GC04 Compilers coursework 2016

Introduction

This year's coursework involves writing a lexer and parser using JFlex (or Jlex if you prefer) and CUP. The language for which you must write this is JSON (JavaScript Object Notation). JSON is a plain text, lightweight data-interchange format commonly used to transfer data between a web application and a client. You can find the specification for JSON at http://www.json.org. If you have not encountered web services that use JSON then you might like to look at the examples on this web site: http://www.theyworkforyou.com/api/. You can complete the forms to create web service queries such as the one below which returns details of the current member of parliament for the Holborn and St Pancras constituency in the UK.

URL for this:
http://www.theyworkforyou.com/api/getMP?postcode=wc1e+6bt&output=js


```
{
  "member_id" : "40319",
  "0" : "40319",
  "house" : "1",
  "1" : "1",
  "first_name" : "Frank",
  "2" : "Frank",
  "last_name" : "Dobson",
  "3" : "Dobson",
  "constituency" : "Holborn and St Pancras",
  "4" : "Holborn and St Pancras",
  "party" : "Labour",
  "5" : "Lab",
  "entered_house" : "2010-05-06",
  "6" : "2010-05-06",
  "left_house" : "9999-12-31",
  "7" : "9999-12-31",
  "entered_reason" : "general_election",
  "8" : "general_election",
  "left_reason" : "still_in_office",
  "9" : "still_in_office",
  "person_id" : "10171",
  "10" : "10171",
  "title" : "",
  "11" : "",
  "lastupdate" : "2010-05-07 07:04:26",
  "12" : "2010-05-07 07:04:26",
  "full_name" : "Frank Dobson",
  "url" : "/mp/10171/frank_dobson/holborn_and_st_pancras",
  "image" : "/images/mps/10171.jpg",
  "image_height" : 59,
  "image_width" : 49
}
```


Many other organisations provide web service APIs as a means of delivering content to users. The benefits of choosing plain text as an interchange format over, say, binary may not seem obvious at first but by using plain text you are more able to "future proof" any software you write. Validation is an important aspect of dealing with the input and output of any computer system and using a plain text format makes it easier to write and maintain tools to do this.

Rae Harbird / Stephen Hailes

Your Task

Your task is to write a lexer and parser for JSON and this should be capable of indicating whether or not some JSON output conforms to the syntax of the language. The aim of this coursework is partly technical and partly aimed at encouraging you to find things out for yourselves. However, to get you started in this task, I have put an example of the use of JFLex and CUP on moodle. **This is intended to be used on a Unix system**.

If you unpack the example from minimal.tgz (type: `tar xzf minimal.tgz`), it will create a directory 'minimal' that contains a minimal example of the use of jflex and cup. It creates a parser for a grammar that looks like this:

```
array ::= LSQBRACKET value_list RSQBRACKET;
value_list ::= value_list COMMA value | value;
value      ::= INT
            ;
```

In other words, a simple list of numeric values enclosed in square brackets like this: [1,2,3,4,5]. To try this out, type the following:

```
$ ant jar
```

It will give you a bunch of output that ends……

```
BUILD SUCCESSFUL
Total time: 1 second
```

This has created a jar file that contains your parser, located in `jar/Compiler.jar` and you can run this by typing:

```
java -jar jar/Compiler.jar input.test
```

This will take input from the file `input.test`.

You should just type `java –jar jar/Compiler.jar` to take input from the command line. Do the latter. And then type `[3,5]` not forgetting the comma. It should do the following:

```
$ java -jar jar/Compiler.jar
 value 3
 value 5
```

Looking more deeply into how this is done, let's see what files there are in the directory:

```
ls minimal
$ bin  build.xml  cup  input.test
jflex  lib  src
```

There are only two directories we need to concern ourselves with: `cup` and `jflex` the remainder are necessary but to do with the system itself (and `input.test` is the example input file).

```
$ ls jflex
Scanner.jflex

$ ls cup
Parser.cup
```

Scanner.jflex

```
package Example;

import java_cup.runtime.SymbolFactory;
%%
%cup
%class Scanner
%{
 public Scanner(java.io.InputStream r, SymbolFactory sf){
  this(r);
  this.sf=sf;
 }
 private SymbolFactory sf;
%}
%eofval{
    return sf.newSymbol("EOF",sym.EOF);
%eofval}


%%
"," { return sf.newSymbol("Comma",sym.COMMA); }
"[" { return sf.newSymbol("Left Square Bracket",sym.LSQBRACKET); }
"]" { return sf.newSymbol("Right Square Bracket",sym.RSQBRACKET); }
[0-9]+ { return sf.newSymbol("Integral Number",sym.NUMBER, new Integer(yytext())); }
[ \t\r\n\f] { /* ignore white space. */ }
. { System.err.println("Illegal character: "+yytext()); }
```

Commented [r1]: Leave this part alone.

Commented [r2]: This is the part you should change. The second parameter should be of the form sym.SOMETHING where SOMETHING is defined in Parser.cup as a terminal.

Parser.cup

```
package Example;

import java_cup.runtime.*;

parser code {:
  public static void main(String args[]) throws Exception {
    SymbolFactory sf = new DefaultSymbolFactory();
    if (args.length==0)
      new Parser(new Scanner(System.in,sf),sf).parse();
    else
      new Parser(new Scanner(new java.io.FileInputStream(args[0]),sf),sf).parse();
  }
:}


terminal COMMA, LSQBRACKET, RSQBRACKET;
terminal Integer NUMBER;

non terminal array, value_list, value;
non terminal Integer expr;

array ::= LSQBRACKET value_list RSQBRACKET;
value_list ::= value_list COMMA value | value;

value::= expr:e {: System.out.println(" value: "+e+""); :} ;
expr      ::= NUMBER:n
          {: RESULT=n; :}
           ;
```

Commented [r3]: Leave this part alone.

Commented [r4]: This part you should change.

Rae Harbird / Stephen Hailes

Information on JFlex and CUP can be found online:
CUP: http://www2.cs.tum.edu/projects/cup/

JFlex: http://jflex.de/
[Alternatively, see JLex: http://www.cs.princeton.edu/~appel/modern/java/JLex]

To reiterate, your task is to create a parser for JSON. It need not output anything other than that the parsing succeeded or failed, but you are free to make it as sophisticated as you like. The coursework should be handed in electronically, on Moodle on Monday 21st November 2016, 12pm. Coursework submitted after 11th December 2016 will be disregarded. You should submit (1) a short (< 2 pages A4) description of your system; (2) commented lexer code; (3) commented CUP code; (4) the results of tests run on your parser. **You should zip all your files into a single file, which must be called yourname.zip**. You will be marked both for content and style – how easy it is for me to understand what you have done will be determined by how well you lay out your code, whether there are comments and whether it is clear how you tested it and why. If there are technical reasons why it is not possible for you to produce a parser for the full JSON specification, then you should discuss them.

This coursework will be marked out of a total of 5 marks.

Rae Harbird / Stephen Hailes

Appendix – Getting Started

To complete this coursework you are advised to use a Linux operating system. All the machines in the Computer Science labs are dual boot and you can to run Linux from the boot menu. As an alternative, you may choose to use the **Linux service**. You can access our linux service remotely but you must install the Thinlinc client first. More information about using the linux service is available here: http://tsg.cs.ucl.ac.uk/basics/connectivity/cs_remote_worker/. Please note: This web page is not accessible offsite unless you are using a VPN.

Setting your CLASSPATH

If you are using the Linux service or Linux machines in the Computer Science department you will need to make some changes to your CLASSPATH. In summary, you need to add the following path to your CLASSPATH:

```
/opt/UCLCScourseware/java/contrib
```

If you have not changed your CLASSPATH before then open the file `.uclcs-csh-options`, which is in your home directory, with your chosen editor. Add these lines to the end of the file:

```
#
# Set CLASSPATH
#
setenv CLASSPATH /opt/UCLCScourseware/java/contrib
```

Save the file and exit the editor. You have to logout and login again for the changes to take effect.[1] You are now ready to begin the coursework.

[1] This is only true on the machines in this department. Usually, you could just 'source' the file.