CSC 448: Compilers

Lecture 10

Joseph Phillips
De Paul University

2015 June 2

Reading

- (Nothing official)

Topics:

- Review
- When your implementation language is not C/C++
  - Javacc
- Should you be making your own language anyway?
  - XML (for “documents”) and JSON (for “data”)
  - PHP
  - libxml2

Review:

- Now that we have Flex/Bison it is quick and easy to create our own tokenizer and interpreter

Simple1.jj

```
options {
  LOOKAHEAD = 1;
  CHOICE_AMBIGUITY_CHECK = 2;
  OTHER_AMBIGUITY_CHECK = 1;
  STATIC = true;
  DEBUG_PARSER = false;
  DEBUG_LOOKAHEAD = false;
  DEBUG_TOKEN_MANAGER = false;
  ERROR_REPORTING = true;
  JAVA_UNICODE_ESCAPE = false;
  UNICODE_INPUT = false;
  IGNORE_CASE = false;
  USER_TOKEN_MANAGER = false;
  USER_CHAR_STREAM = false;
  BUILD_PARSER = true;
  BUILD_TOKEN_MANAGER = true;
  SANITY_CHECK = true;
  FORCE_LA_CHECK = false;
}
```

```
PARSER_BEGIN(Simple1)

public class Simple1 {
  public static void main(String args[]) throws ParseException {
    Simple1 parser = new Simple1(System.in);
    parser.Input();
  }
}

PARSER_END(Simple1)

void Input() :
{}
{
  MatchedBraces() ("\n"|"\r")* <EOF>
}

void MatchedBraces() :
{}
{
  "{" [ MatchedBraces() ] "}"
}
```

Let’s make it:

```
$ javacc Simple1.jj
Java Compiler Compiler Version 4.1 (Parser Generator)
(type "javacc" with no arguments for help)
Reading from file Simple1.jj . . .
File "TokenMgrError.java" is being rebuilt.
File "ParseException.java" is being rebuilt.
File "Token.java" is being rebuilt.
File "SimpleCharStream.java" is being rebuilt.
Parser generated successfully.
$ javac *.java
Note: Simple1.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
```

Let’s run it: (1)

- Some “good” cases:

```
$ java Simple1
{}
(Press Ctrl-D)
$ java Simple1
{{}}
(Press Ctrl-D)
```

Let’s run it: (2a)

- Some “less good” cases:

```
$ java Simple1
{x}
Exception in thread "main" TokenMgrError: Lexical error at
line 1, column 2.  Encountered: "x" (120), after : ""
at
Simple1TokenManager.getNextToken(Simple1TokenManager
.java:173)
at Simple1.jj_ntk(Simple1.java:193)
at Simple1.MatchedBraces(Simple1.java:40)
at Simple1.Input(Simple1.java:10)
at Simple1.main(Simple1.java:6)
```

Let’s run it: (2b)

- Some “less good” cases:

```
$ java Simple1
{ { } }
Exception in thread "main" TokenMgrError: Lexical error at
line 1, column 2.  Encountered: " " (32), after : ""
at
Simple1TokenManager.getNextToken(Simple1TokenManager
.java:173)
at Simple1.jj_ntk(Simple1.java:193)
at Simple1.MatchedBraces(Simple1.java:40)
at Simple1.Input(Simple1.java:10)
at Simple1.main(Simple1.java:6)
```

Let’s run it: (2c)

- Some “less good” cases:

```
$ java Simple1
{{}
Exception in thread "main" ParseException: Encountered " "\n"
"\n "" at line 1, column 4.
Was expecting:
    "}" ...

at Simple1.generateParseException(Simple1.java:230)
at Simple1.jj_consume_token(Simple1.java:168)
at Simple1.MatchedBraces(Simple1.java:48)
at Simple1.Input(Simple1.java:10)
at Simple1.main(Simple1.java:6)
```

–

Let’s consider the program (1):

```
options {
 LOOKAHEAD = 1;
 CHOICE_AMBIGUITY_CHECK = 2;
 OTHER_AMBIGUITY_CHECK = 1;
 STATIC = true;
 DEBUG_PARSER = false;
 DEBUG_LOOKAHEAD = false;
 DEBUG_TOKEN_MANAGER = false;
 ERROR_REPORTING = true;
 JAVA_UNICODE_ESCAPE = false;
 UNICODE_INPUT = false;
 IGNORE_CASE = false;
 USER_TOKEN_MANAGER = false;
 USER_CHAR_STREAM = false;
 BUILD_PARSER = true;
 BUILD_TOKEN_MANAGER = true;
 SANITY_CHECK = true;
 FORCE_LA_CHECK = false;
}
```

- Obviously various options for how the compiler will work
- See https://javacc.java.net/doc/docindex.html

Let’s consider the program (2):

```
PARSER_BEGIN(Simple1)

public class Simple1 {
  public static void main (String args[])
      throws ParseException
  {
    Simple1 parser =
        new Simple1(System.in);
    parser.Input();
  }
}

PARSER_END(Simple1)
```

- Compilation unit enclosed between "PARSER_BEGIN(name)" and "PARSER_END(name)"
- Must define a class called "name" - same as the arguments to PARSER_BEGIN and PARSER_END.
- Used as the prefix for the Java files generated by the parser generator.
- The parser code that is generated is inserted immediately before the closing brace of the class called "name"
- Here we make a parser and pass it a **java.io.InputStream** object (in this case **System.in**)
- Then call starting non-Terminal (**Input()**)

Let’s consider the program (3a):

```
void Input() :
{}
{
  MatchedBraces() ("\n"|"\r")*
<EOF>
}

void MatchedBraces() :
{}
{
  "{" [ MatchedBraces() ] "}"
}
```

- Two productions

```
Type lhs():
  { /* code to do */ }
  {
    /* pattern to match */
  }
```

Let’s consider the program (3b):

```
void Input() :
{}
{
  MatchedBraces() ("\n"|"\r")*
<EOF>
}

void MatchedBraces() :
{}
{
  "{" [ MatchedBraces() ] "}"
}
```

- Two productions

```
Type lhs():
  { /* code to do */ }
  {
    /* pattern to match */
  }
```

Let’s consider the program (3c):

```
void Input() :
{}
{
  MatchedBraces() ("\n"|"\r")*
<EOF>
}

void MatchedBraces() :
{}
{
  "{" [ MatchedBraces() ] "}"
}
```

- Two productions

```
Type lhs():
  { /* code to do */ }
  {
    /* pattern to match */
  }
```

Let’s consider the program (3d):

```
void Input() :
{}
{
MatchedBraces()("\n"|"\r")*<EOF>
}

void MatchedBraces() :
{}
{
  "{" [ MatchedBraces() ] "}"
}
```

- Two productions

```
Type lhs():
 { /* code to do */ }
 {
   /* pattern to match */
 }
```

The calculator parser, Javacc style

- From Shon Vick
- http://userpages.umbc.edu/~vick/431/Lectures/Spring06/3_LexicalAnalysis/3_Tools/2_JavaCC_Example.htm
- Downloaded 2015-06-01

SimpleCalc1.jj (1)

```
// -----------------------------------------------------------------------------
// SimpleCalc1.java
// -----------------------------------------------------------------------------

/*
 * Grammer Rules for a small language that describes basic arthmetric
 * expressions:
 *
 * expr     :=      number
 *          |       expr '+' expr
 *          |       expr '-' expr
 *          |       expr '*' expr
 *          |       expr '/' expr
 *          |       '(' expr ')'
 *          |       - expr
 * number   :=      digit+ ('.' digit+)?
 * digit    :=      '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
 *
 * Three production rules define the grammer elements:
 *      - expr
 *      - number
 *      - digit
 *
 * The following grammer will be used to build a simple command-line calculator.
 * First, we will need to translate the above EBNF grammer into JavaCC format.
 *
 * USAGE:
 *      % javacc SimpleCalc1.jj
 *      % java SimpleCalc1.java
 *      % java SimpleCalc1
 */
```

SimpleCalc1.jj (2)

```
options {
    LOOKAHEAD=2;
}

PARSER_BEGIN(SimpleCalc1)

public class SimpleCalc1 {

    public static void main(String[] args) throws ParseException {
        SimpleCalc1 parser = new SimpleCalc1(System.in);
        while (true) {
            parser.parseOneLine();
        }
    }

}

PARSER_END(SimpleCalc1)
```

SimpleCalc1.jj (3)

```
SKIP:
{
  " " | "\r" | "\t"
}

TOKEN:
{
  < NUMBER: ( <DIGIT> ) + ( "." ( <DIGIT> )+ )? >
  |
  < DIGIT:  [ "0"-"9" ] >
  |
  < EOL: "\n" >
}
```

SimpleCalc1.jj (4)

```
void parseOneLine():
{
  double a;
}
{
  a=expr() <EOL>
    {System.out.println(a);}
  |
  <EOL>
  |
  <EOF>
    { System.exit(-1); }
}
```

```
double expr():
{
  double a;
  double b;
}
{
  a=term()
  (
    "+" b=expr()  {a += b;}
    |
    "-" b=expr()  {a -= b;}
  )*
  { return a;}
}
```

SimpleCalc1.jj (5)

```
double term():
{
  double a;
  double b;
}
{
  a=unary()
  (
    "*" b=term() {a *= b;}
    |
    "/" b=term() {a /= b;}
  )*
  {return a;}
}
```

```
double unary():
{
  double a;
}
{
  "-" a=elem(){return -a;}
  |
  a=elem() {return a;}
}
```

SimpleCalc1.jj (6)

```
double elem():
{
    Token t;
    double a;
}
{
    t=<NUMBER>
                    {return Double.parseDouble(t.toString()); }
    |
    "(" a=expr() ")"    {return a;}
}
```

We have given you these great tools for making your own language

- But here is why you should hesitate
  - Too many languages!
  - If you leave a project, is your personal language documented?
  - Did you optimize, extend, debug, etc. your personal language as much as C? Java? C++?
    - Unicode compatible?
    - Multi-threaded?
    - Optimized?
    - Debugging tools?

Two common alternatives

- XML (eXtendible Markup Language)
- Example:

```
<person>
    <age>12</age>
    <name>Danielle</name>
</person>
```

- Advantanges:
  - Interoperable
  - Open
  - Self-documenting
- A ***document*** exchange format

- JSON (Javascript Object Notation)
- Example:

```
myJSON =
{"age"  : 12,
 "name" : "Danielle"}
```

- Advantages:
  - More concise
  - More readable
  - Some say as interoperable and/or open
- A ***data*** exchange format

PHP approach to XML parsing (1)

- Creates a map (of maps (of maps))
  - Access single value:
    - container->attribute
  - Access multiple values:
    - container->attribute[0], container->attribute[1], ...
- Useful functionality:
  - Class
    - SimpleXMLElement
  - Constructor call
    - new SimpleXMLElement(String toParse)

PHP approach to XML parsing (2)

```
<?php
// example.php
$xmlstr = <<<XML
<?xml version='1.0' standalone='yes'?>
<movies>
 <movie>
  <title>PHP: Behind the Parser</title>
  <characters>
   <character>
    <name>Ms. Coder</name>
    <actor>Onlivia Actora</actor>
   </character>
   <character>
    <name>Mr. Coder</name>
    <actor>El Act&#211;r</actor>
   </character>
  </characters>
  <plot>
   So, this language. It's like, a programming
language. Or is it a
   scripting language? All is revealed in this
thrilling horror spoof
   of a documentary.
  </plot>
  <great-lines>
   <line>PHP solves all my web
problems</line>
  </great-lines>
  <rating type="thumbs">7</rating>
  <rating type="stars">5</rating>
 </movie>
</movies>
XML;
?>
```

PHP approach to XML parsing (3)

```
<?php
  // page.php
include 'example.php';

$movies = new SimpleXMLElement($xmlstr);

echo "Plot:";
echo $movies->movie[0]->plot;
echo "\n";
echo "Characters:\n";
echo $movies->movie[0]->characters->character[0]->name . " (" .
     $movies->movie[0]->characters->character[0]->actor . ")\n";
echo $movies->movie[0]->characters->character[1]->name . " (" .
     $movies->movie[0]->characters->character[1]->actor . ")\n";
echo "\n";
?>
```

PHP approach to XML parsing (4)

```
$ php ../PHP_XMLReader/page.php
Plot:
   So, this language. It's like, a
programming language. Or is it a
   scripting language? All is revealed in
this thrilling horror spoof
   of a documentary.

Characters:
Ms. Coder (Onlivia Actora)
Mr. Coder (El ActÓr)
```

libxml2 approach (1)

- More low-level
  - Have to worry about allocating memory
- Useful functions:
  - xmlDocPtr xmlParseFile   (const char * filename);
  - xmlDocPtr xmlParseMemory (const char * buffer, int size);
  - xmlNodePtr xmlDocGetRootElement (const xmlDoc * doc);
  - xmlChar *   xmlNodeGetContent   (const xmlNode * cur);
  - xmlChar *   xmlGetProp (const xmlNode * node, const xmlChar * name);
  - void   xmlFreeDoc (xmlDocPtr cur);
  - void   xmlCleanupParser   (void);
- Annoyances:
  - Don’t forget xmlFreeDoc() and xmlCleanupParser()
  - Uses char type xmlChar* (char* interpreted as UTF-8?)
  -

libxml2 approach (2)

```
<!-- display.xml -->
<som from="httpd" to="SessionInterface">
  <command>display</command>
  <accountId>accountId</accountId>
<item>firstNode</item> </som>

<?xml version="1.0" standalone="yes"?>
<!-- display.xml -->
<som sessionId="sessionIdNum">
  <success action="new session" accountId="accountId"/>
  <node>firstNode</node>
  <siIpAddr>IP address to session interface</siIpAddr>
  <siPort>port number of session interface</siPort>
  <msg>
    Welcoming message text
  </msg>
</som>
```

libxml2 approach (3)

```
/**
 * section: Tree
 * synopsis: Navigates a tree to print element names
 * purpose: Parse a file to a tree, use xmlDocGetRootElement() to
 *          get the root element, then walk the document and print
 *          all the element name in document order.
 * usage: tree1 filename_or_URL
 * test: tree1 test2.xml > tree1.tmp && diff tree1.tmp $(srcdir)/tree1.res
 * author: Dodji Seketeli
 * copy: see Copyright for the status of this software.
 * Modified by Joe Phillips, 2015
 */
```

libxml2 approach (4)

```
#include <string.h>
#include <libxml/parser.h>
#include <libxml/tree.h>

#ifdef LIBXML_TREE_ENABLED

/*
 *To compile this file using gcc you can type
 *gcc `xml2-config --cflags --libs` -o xmlexample libxml2-example.c
 * gcc tree1.c -o tree1 -lxml2
 */

/**
 * print_element_names:
 * @a_node: the initial xml node to consider.
 *
 * Prints the names of the all the xml elements
 * that are siblings or children of a given xml node.
 */
```

libxml2 approach (5)

```
static void
print_element_names(xmlNode * a_node, int level)
{
    xmlNode *cur_node = NULL;

    for (cur_node = a_node; cur_node; cur_node = cur_node->next) {
        if (cur_node->type == XML_ELEMENT_NODE) {
            printf("(%d) node type: Element, name: %s\n",
level,cur_node->name);
    xmlChar* attrValPtr;

    if ((attrValPtr = xmlGetProp(cur_node, "sessionId")) != NULL)
      printf("  sessionId = %s\n",attrValPtr);

    if ((attrValPtr = xmlGetProp(cur_node, "action")) != NULL)
      printf("  action = %s\n",attrValPtr);

    if ((attrValPtr = xmlGetProp(cur_node, "accountId")) != NULL)
      printf("  accountId = %s\n",attrValPtr);

    if ((attrValPtr = xmlGetProp(cur_node, "from")) != NULL)
      printf("  from = %s\n",attrValPtr);

    if ((attrValPtr = xmlGetProp(cur_node, "to")) != NULL)
      printf("  to = %s\n",attrValPtr);
```

```
    xmlChar* nodeTextPtr;

    if  ( xmlNodeIsText(cur_node->xmlChildrenNode) &&
  ((nodeTextPtr = xmlNodeGetContent(cur_node->xmlChildrenNode))
   != NULL
  )
)
      {
xmlChar* run;

for  (run = nodeTextPtr; *run != '\0';  run++)
  if  (!isspace(*run) )
    break;

if  (*run == '\0')
  printf("  value = <empty spaces>\n");
else
  printf("  value = %s\n",nodeTextPtr);

xmlFree(nodeTextPtr);
      }
        }

print_element_names(cur_node->children,level+1);
    }
}
```

libxml2 approach (6)

```
/**
 * Simple example to parse a file called "file.xml",
 * walk down the DOM, and print the name of the
 * xml elements nodes.
 */
int main(int argc, char **argv)
{
    printf("sizeof(xmlChar) == %d\n",sizeof(xmlChar));
    xmlDoc *doc = NULL;
    xmlNode *root_element = NULL;

    if (argc != 2)
        return(1);

    /*
     * this initialize the library and check potential ABI mismatches
     * between the version it was compiled for and the actual shared
     * library used.
     */
    LIBXML_TEST_VERSION
```

libxml2 approach (7)

```
    /*parse the file and get the DOM */
    doc = xmlReadFile(argv[1], NULL, 0);

    if (doc == NULL) {
        printf("error: could not parse file %s\n", argv[1]);
    }

    /*Get the root element node */
    root_element = xmlDocGetRootElement(doc);

    print_element_names(root_element,0);

    /*free the document */
    xmlFreeDoc(doc);

    /*
     *Free the global variables that may
     *have been allocated by the parser.
     */
    xmlCleanupParser();

    return 0;
}
```

libxml2 – A better approach
Define a path class: