JFlex Lexical Analyzer Generator

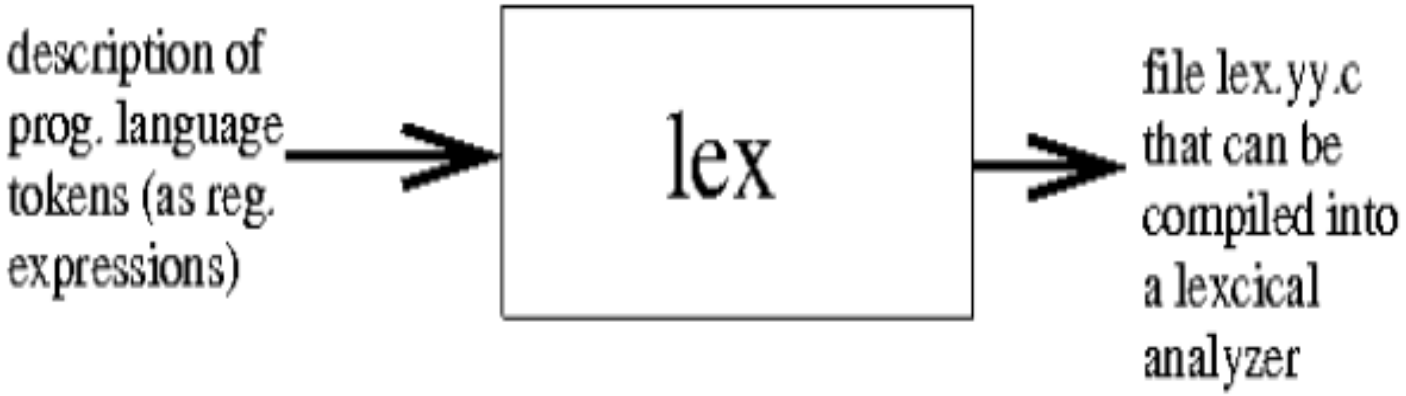
CSE420: Compiler Design

Fall 2015

Introduction

- JFlex is a lexical analyzer generator written in Java.
- Write C/Java program for lexical analysis by hand
  - Tiresome and tricky
- Use of lexical analyzer generator makes life easier -
  - Generates analyzer automatically from “descriptions" (regular expressions) of tokens in the programming language.

Introduction



Input and Output of JFlex Program

- **Input**
  - Description of token structure (regular expressions).
  - Info. on how to “process" different tokens.
- **Output**
  - Implementation of NFA-based function that
    - Recognizes tokens (as specified by RE rules)
    - Processes them (as specified by actions)

Overview of JFlex

- JFlex takes a JFlex program and creates a Java file (lexical analyzer).
- The default name for the Java class generated is *Yylex*, and the code is written to a file called *Yylex.java*.
- The lexical analyzer class has a method for getting a token. The default name for this method is *yylex()*.

Structure of JFlex Program

```
/* User code */

%%

/* Options and declarations */

%%

/* Lexical Rules */
```

- Lexical specification file for JFlex consists of three parts divided by a single line starting with %%.

User code

- The first part contains user code that is copied verbatim into the beginning of the source file of the generated *lexer* before the scanner class is declared.

- This is the place to put package declarations and import statements.

Options and Declarations

- Contains options to customize generated *lexer*.
- JFlex directives.
  - Each JFlex directive must be situated at the beginning of a line and starts with the % character.
  - Example:

    %class myclass.

    means that you start a line with %class followed by a space followed by the name of the class for the generated scanner (“myclass in this case”).

- Java code to include in different parts of the lexer.
- Declarations of lexical states.
- Macro definitions for use in the third section “Lexical rules" of the lexical specification file.

Some Built-in Methods and Fields

- **int yychar**

  Represents the number of characters processed since the start of input.

- **int yyline**

  Represents the number of line breaks processed since the start of input.

- **int yycolumn**

  Represents the number of characters processed since the start of the current line.

- **String yytext()**

  Returns the text matched by the current rule.

- **int yylength()**

  Returns the length of the text matched by the current rule.

- **int yystate()**

  Returns the current state.

- **void yybegin(int lexicalState)**

  Sets the current state.

Some Jflex Directives

- **%class <classname>**

  To give the generated class the name <classname> and to write the generated code to a file "classname.java".

  Example: %class myclass will give the generated class name ‘mylcass’ and the java file ‘myclass.java’

- **%implements <interface>**

  Makes the generated class implement the specified interfaces.

- **%extends <classname>**

  Makes the generated class a subclass of the specified class.

Some Jflex Directives

- **%public**

  Makes the generated class public

- **%standalone**

  Creates a main function in the generated class that expects the name of an input le on the command line and then runs the scanner on this input file.

- **%char**

  Turns character counting on. *yychar* contains the number of Characters.

- **%column**

  Turns column counting on. *yycolumn* contains the number of column.

Some Jflex Directives

- **%function <name>**

  Causes the scanning method to get the specified name. If no %function directive is present in the specification, the scanning method gets the name “yylex“.

- **%type <typename>**

  Causes the scanning method to be declared as returning values of the specified type.

Some JFlex Directives

- **%{**

  <Code>

  %}

The code enclosed in %{ and %} is copied verbatim into the generated class. Here you can define your own member variables and functions in the generated scanner. Both %{ and %} must start a line in the specification.

- %{ <Code> %} - **Wrong**
- **%{**

  <Code>

  %} - **Correct**

Some JFlex Directives

- **%init{**

  <Code>

  %init}

- The code enclosed in %init{ and %init} is copied verbatim into the constructor of the generated class.
- Here, member variables declared in the %{...%} directive can be initialised.

Macro Definition

- You can attach names (e.g. Whitespace) to Regular Expression for brevity/clarity – Macro Definition.
- A macro definition has the form -

  macroidentifier = regular expression
- Example:

  Whitespace = [\t\n]+

  Letter = [a-zA-Z]
- Later in the *Lexical Rule* part, instead of [\t\n]+ we can use {Whitespace}.

Lexical Rules

- Third section of Jflex program.
- Lexical Rules

  Rule = Pattern + Action

  Pattern = Regular Expression

  Action = Snippet of Java code (Actions triggered whenever pattern matched)

Example:

```
(a|b)*    { System.out.println("*** found match\n");}
```

Pattern

Action

Lexical Rule

- When a pattern is matched, the action specified by the author is performed.
- “Unmatched" portions of the input are copied unaltered into output.

Putting Things Together!

Example 1

Search through source and report any occurrences of pattern (a|b)*abb found.

Solution

```
%%
%class search
%standalone
%line
%column
%%
(a|b)*abb     {System.out.printf(
               "*** found match [%s] at line %d, column %d\n",
               yytext(), yyline, yycolumn);
              }

\n              { /* Do Nothing */}
.               {/* Do Nothing */}
```

Explanation

- **Pattern:** (a|b)*abb
- **Action:** {System.out.println}
- **Operation:** For each match detected (i.e. string consisting of as and bs that ends abb), the action (i.e. println) is performed.

Explanation

```
%%
(a|b)*abb    {System.out.printf(
              "*** found match [%s] at l%d, c%d\n ",
              yytext(), yyline, yycolumn);
             }
\n           { /* do nothing */}
.            { /* do nothing */}
```

- By default, unmatched fragments of input are echoed unaltered to output.
- Final two rules suppress this (by gobbling up every char not matched by main rule).
- RE . (dot) matches any char except newline.

Refined Solution Using Macro Definition

```
%%
%class search
%standalone
%line
%column
Whitespace = [ \t\n]+
%%
(a|b)*abb     {  System.out.printf(
                 "*** found match [%s] at line %d, column %d\n",
                 yytext(), yyline, yycolumn);
                }
{Whitespace}                 { /* Do Nothing */}
.                            {/* Do Nothing */}
```

Execution of JFlex

```
jflex <jflex file name>
javac Yylex.java
java Yylex <input file name>
```

- jFex generates Java file *YYlex.java* .
- Executing *Yylex.class* reads from input txt file and prints "*** …..” once for each occurrence of a string matching RE (a|b)*abb.

Reference

- JFlex User's Manual, Version 1.6.1, March 16, 2015.
- Lexical Analysis and jFlex, Dr Kieran T. Herley, Department of Computer Science, University College Cork, 2015/16.