# The Bertha™ Parser Generator for Ordered Context-Free Grammars

## Version 1.04

Ziemowit Laski

laski@ics.uci.edu
http://www.ics.uci.edu/~laski

User Manual

Department of Information and Computer Science
University of California, Irvine
Irvine, CA 92697-3425, USA

Created: April 26, 1999
Updated: August 5, 1999

**Abstract**

The Bertha™ parser generator for ordered context-free grammars (OCFGs) offers an easy-to-use and type-safe alternative to bottom-up parser generators such as `bison` and `yacc`. It is designed to handle the LALRP(1) (for "LALR(1) with precedence") subset of OCFGs. As its name indicates, LALRP(1) is analogous to LALR(1) in that it attempts to construct an LR(0) deterministic pushdown automaton (DPDA) while employing 1 token of lookahead. However, due to the established precedence relations among productions and derivation sequences, many ambiguities that arise during LALR(1) or even LR(1) parser construction are eliminated. At the same time, all valid LALR(1) grammars are automatically valid under the LALRP(1) regime as well – even though precedence and associativity are missing, they are not needed to resolve any conflicts. Bertha utilizes an input syntax and semantics differing only slightly from that of the Java™ programming language. Because of this, Bertha grammar specifications leverage the benefits of programming language features such as encapsulation, inheritance and type-safety.

# 1 Modification History

| Version | Date | New Features |
|---|---|---|
| 1.00 | 26-Apr-1999 | Initial public release. |
| 1.01 | 10-May-1999 | Maintenance release only. |
| 1.02 | 23-Jun-1999 | Changed namespace from `zll.bertha` to `zll.bertha1`. Barring any last-minute bug fixes, this will be the final release supporting only linear orderings of productions; future versions (with namespace `zll.bertha2` will allow arbitrary partial orders. |
| 1.03 | 06-Jul-1999 | *Massive* last-minute bug fixes! In addition, `symbol`s may now use the `implements` clause, in addition to `extends`. The run-time scanner has been enhanced to normalize line terminations in text files: CR, LF and CR+LF now all get converted to LF before being passed on to Bertha. |
| 1.04 | 05-Aug-1999 | All `reduce` methods may now employ `Java throws` clauses; this feature may be used with the new `ParseException`, an inner class of `zll.bertha1.run.Grammar` automatically visible within each Bertha `grammar`, or with any subclass of `zll.bertha1.run.ParseError`. In addition to `int get()`, `boolean get(String)` and `int peek()`, a `boolean peek(String)` method is now available to test for the presence of an entire string in the lookahead. The string itself remains to be read, of course, via one of the `get(...)` methods. |

# 2 An Example Grammar

The following ordered context-free grammar, describing the input syntax of a simple calculator, is used as an illustrative example in this section.

$$
\begin{array}{rlll}
S & \longrightarrow E & E & \longrightarrow num \\
num & \longrightarrow \text{(terminal)} & E & \longrightarrow \text{“(” } E \text{“)”} \\
add\_op & \longrightarrow \text{“+”} & E & \longrightarrow add\_op\ E \\
add\_op & \longrightarrow \text{“-”} & E & \longrightarrow E \text{ “↑” } E \quad \text{(right)} \\
mul\_op & \longrightarrow \text{“*”} & E & \longrightarrow E\ mul\_op\ E \quad \text{(left)} \\
mul\_op & \longrightarrow \text{“/”} & E & \longrightarrow E\ add\_op\ E \quad \text{(left)}
\end{array}
$$

## 2.1 Example Bertha(tm) Input Grammar

The following listing contains a Bertha grammar specification, stored in file `lc.bertha`, for the ordered context-free grammar (OCFG) presented above. This specification will work with the 1.x releases of Bertha. Future versions of Bertha will support arbitrary partial orders and will require a different, slightly more complicated input syntax.

```
// lc.bertha
// This file defines a simple grammar for a line calculator
import java.io.*;
import zll.bertha1.run.*;

grammar lc {   // must match lc.bertha; will create lc.java
   int result;  // attribute of the grammar object

   // the program processes what is on the command line
   public static void main(String args[]) throws ParseError {
      lc calc = new lc(new java.io.StringReader(ArgString(args)));
      calc.Parse();
      System.out.println("The result is " + calc.result);
   }
   symbol S { reduce(E expr) { result = expr.val; }}
   symbol E { int val; // attribute for nonterminal E
      reduce(num n) { val = Integer.decode(n.val).intValue(); }
      reduce("(", E s, ")") { val = s.val; }
      reduce(add_op o, E s) { switch(o.oper) {
            case '+': val = s.val; break;
            case '-': val = - s.val; break;
      }}
      rightassoc reduce(E base, "^", E exponent) {
         val = (int)Math.pow(base.val, exponent.val);
      }
      leftassoc reduce(E s1, mul_op o, E s2) { switch(o.oper) {
            case '*': val = s1.val * s2.val; break;
            case '/': val = s1.val / s2.val; break;
      }}
      leftassoc reduce(E s1, add_op o, E s2) { switch(o.oper) {
            case '+': val = s1.val + s2.val; break;
            case '-': val = s1.val - s2.val; break;
      }}
   }
   class char_sym { char oper; }  // base class
   symbol add_op extends char_sym {
      reduce("+") { oper = '+'; }
      reduce("-") { oper = '-'; }
   }
   symbol mul_op extends char_sym {
      reduce("*") { oper = '*'; }
      reduce("/") { oper = '/'; }
   }
   symbol num { String val = new String();
      scan() {
         int c = peek();
         if(c < '0' || c > '9') return false;
         do {
            val += ((char)get()); c = peek();
         }
         while(c >= '0' && c <= '9');
         return true;
      }
   }
} // end of grammar lc
```

## 2.2 Example Bertha(tm) Output

The following listing depicts the Java output, generated by Bertha, corresponding to the foregoing `lc.bertha` input. This output, stored in a file named `lc.java`, may then be compiled by any Java compiler that supports Java 1.1 language features such as inner classes.

```
/*************************************************************
 *                                                           *
 * File generated by the Bertha(tm) parser generator (Ver 1.04)   *
 * (c) 1998-1999 by Regents of the University of California       *
 * Created by Ziemowit Laski (University of California, Irvine)   *
 *                                                           *
 * Wed Jul 28 21:03:09 PDT 1999                    DO NOT EDIT!  *
 *                                                           *
 *************************************************************/

// lc.bertha
// This file defines a simple grammar for a line calculator
import java.io.*;
import zll.bertha1.run.*;

public class lc extends zll.bertha1.run.Grammar {
   int result;  // attribute of the grammar object

   // the program processes what is on the command line
   public static void main(String args[]) throws ParseError {
      lc calc = new lc(new java.io.StringReader(ArgString(args)));
      calc.Parse();
      System.out.println("The result is " + calc.result);
   }
   public class S implements zll.bertha1.run.Symbol {
      public void reduce0(E expr) { result = expr.val; }
      //*** symbol reduction routine
      public boolean process(byte s) {
         switch(s) {
            case 0: reduce0((E)st(0)); return true;
            default: return false;
         }
      }
   }
   public class E implements zll.bertha1.run.Symbol { int val;
      public void reduce0(num n) { val = Integer.decode(n.val).intValue(); }
      public void reduce1(zll.bertha1.run.Grammar.Literal PARM1, E s,
            zll.bertha1.run.Grammar.Literal PARM3) { val = s.val; }
      public void reduce2(add_op o, E s) { switch(o.oper) {
            case '+': val = s.val; break;
            case '-': val = - s.val; break;
      }}
      public void reduce3(E base, zll.bertha1.run.Grammar.Literal PARM2,
            E exponent) /* rightassoc */ {
         val = (int)Math.pow(base.val, exponent.val);
      }
      public void reduce4(E s1, mul_op o, E s2) /* leftassoc */ {
         switch(o.oper) {
            case '*': val = s1.val * s2.val; break;
            case '/': val = s1.val / s2.val; break;
      }}
      public void reduce5(E s1, add_op o, E s2) /* leftassoc */ {
         switch(o.oper) {
            case '+': val = s1.val + s2.val; break;
            case '-': val = s1.val - s2.val; break;
      }}
```

```
   //*** symbol reduction routine
   public boolean process(byte s) {
      switch(s) {
         case 0: reduce0((num)st(0)); return true;
         case 1: reduce1((zll.bertha1.run.Grammar.Literal)st(0), (E)st(1),
                    (zll.bertha1.run.Grammar.Literal)st(2)); return true;
         case 2: reduce2((add_op)st(0), (E)st(1)); return true;
         case 3: reduce3((E)st(0), (zll.bertha1.run.Grammar.Literal)st(1),
                    (E)st(2)); return true;
         case 4: reduce4((E)st(0), (mul_op)st(1), (E)st(2)); return true;
         case 5: reduce5((E)st(0), (add_op)st(1), (E)st(2)); return true;
         default: return false;
      }
   }
}
class char_sym { char oper; }  // base class
public class add_op extends char_sym implements zll.bertha1.run.Symbol {
   public void reduce0(zll.bertha1.run.Grammar.Literal PARM1) { oper = '+'; }
   public void reduce1(zll.bertha1.run.Grammar.Literal PARM1) { oper = '-'; }
   //*** symbol reduction routine
   public boolean process(byte s) {
      switch(s) {
         case 0: reduce0((zll.bertha1.run.Grammar.Literal)st(0)); return true;
         case 1: reduce1((zll.bertha1.run.Grammar.Literal)st(0)); return true;
         default: return false;
      }
   }
}
public class mul_op extends char_sym implements zll.bertha1.run.Symbol {
   public void reduce0(zll.bertha1.run.Grammar.Literal PARM1) { oper = '*'; }
   public void reduce1(zll.bertha1.run.Grammar.Literal PARM1) { oper = '/'; }
   //*** symbol reduction routine
   public boolean process(byte s) {
      switch(s) {
         case 0: reduce0((zll.bertha1.run.Grammar.Literal)st(0)); return true;
         case 1: reduce1((zll.bertha1.run.Grammar.Literal)st(0)); return true;
         default: return false;
      }
   }
}
public class num implements zll.bertha1.run.Symbol { String val = new String();
   public boolean process(byte PARM) {
      int c = peek();
      if(c < '0' || c > '9') return false;
      do {
         val += ((char)get()); c = peek();
      }
      while(c >= '0' && c <= '9');
      return true;
   }
}
//*** symbol creation routine
public zll.bertha1.run.Symbol create(short s) {
   switch(s) {
      case 5: return new S();
      case 7: return new E();
      case 8: return new num();
      case 10: return new zll.bertha1.run.Grammar.Literal("(");
      case 12: return new zll.bertha1.run.Grammar.Literal(")");
      case 15: return new add_op();
      case 16: return new zll.bertha1.run.Grammar.Literal("^");
      case 19: return new mul_op();
      case 20: return new zll.bertha1.run.Grammar.Literal("+");
      case 22: return new zll.bertha1.run.Grammar.Literal("-");
```

```
            case 24: return new zll.bertha1.run.Grammar.Literal("*");
            case 26: return new zll.bertha1.run.Grammar.Literal("/");
            default: return null;
         }
      }
      //*** parse tables
      final short action[] = {
         8, 1, 51, 10, 1, 54, 20, 1, 96, 22, 1, 111, 5, 2, 24, 15, 2, 75,
         7, 2, 27, 0, 10, 0, 0, 9, 0, 16, 1, 126, 24, 1, 189, 26, 1, 204,
         20, 1, 96, 22, 1, 111, 19, 2, 147, 15, 2, 168, 0, 4, 12, 0, 4, 15,
         8, 1, 51, 10, 1, 54, 20, 1, 96, 22, 1, 111, 15, 2, 75, 7, 2, 219,
         0, 10, 0, 8, 1, 51, 10, 1, 54, 20, 1, 96, 22, 1, 111, 15, 2, 75,
         7, 2, 246, 0, 10, 0, 8, 4, 18, 10, 4, 18, 20, 4, 18, 22, 4, 18,
         0, 10, 0, 8, 4, 21, 10, 4, 21, 20, 4, 21, 22, 4, 21, 0, 10, 0,
         8, 1, 51, 10, 1, 54, 20, 1, 96, 22, 1, 111, 15, 2, 75, 7, 2, 249,
         0, 10, 0, 8, 1, 51, 10, 1, 54, 20, 1, 96, 22, 1, 111, 15, 2, 75,
         7, 2, 270, 0, 10, 0, 8, 1, 51, 10, 1, 54, 20, 1, 96, 22, 1, 111,
         15, 2, 75, 7, 2, 291, 0, 10, 0, 8, 4, 24, 10, 4, 24, 20, 4, 24,
         22, 4, 24, 0, 10, 0, 8, 4, 27, 10, 4, 27, 20, 4, 27, 22, 4, 27,
         0, 10, 0, 12, 1, 315, 16, 1, 126, 24, 1, 189, 26, 1, 204, 20, 1, 96,
         22, 1, 111, 19, 2, 147, 15, 2, 168, 0, 10, 0, 0, 4, 0, 16, 1, 126,
         24, 4, 3, 26, 4, 3, 20, 4, 3, 22, 4, 3, 12, 4, 3, 0, 4, 3,
         16, 1, 126, 24, 4, 6, 26, 4, 6, 20, 4, 6, 22, 4, 6, 12, 4, 6,
         0, 4, 6, 16, 1, 126, 24, 1, 189, 26, 1, 204, 19, 2, 147, 20, 4, 9,
         22, 4, 9, 12, 4, 9, 0, 4, 9, 0, 4, 30
      };
      final short prod[] = {
         7, 2, 2, 7, 3, 3, 7, 4, 3, 7, 5, 3, 5, 0, 1, 7, 0, 1,
         15, 0, 1, 15, 1, 1, 19, 0, 1, 19, 1, 1, 7, 1, 3
      };
      //*** parse table access
      public short[] action() { return this.action; }
      public short[] prod() { return this.prod; }
      //*** constructor(s)
      public lc(java.io.Reader input, String name, int reclength) {
         super(input, name, reclength);
         this.inner = null;
      }
      public lc(java.io.Reader input, int reclength) {
         this(input, "<input>", reclength);
      }
      public lc(java.io.Reader input, String name) {
         this(input, name, zll.bertha1.run.Scanner.CRLF);
      }
      public lc(java.io.Reader input) {
         this(input, "<input>", zll.bertha1.run.Scanner.CRLF);
      }
      public lc(String name, int reclength) throws java.io.FileNotFoundException {
         this(new java.io.FileReader(name), name, reclength);
      }
      public lc(String name) throws java.io.FileNotFoundException {
         this(name, zll.bertha1.run.Scanner.CRLF);
      }
      public lc(String name, String encoding) throws java.io.FileNotFoundException,
            java.io.UnsupportedEncodingException {
         this(new java.io.InputStreamReader(new java.io.FileInputStream(name), encoding),
               name, zll.bertha1.run.Scanner.CRLF);
      }
   } // end of grammar lc
```