|  |
| --- |
| CS321 |
| Homework2 |
| The Lexer |
|  |
| **Ben glasser** |
| **10/18/2012** |

|  |
| --- |
| This assignment accomplishes two goals. First a lexer is built to recognize four specified tokens for the following, keywords, comments, integer literals, and invalid tokens. Second, rules are created and added to the lexer in order to recognize two types of integer literals as valid, anything not matching these rules will be discarded as invalid. The result is that it will recognize all integers in the range of 0 – 2147483647 as well as all multiples of four.  A git repository has been created for this assignment at <https://github.com/BenGlasser/Lexer>  This repo can be cloned by using the following command:  git clone https://github.com/BenGlasser/Lexer.git |
|  |

# Question 1

This main goal of this task was to implement some rudimentary syntax highlighting for mini using html. In order to accomplish this It was only necessary to have rules to recognize keywords, literals, comments, and invalids. Since the text was going to be output to HTML it was also necessary to recognize ‘<’, ‘>’, and ‘&’ in order to encode these characters properly in the HTML file. I implemented this by adding a function to the jflex file which would return specific tokens surrounded by the proper span tags as follows:

…

public String getTag**()** **{**

**switch** **(**token**)** **{**

**case** AMP **:** **return** "&amp;"**;**

**case** GT **:** **return** "&gt;"**;**

**case** LT **:** **return** "&lt;"**;**

**case** COMMENT **:** **return** "<span class=\"comment\">"**+**yytext**()+**"</span>"**;**

**case** KEYWORD **:** **return** "<span class=\"keyword\">"**+**yytext**()+**"</span>"**;**

**case** INTLIT **:** **return** "<span class=\"literal\">"**+**yytext**()+**"</span>"**;**

**case** INVLD **:** **return** "<span class=\"invalid\">"**+**yytext**()+**"</span>"**;**

**default** **:** **return** yytext**();**

**}**

**}  
…**

Each case in the switch statement above is returned by using the following regexs and rules:

Identifier **=** **[:**jletter**:]** **[:**jletterdigit**:]\***

Char **=** **[:**jletter**:]|**"{"**|**"}"**|**"["**|**"]"**|**"("**|**")"**|**","**|**";"**|**"/"**|**"-"**|**"+"**|**"\*"**|**"^"**|**"|"**|**"~"**|**"!"**|**"="**|**"\""**|**"%"**|**":"**|**"."**|**"'"**|**"?"**|**"@"**|**"#"

LineTerminator **=** \r**|**\n**|**\r\n

WhiteSpace **=** **{**LineTerminator**}** **|** **[** \t\f**]**

InputCharacter **=** **[^**\r\n**]**

Comment **=** **{**TraditionalComment**}** **|** **{**EndOfLineComment**}**

TraditionalComment **=** "/\*" **[^\*]** **~**"\*/" **|** "/\*" "\*"**+** "/"

EndOfLineComment **=** "//" **{**InputCharacter**}\*** **{**LineTerminator**}**

AnyInt **=** **[**0**-**9**]+**

MultFour **=** **((**0**|**4**|**8**)|([**0**-**9**]\*(**00**|**04**|**08**|**12**|**16**|**20**|**24**|**28**|**32**|**36**|**40**|**44**|**48**|**52**|**56**|**60**|**64**|**68**|**72**|**76**|**80**|**84**|**88**|**92**|**96**)))**

MaxInt **=** **([**0**-**9**]{**1**,**9**}|**1**[**0**-**9**]{**9**}|**2**(**0**[**0**-**9**]{**8**}|**1**([**0**-**3**][**0**-**9**]{**7**}|**4**([**0**-**6**][**0**-**9**]{**6**}|**7**([**0**-**3**][**0**-**9**]{**5**}|**4**([**0**-**7**][**0**-**9**]{**4**}|**8**([**0**-**2**][**0**-**9**]{**3**}|**3**([**0**-**5**][**0**-**9**]{**2**}|**6**([**0**-**3**][**0**-**9**]|**4**[**0**-**7**])))))))))**

**%%**

/\*re-encode html\*/

">" **{** **return** GT**;**  **}**

"<" **{** **return** LT**;** **}**

"&" **{** **return** AMP**;** **}**

/\*keywords\*/

"int" **{** **return** KEYWORD**;** **}**

"boolean" **{** **return** KEYWORD**;** **}**

"double" **{** **return** KEYWORD**;** **}**

"void" **{** **return** KEYWORD**;** **}**

"while" **{** **return** KEYWORD**;** **}**

"if" **{** **return** KEYWORD**;** **}**

"else" **{** **return** KEYWORD**;** **}**

"print" **{** **return** KEYWORD**;** **}**

"return" **{** **return** KEYWORD**;** **}**

"new" **{** **return** KEYWORD**;** **}**

/\*Integers\*/

**{**MaxInt**}** **{** **return** INTLIT**;** **}**

**{**MultFour**}** **{** **return** INTLIT**;** **}**

**{**AnyInt**}** **{** **return** INVLD**;** **}**

/\*everything else\*/

**{**Identifier**}** **{** **return** IDENT**;** **}**

**{**WhiteSpace**}** **{** **return** WHITESPACE**;** **}**

**{**Comment**}** **{** **return** COMMENT**;** **}**

**{**Char**}** **{** **return** IDENT**;** **}**

**.|**\n **{** **return** INVLD**; }**

I call the getTag() function above from my PageGen.java code. PageGen.java is used to build the html page with highlighting from the mini file. I have included an annotated copy of the code below:

…

public class PageGen **{**

// initialize a string builder with thi initial elements of an HTML file.

private static final String header **=**

"<html>\n" **+**

"<head>\n" **+**

"<title>My Syntax Colored Web Page</title>\n" **+**

"<style type=\"text/css\">\n" **+**

" body {white-space:pre;\n" **+**

" background-color:#ffffcc;\n" **+**

" color:black;\n" **+**

" font-family:\"Lucida Console\",\"Courier New\",Monotype}\n" **+**

" .keyword {color:blue}\n" **+**

" .comment {color:orange}\n" **+**

" .literal {color:green}\n" **+**

" .invalid {color:red}\n" **+**

"</style>\n" **+**

"</head>\n" **+**

"<body>"**;**

// This constant will be appended at the end of every html file in order to close the opening

// head and body tags

private static final String footer **=** "</body>\n</html>"**;**

// The main function of PageGen.java takes 2 arguments in the following form

// PageGen [Input File] [Output File]

public static void main**(**String**[]** args**)** **{**

StringBuilder output **=** **new** StringBuilder**(**header**);**

**if** **(**args**.**length**!=**2**)** **{**

System**.**out**.**println**(**"Program requires exactly two arguments"**);**

**}** **else** **{**

Handler handler **=** **new** SimpleHandler**();**

**try** **{**

FileReader reader **=** **new** FileReader**(**args**[**0**]+**".mini"**);**

MiniLexer lexer **=** **new** MiniLexer**(**reader**,** handler**);**

// this is th business loop of main() it retrieves tokens from the lexer surrounded

// by the appropriate tags and appends them to the output string

**while** **(**lexer**.**nextToken**()** **!=** MiniTokens**.**ENDINPUT**)** **{**

output**.**append**(**lexer**.**getTag**());**

**}**

// at this point the entire input program has been read and formatted so the footer

// is appended to the output string and the entire string is output to an html file

System**.**out**.**println**();**

output**.**append**(**footer**);**

printFile**(**args**[**1**],** output**.**toString**());**

**}** **catch** **(**Exception e**)** **{**

handler**.**report**(new** Failure**(**"Exception: " **+** e**));**

**}**

**}**

**}**

public static void printFile**(**String file**,** String text**){**

**try** **{**

FileWriter writer **=** **new** FileWriter**(**file **+** ".html"**);**

writer**.**write**(**text**);**

writer**.**close**();**

**}** **catch** **(**IOException e**)** **{**

e**.**printStackTrace**();**

**}**

**}**

**}**

Now, to be clear, my lexer follows the same structure as the Mini lexer from which it was derived. It includes three files; Mini.jflex, MiniTokens.java, PageGen.java. PageGen.java was written from scratch, more or less, it’s only fucnction is to get tokens and output them to an HTML file. It takes EXACTLY two commands on the command line as follows:   
  
java PageGen [Input Filename] [Output Filename]   
  
MiniTokens.java is implemented by the jflex file and subsequently in the jflex generated java file. It is used as a way to enumerate various tokens.

Mini.jflex is the jflex file which contains all the rules for breaking text up into tokens. I added a function in this file called getTag() which returns tokens surrounded by their respective html tags. It also recognizes certain individual characters but does not wrap them in any tag so they will be printed verbatim by the browser as a result.

# Question 2

1. This problem requires us to find a regex which matches numbers in the range of 0 – 2147483647. In order to derive the regex I worked the integer from left to right ‘or’ing together number patterns to match each successive digit in the integer.   
      
   1[0-9]{9}  
   20[0-9]{8}  
   21[0-3][0-9]{7}  
   214[0-6][0-9]{6}  
   2147[0-3][0-9]{5}  
   21474[0-7][0-9]{4}  
   214748[0-2][0-9]{3}  
   2147483[0-5][0-9]{2}  
   21474836[0-3][0-9]  
   214748364[0-7]  
     
   Then I ‘or’ together each possible regex and end up with this:   
      
   ([0-9]{1,9}|1[0-9]{9}|20[0-9]{8}|21[0-3][0-9]{7}|214[0-6][0-9]{6}|2147[0-3][0-9]{5}|21474[0-7][0-9]{4}|214748[0-2][0-9]{3}|2147483[0-5][0-9]{2}|21474836[0-3][0-9]|214748364[0-7])  
     
   This works because ultimately what is happening is a valid acceptable range of numbers is being stitched together for strings of single integers . Now because this regex will be valid for all numbers from 0 – 2147483647 I put it as the first “integer” rule in the rule table and have it return the token value for INTLIT. Thus as soon as the lexer finds a string of numbers which falls into this range it moves on to trying to match the next token. Now because the lexer also might match a string of digits as valid single digits I may get a result which appears as if the lexer has returned an integer greater than “MaxInt” as valid such as the following string:   
     
    2147483648  
     
   Clearly 2147483648 > 2147483647 so it should be invalid, is the lexer wrong? NO. What has happened is that the lexer returned two separate tokens as valid those two tokens are 214748364 followed by 8. To avoid this I included a rule in the lexer to return EVERY single digit as invalid. Now the lexer will see strings of digits larger than MaxInt as a string of single invalid digits.

1. To answer this question I wrote a quick program to print out all multiples of four from 1 – 500. What I noticed is that the last digit of each number was alternating between 0 4 8 2 6. I was tempted to use this immediately and just check all strings of digits to see if they end in one of these numbers, but that wouldn't work, since other integers also end with those digits that aren't themselves divisible by four such as 10, 14, 18, 22, 26, etc... Next I looked at the last two digits and noticed a repeating pattern between 0 and 100

4|8|12|16|20|24|28|32|36|40|44|48|52|56|60|64|68|72|76|80|84|88|92|96|100|...|204|208|212|...

To test my theory further I pulled up Google and typed in 2147483648 % 4 and as it turns out 2147483648 % 4 = 0 so I felt pretty confident my theory was right. I suppose you could actually write out a mathematical proof and prove that this theory works, but I'm more into application. So I figure at this point all I have to do is write up this regex and then I can test it against the output of the program written above. So my next goal is to write the actual regex.

(00|04|08|12|16|20|24|28|32|36|40|44|48|52|56|60|64|68|72|76|80|84|88|92|96)

I prefixed the zeros to the single digits and added 00 to the front because I want to match the last TWO chars including the 00 from 100 (this will also return strings of 0 as a valid multiple of four as it should). So now I have my regex suffix written. According to my theory any string of digits suffixed by the aforementioned two digits is a multiple of four so I just need to write a rule for the prefix, any digit, [0-9]\* So now my regex looks like this:

[0-9]\*(00|04|08|12|16|20|24|28|32|36|40|44|48|52|56|60|64|68|72|76|80|84|88|92|96)

Now I'm almost done. What have I forgotten? Single digits!!! 0,4 and 8 will be rejected by the regex above since they are single digits and the above pattern only matches two digits preceded by 0 or more digits. so I tweaked the regex a little and I end up with this:

(0|4|8)|([0-9]\*(00|04|08|12|16|20|24|28|32|36|40|44|48|52|56|60|64|68|72|76|80|84|88|92|96))

**—Attached are printouts of all code used in this assignment—**

(with the exception of auto generated files from Mini.jflex)

Mini.jflex

package mini;

import compiler.\*;

import java.io.\*;

%%

%class MiniLexer

%public

%extends Phase

%implements MiniTokens

%ctorarg Handler handler

%init{

super(handler);

%init}

%function yylex

%int

%eofval{

return ENDINPUT;

%eofval}

%{

private int token;

public int getToken() {

return token;

}

public int nextToken() {

try {

token = yylex();

} catch (java.io.IOException e) {

System.out.println("IO Exception occurred:\n" + e);

}

return token;

}

public String getTag() {

switch (token) {

case AMP : return "&amp;";

case GT : return "&gt;";

case LT : return "&lt;";

case COMMENT : return "<span class=\"comment\">"+yytext()+"</span>";

case KEYWORD : return "<span class=\"keyword\">"+yytext()+"</span>";

case INTLIT : return "<span class=\"literal\">"+yytext()+"</span>";

case INVLD : return "<span class=\"invalid\">"+yytext()+"</span>";

default : return yytext();

}

}

%}

Identifier = [:jletter:] [:jletterdigit:]\*

Char = [:jletter:]|"{"|"}"|"["|"]"|"("|")"|","|";"|"/"|"-"|"+"|"\*"|"^"|"|"|"~"|"!"|"="|"\""|"%"|":"|"."|"'"|"?"|"@"|"#"

LineTerminator = \r|\n|\r\n

WhiteSpace = {LineTerminator} | [ \t\f]

InputCharacter = [^\r\n]

Comment = {TraditionalComment} | {EndOfLineComment}

TraditionalComment = "/\*" [^\*] ~"\*/" | "/\*" "\*"+ "/"

EndOfLineComment = "//" {InputCharacter}\* {LineTerminator}

AnyInt = [0-9]+

MultFour = ((0|4|8)|([0-9]\*(00|04|08|12|16|20|24|28|32|36|40|44|48|52|56|60|64|68|72|76|80|84|88|92|96)))

MaxInt = ([0-9]{1,9}|1[0-9]{9}|20[0-9]{8}|21[0-3][0-9]{7}|214[0-6][0-9]{6}|2147[0-3][0-9]{5}|21474[0-7][0-9]{4}|214748[0-2][0-9]{3}|2147483[0-5][0-9]{2}|21474836[0-3][0-9]|214748364[0-7])

%%

/\*re-encode html\*/

">" { return GT; }

"<" { return LT; }

"&" { return AMP; }

/\*keywords\*/

"int" { return KEYWORD; }

"boolean" { return KEYWORD; }

"double" { return KEYWORD; }

"void" { return KEYWORD; }

"while" { return KEYWORD; }

"if" { return KEYWORD; }

"else" { return KEYWORD; }

"print" { return KEYWORD; }

"return" { return KEYWORD; }

"new" { return KEYWORD; }

/\*Integers\*/

{MaxInt} { return INTLIT ; }

{MultFour} { return INTLIT ; }

{AnyInt} { return INVLD ; }

/\*everything else\*/

{Identifier} { return IDENT; }

{WhiteSpace} { return WHITESPACE ; }

{Comment} { return COMMENT ; }

{Char} { return IDENT ; }

.|\n { return INVLD;}

PageGen.java

**import** compiler**.**Failure**;**

**import** compiler**.**Handler**;**

**import** compiler**.**SimpleHandler**;**

**import** mini**.**MiniLexer**;**

**import** mini**.**MiniTokens**;**

/\*\*

\* Created with IntelliJ IDEA.

\* User: Ben

\* Date: 10/14/12

\* Time: 3:22 PM

\* To change this template use File | Settings | File Templates.

\*/

**import** java**.**io**.**FileReader**;**

**import** java**.**io**.**FileWriter**;**

**import** java**.**io**.**IOException**;**

**import** compiler**.\*;**

**import** mini**.\*;**

public class PageGen **{**

private static final String header **=**

"<html>\n" **+**

"<head>\n" **+**

"<title>My Syntax Colored Web Page</title>\n" **+**

"<style type=\"text/css\">\n" **+**

" body {white-space:pre;\n" **+**

" background-color:#ffffcc;\n" **+**

" color:black;\n" **+**

" font-family:\"Lucida Console\",\"Courier New\",Monotype}\n" **+**

" .keyword {color:blue}\n" **+**

" .comment {color:orange}\n" **+**

" .literal {color:green}\n" **+**

" .invalid {color:red}\n" **+**

"</style>\n" **+**

"</head>\n" **+**

"<body>"**;**

private static final String footer **=** "</body>\n</html>"**;**

public static void main**(**String**[]** args**)** **{**

StringBuilder output **=** **new** StringBuilder**(**header**);**

**if** **(**args**.**length**!=**2**)** **{**

System**.**out**.**println**(**"Program requires exactly two arguments"**);**

**}** **else** **{**

Handler handler **=** **new** SimpleHandler**();**

**try** **{**

FileReader reader **=** **new** FileReader**(**args**[**0**]+**".mini"**);**

MiniLexer lexer **=** **new** MiniLexer**(**reader**,** handler**);**

**while** **(**lexer**.**nextToken**()** **!=** MiniTokens**.**ENDINPUT**)** **{**

output**.**append**(**lexer**.**getTag**());**

**}**

System**.**out**.**println**();**

output**.**append**(**footer**);**

printFile**(**args**[**1**],** output**.**toString**());**

**}** **catch** **(**Exception e**)** **{**

handler**.**report**(new** Failure**(**"Exception: " **+** e**));**

**}**

**}**

**}**

public static void printFile**(**String file**,** String text**){**

**try** **{**

FileWriter writer **=** **new** FileWriter**(**file **+** ".html"**);**

writer**.**write**(**text**);**

writer**.**close**();**

**}** **catch** **(**IOException e**)** **{**

e**.**printStackTrace**();** //To change body of catch statement use File | Settings | File Templates.

**}**

**}**

**}**

MiniToken.java

// Output created by jacc on Sun Sep 16 16:33:19 PDT 2012

package mini**;**

public interface MiniTokens **{**

int ENDINPUT **=** 0**;**

int IDENT **=** 6**;**

int INTLIT **=** 9**;**

int COMMENT **=** 22**;**

int WHITESPACE **=** 23**;**

int KEYWORD **=** 24**;**

int AMP **=** 25**;**

int GT **=** 26**;**

int LT **=** 27**;**

int INVLD **=** 28**;**

**}**