

# Parser.java

```

1 /* Generated By:JavaCC: Do not edit this line. Parser.java */
2 import AST.*;
3
4
5
6
7
8 public class Parser implements ParserConstants {
9     /** Main entry point. */
10    public static void main(String args [])
11    {
12        Parser parser = new Parser(System.in);
13        ASTNode exp;
14        while (true)
15        {
16            try
17            {
18                exp = parser.Start();
19                System.out.println(exp.eval(new Environment()));
20            }
21            catch (Exception e)
22            {
23                System.out.println("Syntax Error!");
24                parser.ReInit(System.in);
25            }
26        }
27    }
28
29    static final public ASTNode Start() throws ParseException {
30        ASTNode t;
31        t = Seq();
32        jj_consume_token(EL);
33        {if (true) return t;}
34        throw new Error("Missing return statement in function");
35    }
36
37    static final public ASTNode Seq() throws ParseException {
38        ASTNode e1, e2;
39        e1 = Comp();
40        label_1:
41        while (true) {
42            switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
43                case SEMICOL:
44                    ;
45                    break;
46                default:
47                    jj_la1[0] = jj_gen;
48                    break label_1;
49            }
50            jj_consume_token(SEMICOL);
51            e2 = Seq();
52            e1 = new ASTSeq(e1, e2);
53        }
54        {if (true) return e1;}
55        throw new Error("Missing return statement in function");
56    }
57
58    static final public ASTNode Comp() throws ParseException {
59        Token op;
60        ASTNode e1, e2;
61        e1 = Exp();
62        switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
63            case LOET:
64            case GOET:
65            case LT:
66            case GT:

```

```

67     case COMP_EQUALS:
68         switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
69             case LOET:
70                 op = jj_consume_token(LOET);
71                 break;
72             case GOET:
73                 op = jj_consume_token(GOET);
74                 break;
75             case LT:
76                 op = jj_consume_token(LT);
77                 break;
78             case GT:
79                 op = jj_consume_token(GT);
80                 break;
81             case COMP_EQUALS:
82                 op = jj_consume_token(COMP_EQUALS);
83                 break;
84             default:
85                 jj_la1[1] = jj_gen;
86                 jj_consume_token(-1);
87                 throw new ParseException();
88         }
89         e2 = Exp();
90         switch (op.kind)
91         {
92             case LOET :
93                 e1 = new ASTLOET(e1, e2);
94                 break;
95             case GOET :
96                 e1 = new ASTGOET(e1, e2);
97                 break;
98             case LT :
99                 e1 = new ASTLT(e1, e2);
100             break;
101             case GT :
102                 e1 = new ASTGT(e1, e2);
103                 break;
104             case COMP_EQUALS :
105                 e1 = new ASTEq(e1, e2);
106                 break;
107         }
108         break;
109     default:
110         jj_la1[2] = jj_gen;
111         ;
112     }
113     {if (true) return e1;}
114     throw new Error("Missing return statement in function");
115 }
116
117 static final public ASTNode Exp() throws ParseException {
118     Token op;
119     ASTNode t1, t2;
120     t1 = Term();
121     label_2:
122     while (true) {
123         switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
124             case PLUS:
125             case MINUS:
126                 ;
127                 break;
128             default:

```

```

129     jj_La1[3] = jj_gen;
130     break label_2;
131 }
132 switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
133 case PLUS:
134     op = jj_consume_token(PLUS);
135     break;
136 case MINUS:
137     op = jj_consume_token(MINUS);
138     break;
139 default:
140     jj_La1[4] = jj_gen;
141     jj_consume_token(-1);
142     throw new ParseException();
143 }
144 t2 = Exp();
145 if (op.kind == PLUS)
146     t1 = new ASTPlus(t1, t2);
147 else t1 = new ASTSub(t1, t2);
148 }
149 {if (true) return t1;}
150 throw new Error("Missing return statement in function");
151 }
152
153 static final public List < String > ParamList() throws ParseException {
154 List < String > params = new LinkedList < String > ();
155 Token onePar, multiplePar;
156 switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
157 case Id:
158     onePar = jj_consume_token(Id);
159     params.add(onePar.image);
160     label_3:
161     while (true) {
162         switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
163         case COMMA:
164             ;
165             break;
166         default:
167             jj_La1[5] = jj_gen;
168             break label_3;
169         }
170         jj_consume_token(COMMA);
171         multiplePar = jj_consume_token(Id);
172         params.add(multiplePar.image);
173     }
174     break;
175 default:
176     jj_La1[6] = jj_gen;
177     ;
178 }
179 {if (true) return params;}
180 throw new Error("Missing return statement in function");
181 }
182
183 static final public List < ASTNode > ArgsList() throws ParseException {
184 List < ASTNode > args = new LinkedList < ASTNode > ();
185 ASTNode oneArg, multipleArgs;
186 oneArg = Seq();
187 args.add(oneArg);
188 label_4:
189 while (true) {
190     switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {

```

```

191     case COMMA:
192         ;
193         break;
194     default:
195         jj_la1[7] = jj_gen;
196         break label_4;
197     }
198     jj_consume_token(COMMA);
199     multipleArgs = Seq();
200     args.add(multipleArgs);
201 }
202 {if (true) return args;}
203 throw new Error("Missing return statement in function");
204 }
205
206 static final public ASTNode Term() throws ParseException {
207     Token op;
208     ASTNode f, t;
209     List < ASTNode > args;
210     f = Fact();
211     switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
212     case LPAR:
213         jj_consume_token(LPAR);
214         args = ArgsList();
215         jj_consume_token(RPAR);
216         f = new ASTApply(f, args);
217         break;
218     default:
219         jj_la1[8] = jj_gen;
220         ;
221     }
222     switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
223     case ASSIGN:
224         jj_consume_token(ASSIGN);
225         t = Comp();
226         f = new ASTAssign(f, t);
227         break;
228     default:
229         jj_la1[11] = jj_gen;
230         label_5:
231         while (true) {
232             switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
233             case TIMES:
234             case DIV:
235                 ;
236                 break;
237             default:
238                 jj_la1[9] = jj_gen;
239                 break label_5;
240             }
241             switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
242             case TIMES:
243                 op = jj_consume_token(TIMES);
244                 break;
245             case DIV:
246                 op = jj_consume_token(DIV);
247                 break;
248             default:
249                 jj_la1[10] = jj_gen;
250                 jj_consume_token(-1);
251                 throw new ParseException();
252             }

```

```

253     t = Term();
254     if (op.kind == TIMES)
255         f = new ASTMul(f, t);
256     else f = new ASTDiv(f, t);
257 }
258 }
259 {if (true) return f;}
260 throw new Error("Missing return statement in function");
261 }
262
263 static final public ASTNode Fact() throws ParseException {
264     Token n;
265     ASTNode t;
266     switch ((jj_ntk== -1)?jj_ntk():jj_ntk) {
267     case Id:
268         n = jj_consume_token(Id);
269         t = new ASTId(n.image);
270         break;
271     case Num:
272         n = jj_consume_token(Num);
273         t = new ASTNum(Integer.parseInt(n.image));
274         break;
275     case TRUE:
276     case FALSE:
277         switch ((jj_ntk== -1)?jj_ntk():jj_ntk) {
278         case TRUE:
279             n = jj_consume_token(TRUE);
280             break;
281         case FALSE:
282             n = jj_consume_token(FALSE);
283             break;
284         default:
285             jj_la1[12] = jj_gen;
286             jj_consume_token(-1);
287             throw new ParseException();
288         }
289         t = new ASTBool(Boolean.parseBoolean(n.image));
290         break;
291     case LET:
292         t = LetBuild();
293         break;
294     case FUN:
295         t = FunBuild();
296         break;
297     case LPAR:
298         jj_consume_token(LPAR);
299         t = Seq();
300         jj_consume_token(RPAR);
301         break;
302     case NEW:
303         t = NewBuild();
304         break;
305     case DESREF:
306         t = DesrefBuild();
307         break;
308     case WHILE:
309         t = WhileBuild();
310         break;
311     case IF:
312         t = IfBuild();
313         break;
314     default:

```

```

315     jj_la1[13] = jj_gen;
316     jj_consume_token(-1);
317     throw new ParseException();
318 }
319 {if (true) return t;}
320 throw new Error("Missing return statement in function");
321 }
322
323 static final public ASTNode IfBuild() throws ParseException {
324     ASTNode cond, e1, e2;
325     jj_consume_token(IF);
326     cond = Seq();
327     jj_consume_token(THEN);
328     e1 = Seq();
329     jj_consume_token(ELSE);
330     e2 = Seq();
331     jj_consume_token(END);
332     {if (true) return new ASTIf(cond, e1, e2);}
333     throw new Error("Missing return statement in function");
334 }
335
336 static final public ASTNode NewBuild() throws ParseException {
337     ASTNode f;
338     jj_consume_token(NEW);
339     f = Fact();
340     {if (true) return new ASTNew(f);}
341     throw new Error("Missing return statement in function");
342 }
343
344 static final public ASTNode DesrefBuild() throws ParseException {
345     ASTNode f;
346     jj_consume_token(DESREF);
347     f = Fact();
348     {if (true) return new ASTDesref(f);}
349     throw new Error("Missing return statement in function");
350 }
351
352 static final public ASTNode LetBuild() throws ParseException {
353     List < String > ids = new LinkedList < String > ();
354     List < ASTNode > exps = new LinkedList < ASTNode > ();
355     Token id;
356     ASTNode exp_init, exp_body;
357     jj_consume_token(LET);
358     label_6:
359     while (true) {
360         id = jj_consume_token(Id);
361         jj_consume_token(EQUALS);
362         exp_init = Seq();
363         ids.add(id.image);
364         exps.add(exp_init);
365         switch ((jj_ntk== -1)?jj_ntk():jj_ntk) {
366             case Id:
367                 ;
368                 break;
369             default:
370                 jj_la1[14] = jj_gen;
371                 break label_6;
372         }
373     }
374     jj_consume_token(IN);
375     exp_body = Seq();
376     jj_consume_token(END);

```

```

377     {if (true) return new ASTLet(ids, exps, exp_body);}
378     throw new Error("Missing return statement in function");
379 }
380
381 static final public ASTNode WhileBuild() throws ParseException {
382     ASTNode cond, e;
383     jj_consume_token(WHILE);
384     cond = Seq();
385     jj_consume_token(DO);
386     e = Seq();
387     jj_consume_token(END);
388     {if (true) return new ASTWhile(cond, e);}
389     throw new Error("Missing return statement in function");
390 }
391
392 static final public ASTNode FunBuild() throws ParseException {
393     List < String > param;
394     ASTNode exp;
395     jj_consume_token(FUN);
396     param = ParamList();
397     jj_consume_token(ARROW);
398     exp = Seq();
399     jj_consume_token(END);
400     {if (true) return new ASTFun(param, exp);}
401     throw new Error("Missing return statement in function");
402 }
403
404 static private boolean jj_initialized_once = false;
405 /** Generated Token Manager. */
406 static public ParserTokenManager token_source;
407 static SimpleCharStream jj_input_stream;
408 /** Current token. */
409 static public Token token;
410 /** Next token. */
411 static public Token jj_nt;
412 static private int jj_ntk;
413 static private int jj_gen;
414 static final private int[] jj_la1 = new int[15];
415 static private int[] jj_la1_0;
416 static private int[] jj_la1_1;
417 static {
418     jj_la1_init_0();
419     jj_la1_init_1();
420 }
421 private static void jj_la1_init_0() {
422     jj_la1_0 = new int[]
423     {0x200000,0x1f000,0x1f000,0xc0000000,0xc0000000,0x100000,0x10000000,0x100000,0x0,0x0,0x0,0x
424     20000,0x60,0x314c0a60,0x10000000,};
425 }
426 private static void jj_la1_init_1() {
427     jj_la1_1 = new int[] {0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x4,0x3,0x3,0x0,0x0,0x4,0x0,};
428 }
429 /** Constructor with InputStream. */
430 public Parser(java.io.InputStream stream) {
431     this(stream, null);
432 }
433 /** Constructor with InputStream and supplied encoding */
434 public Parser(java.io.InputStream stream, String encoding) {
435     if (jj_initialized_once) {
436         System.out.println("ERROR: Second call to constructor of static parser. ");
437         System.out.println("    You must either use ReInit() or set the JavaCC option

```

# Parser.java

```

    STATIC to false");
437     System.out.println("        during parser generation.");
438     throw new Error();
439 }
440 jj_initialized_once = true;
441 try { jj_input_stream = new SimpleCharStream(stream, encoding, 1, 1); }
    catch(java.io.UnsupportedEncodingException e) { throw new RuntimeException(e); }
442 token_source = new ParserTokenManager(jj_input_stream);
443 token = new Token();
444 jj_ntk = -1;
445 jj_gen = 0;
446 for (int i = 0; i < 15; i++) jj_La1[i] = -1;
447 }
448
449 /** Reinitialise. */
450 static public void ReInit(java.io.InputStream stream) {
451     ReInit(stream, null);
452 }
453 /** Reinitialise. */
454 static public void ReInit(java.io.InputStream stream, String encoding) {
455     try { jj_input_stream.ReInit(stream, encoding, 1, 1); }
    catch(java.io.UnsupportedEncodingException e) { throw new RuntimeException(e); }
456 token_source.ReInit(jj_input_stream);
457 token = new Token();
458 jj_ntk = -1;
459 jj_gen = 0;
460 for (int i = 0; i < 15; i++) jj_La1[i] = -1;
461 }
462
463 /** Constructor. */
464 public Parser(java.io.Reader stream) {
465     if (jj_initialized_once) {
466         System.out.println("ERROR: Second call to constructor of static parser. ");
467         System.out.println("        You must either use ReInit() or set the JavaCC option
    STATIC to false");
468         System.out.println("        during parser generation.");
469         throw new Error();
470     }
471     jj_initialized_once = true;
472     jj_input_stream = new SimpleCharStream(stream, 1, 1);
473     token_source = new ParserTokenManager(jj_input_stream);
474     token = new Token();
475     jj_ntk = -1;
476     jj_gen = 0;
477     for (int i = 0; i < 15; i++) jj_La1[i] = -1;
478 }
479
480 /** Reinitialise. */
481 static public void ReInit(java.io.Reader stream) {
482     jj_input_stream.ReInit(stream, 1, 1);
483     token_source.ReInit(jj_input_stream);
484     token = new Token();
485     jj_ntk = -1;
486     jj_gen = 0;
487     for (int i = 0; i < 15; i++) jj_La1[i] = -1;
488 }
489
490 /** Constructor with generated Token Manager. */
491 public Parser(ParserTokenManager tm) {
492     if (jj_initialized_once) {
493         System.out.println("ERROR: Second call to constructor of static parser. ");
494         System.out.println("        You must either use ReInit() or set the JavaCC option

```



# Parser.java

```

    STATIC to false");
495     System.out.println("        during parser generation.");
496     throw new Error();
497 }
498 jj_initialized_once = true;
499 token_source = tm;
500 token = new Token();
501 jj_ntk = -1;
502 jj_gen = 0;
503 for (int i = 0; i < 15; i++) jj_la1[i] = -1;
504 }
505
506 /** Reinitialise. */
507 public void ReInit(ParserTokenManager tm) {
508     token_source = tm;
509     token = new Token();
510     jj_ntk = -1;
511     jj_gen = 0;
512     for (int i = 0; i < 15; i++) jj_la1[i] = -1;
513 }
514
515 static private Token jj_consume_token(int kind) throws ParseException {
516     Token oldToken;
517     if ((oldToken = token).next != null) token = token.next;
518     else token = token.next = token_source.getNextToken();
519     jj_ntk = -1;
520     if (token.kind == kind) {
521         jj_gen++;
522         return token;
523     }
524     token = oldToken;
525     jj_kind = kind;
526     throw generateParseException();
527 }
528
529
530 /** Get the next Token. */
531 static final public Token getNextToken() {
532     if (token.next != null) token = token.next;
533     else token = token.next = token_source.getNextToken();
534     jj_ntk = -1;
535     jj_gen++;
536     return token;
537 }
538
539 /** Get the specific Token. */
540 static final public Token getToken(int index) {
541     Token t = token;
542     for (int i = 0; i < index; i++) {
543         if (t.next != null) t = t.next;
544         else t = t.next = token_source.getNextToken();
545     }
546     return t;
547 }
548
549 static private int jj_ntk() {
550     if ((jj_nt=token.next) == null)
551         return (jj_ntk = (token.next=token_source.getNextToken()).kind);
552     else
553         return (jj_ntk = jj_nt.kind);
554 }
555

```

```

556 static private java.util.List<int[]> jj_expentries = new java.util.ArrayList<int[]>();
557 static private int[] jj_expentry;
558 static private int jj_kind = -1;
559
560 /** Generate ParseException. */
561 static public ParseException generateParseException() {
562     jj_expentries.clear();
563     boolean[] la1tokens = new boolean[37];
564     if (jj_kind >= 0) {
565         la1tokens[jj_kind] = true;
566         jj_kind = -1;
567     }
568     for (int i = 0; i < 15; i++) {
569         if (jj_la1[i] == jj_gen) {
570             for (int j = 0; j < 32; j++) {
571                 if ((jj_la1_0[i] & (1<<j)) != 0) {
572                     la1tokens[j] = true;
573                 }
574                 if ((jj_la1_1[i] & (1<<j)) != 0) {
575                     la1tokens[32+j] = true;
576                 }
577             }
578         }
579     }
580     for (int i = 0; i < 37; i++) {
581         if (la1tokens[i]) {
582             jj_expentry = new int[1];
583             jj_expentry[0] = i;
584             jj_expentries.add(jj_expentry);
585         }
586     }
587     int[][] exptokseq = new int[jj_expentries.size()][2];
588     for (int i = 0; i < jj_expentries.size(); i++) {
589         exptokseq[i] = jj_expentries.get(i);
590     }
591     return new ParseException(token, exptokseq, tokenImage);
592 }
593
594 /** Enable tracing. */
595 static final public void enable_tracing() {
596 }
597
598 /** Disable tracing. */
599 static final public void disable_tracing() {
600 }
601
602 }
603

```