compiler, syntax tree, etc.**

src



Test file for example parser

input-examples



Java source code example syntax tree

src-examples



JFlex and CUP libs

lib



Generated Java source code for lexer and parser

src-gen



**Generated abstract syntax tree**

oblila-ast

# Overview of provided code

grammars



grammars

**Three pairs of .lex/.cup files**

## Three grammar pairs

- expression-eval (example expression language)
- expression-par (example language handling parentheses)
- **oblig1** (oblig grammars)

# Overview of provided code

oblila-ast



oblila-ast

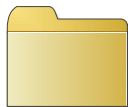
**Generated abstract syntax tree**

One abstract syntax tree

- Oblila.ast

# Overview of provided code

oblila-code



oblila-code

**Oblila source code**

One Oblila source file

- Oblila.obl

# Overview of provided code

src



src

**Java source code for  
compiler, syntax tree, etc.**

## Two folders

- compiler
- **syntaxtree**

# Using build script

## Ant

### Ant targets

**build.xml** contains an Ant-based build script with targets and task definitions

- \$ ant **clean**
- \$ ant **build**
- \$ ant **run**
- ...

# The tasks of this assignment

## Three main parts

- Create JFlex specification
- Create CUP specification (two versions)
- Define abstract syntax tree model using Java

```
USER CODE { package oblig.parser;
import java_cup.runtime.*;
%N
%unicode
%ncup
}
OPTIONS /
MACROS { %E
private Symbol symbol(int type) {
return new Symbol(type, yyline, yycolumn);
}
}
%S
identifier = [{letter}] [{letterdigit}]*
}
LEXICAL RULES {
<YYINITIAL>
{ "class" { returns symbol(sym.CLASS); }
@identifier { returns symbol(sym.ID, yytext()); }
}
```

```
PACKAGE /
IMPORTS { package oblig.parser;
import java_cup.runtime.*;
import syntree.*;
}
USER CODE { parser code { };
}
SYMBOL LIST { terminal CLASS;
terminal ID;
non terminal ClassDecl class_decl, decl;
}
PRECEDENCE DECLARATIONS { precedence left Cst;
precedence left AND;
}
GRAMMAR { program ::= decl_list dl { RESULT = new Program(dl); }
;
decl_list ::= decl dl { List<ClassDecl> l = new LinkedList<ClassDecl>(); l.add(dl); RESULT = l; }
;
decl ::= class_decl cd { RESULT = cd; }
;
class_decl ::= CLASS ID name LBRACK RRBRACK { RESULT = new ClassDecl(name); }
```

Code copied to  
parser.java

```
package syntree.nonterminals;
import java.util.List;
import util.Util;

public class ClassDecl extends Decl {
private String name;
private List<VarDecl> varDecls;

public ClassDecl(String name, List<VarDecl> varDecls) {
this.name = name;
this.varDecls = varDecls;
}

public void extractAST() StringBuilder sb, String indent ) {
sb.append(indent + "jn" + indent + "[CLASS_DECL (NAME " + name + ")"];
...
}
}
```



# Additional information

## More information

- Exercise and Oblila descriptions
- JFlex manual
- CUP manual