```
PARSER_BEGIN(Parser)
/** ID lister. */
import AST.*;
import other.Environment;
import java.util.LinkedList;
import java.util.List;

public class Parser
{
  /** Main entry point. */
  public static void main(String args [])
  {
    Parser parser = new Parser(System.in);
    ASTNode exp;
    while (true)
    {
      try
      {
        exp = parser.Start();
        System.out.println(exp.eval(new Environment()));
      }
      catch (Exception e)
      {
        System.out.println("Syntax Error!");
        parser.ReInit(System.in);
      }
    }
  }
}

PARSER_END(Parser)

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

TOKEN :
{
  < TRUE : "true" >
|
  < FALSE : "false" >
|
  < ELSE : "else" >
|
  < THEN : "then" >
|
  < IF : "if" >
|
  < DO : "do" >
|
  < WHILE : "while" >
|
  < LOET : "<=" >
|
  < GOET : ">=" >
|
  < LT : "<" >
|
```

```
  < GT : ">" >
|
  < COMP_EQUALS : "==" >
|
  < ASSIGN : ":=" >
|
  < DESREF : "!" >
|
  < NEW : "new" >
|
  < COMMA : "," >
|
  < SEMICOL : ";" >
|
  < FUN : "fun" >
|
  < ARROW : "=>" >
|
  < LET : "let" >
|
  < IN : "in" >
|
  < END : "end" >
|
  < EQUALS : "=" >
|
  < Id : [ "a"-"z", "A"-"Z" ] ([ "a"-"z", "A"-"Z", "0"-"9" ])* >
|
  < Num : ([ "0"-"9" ])+ >
|
  < PLUS : "+" >
|
  < MINUS : "-" >
|
  < TIMES : "*" >
|
  < DIV : "/" >
|
  < LPAR : "(" >
|
  < RPAR : ")" >
|
  < EL : ";;" >
}

ASTNode Start() :
{
  ASTNode t;
}
{
  t = Seq() < EL >
  {
    return t;
  }
}

ASTNode Seq() :
{
  ASTNode e1, e2;
}
{
  e1 = Comp()
```

```
125   (
126     < SEMICOL >
127     e2 = Seq()
128     {
129       e1 = new ASTSeq(e1, e2);
130     }
131   )*
132   {
133     return e1;
134   }
135 }
136
137 ASTNode Comp() :
138 {
139   Token op;
140   ASTNode e1, e2;
141 }
142 {
143   e1 = Exp()
144   (
145     (
146       op = < LOET >
147     | op = < GOET >
148     | op = < LT >
149     | op = < GT >
150     | op = < COMP_EQUALS >
151     )
152     e2 = Exp()
153     {
154       switch (op.kind)
155       {
156         case LOET :
157         e1 = new ASTLOET(e1, e2);
158         break;
159         case GOET :
160         e1 = new ASTGOET(e1, e2);
161         break;
162         case LT :
163         e1 = new ASTLT(e1, e2);
164         break;
165         case GT :
166         e1 = new ASTGT(e1, e2);
167         break;
168         case COMP_EQUALS :
169         e1 = new ASTEq(e1, e2);
170         break;
171       }
172     }
173   )?
174   {
175     return e1;
176   }
177 }
178
179 ASTNode Exp() :
180 {
181   Token op;
182   ASTNode t1, t2;
183 }
184 {
185   t1 = Term()
186   (
```

```
187     (
188       op = < PLUS >
189     | op = < MINUS >
190     )
191     t2 = Exp()
192     {
193       if (op.kind == PLUS)
194       t1 = new ASTPlus(t1, t2);
195       else t1 = new ASTSub(t1, t2);
196     }
197   )*
198   {
199     return t1;
200   }
201 }
202
203 List < String > ParamList() :
204 {
205   List < String > params = new LinkedList < String > ();
206   Token onePar, multiplePar;
207 }
208 {
209   (
210     onePar = < Id >
211     {
212       params.add(onePar.image);
213     }
214     (
215       < COMMA > multiplePar = < Id >
216       {
217         params.add(multiplePar.image);
218       }
219     )*
220   )?
221   {
222     return params;
223   }
224 }
225
226 List < ASTNode > ArgsList() :
227 {
228   List < ASTNode > args = new LinkedList < ASTNode > ();
229   ASTNode oneArg, multipleArgs;
230 }
231 {
232   (
233     oneArg = Seq()
234     {
235       args.add(oneArg);
236     }
237     (
238       < COMMA > multipleArgs = Seq()
239       {
240         args.add(multipleArgs);
241       }
242     )*
243   )
244   {
245     return args;
246   }
247 }
248
```

```
249 ASTNode Term() :
250 {
251   Token op;
252   ASTNode f, t;
253   List < ASTNode > args;
254 }
255 {
256   f = Fact()
257   (
258     < LPAR > args = ArgsList() < RPAR >
259     {
260       f = new ASTApply(f, args);
261     }
262   )?
263   (
264     (
265       < ASSIGN > t = Comp()
266       {
267         f = new ASTAssign(f, t);
268       }
269     )
270   |
271     (
272       (
273         op = < TIMES >
274       | op = < DIV >
275       )
276       t = Term()
277       {
278         if (op.kind == TIMES)
279         f = new ASTMul(f, t);
280         else f = new ASTDiv(f, t);
281       }
282     )*
283   )
284   {
285     return f;
286   }
287 }
288
289 ASTNode Fact() :
290 {
291   Token n;
292   ASTNode t;
293 }
294 {
295   (
296     n = < Id >
297     {
298       t = new ASTId(n.image);
299     }
300   |
301     n = < Num >
302     {
303       t = new ASTNum(Integer.parseInt(n.image));
304     }
305   |
306     (
307       n = < TRUE >
308     | n = < FALSE >
309     )
310     {
```

```
311       t = new ASTBool(Boolean.parseBoolean(n.image));
312     }
313   | t = LetBuild()
314   | t = FunBuild()
315   | < LPAR > t = Seq() < RPAR >
316   | t = NewBuild()
317   | t = DesrefBuild()
318   | t = WhileBuild()
319   | t = IfBuild()
320   )
321   {
322     return t;
323   }
324 }
325
326 ASTNode IfBuild() :
327 {
328   ASTNode cond, e1, e2;
329 }
330 {
331   (
332     < IF > cond = Seq() < THEN > e1 = Seq() < ELSE > e2 = Seq() < END >
333   )
334   {
335     return new ASTIf(cond, e1, e2);
336   }
337 }
338
339 ASTNode NewBuild() :
340 {
341   ASTNode f;
342 }
343 {
344   < NEW > f = Fact()
345   {
346     return new ASTNew(f);
347   }
348 }
349
350 ASTNode DesrefBuild() :
351 {
352   ASTNode f;
353 }
354 {
355   < DESREF > f = Fact()
356   {
357     return new ASTDesref(f);
358   }
359 }
360
361 ASTNode LetBuild() :
362 {
363   List < String > ids = new LinkedList < String > ();
364   List < ASTNode > exps = new LinkedList < ASTNode > ();
365   Token id;
366   ASTNode exp_init, exp_body;
367 }
368 {
369   (
370     < LET >
371     (
372       (id = < Id >) < EQUALS >
```

```
373       exp_init = Seq()
374       {
375         ids.add(id.image);
376         exps.add(exp_init);
377       }
378     )+
379     < IN >
380     (
381       exp_body = Seq()
382     )
383     < END >
384   )
385   {
386     return new ASTLet(ids, exps, exp_body);
387   }
388 }
389
390 ASTNode WhileBuild() :
391 {
392   ASTNode cond, e;
393 }
394 {
395   < WHILE > cond = Seq() < DO > e = Seq() < END >
396   {
397     return new ASTWhile(cond, e);
398   }
399 }
400
401 ASTNode FunBuild() :
402 {
403   List < String > param;
404   ASTNode exp;
405 }
406 {
407   (
408     < FUN >
409     (
410       param = ParamList()
411     )
412     < ARROW >
413     (
414       exp = Seq()
415     )
416     < END >
417   )
418   {
419     return new ASTFun(param, exp);
420   }
421 }
422
```