```
 1: %{
 2: // $Id: parser.y,v 1.15 2013-09-03 20:11:51-07 - - $
 3:
 4: // Convert infix notation to RPN.  */
 5:
 6: #define YYDEBUG 1
 7: #define YYERROR_VERBOSE 1
 8:
 9: #include "extern.h"
10:
11: %}
12:
13: %token IDENT
14: %start program
15:
16: %%
17:
18: program : program expr ';'       { printchar ('\n'); }
19:         | program error ';'      { }
20:         |                        { }
21:         ;
22:
23: expr    : expr '+' term          { printchar ($2); }
24:         | expr '-' term          { printchar ($2); }
25:         | term                   { }
26:         ;
27:
28: term    : term '*' factor        { printchar ($2); }
29:         | term '/' factor        { printchar ($2); }
30:         | factor                 { }
31:         ;
32:
33: factor  : '(' expr ')'           { }
34:         | IDENT                  { printchar ($1); }
35:         ;
36:
37: %%
38:
```

```
 1: %{
 2: // $Id: scanner.l,v 1.12 2014-10-10 14:40:19-07 - - $
 3:
 4: #define YY_USER_ACTION { yylval = *yytext; }
 5:
 6: #include "extern.h"
 7: #include "parser.h"
 8:
 9: %}
10:
11: %option 8bit
12: %option debug
13: %option nodefault
14: %option nounput
15: %option noyywrap
16: %option verbose
17: %option warn
18:
19: %%
20:
21: [a-zA-Z]         { return IDENT ; }
22: "+"              { return '+'; }
23: "-"              { return '-'; }
24: "*"              { return '*'; }
25: "/"              { return '/'; }
26: "("              { return '('; }
27: ")"              { return ')'; }
28: ";"              { return ';'; }
29: [ \n\t]          { /* skip white space */ }
30: "#".*            { /* skip comments */ }
31: .                { scanerror (); }
32:
33: %%
34:
```

```
 1: // $Id: main.cc,v 1.5 2014-10-10 14:48:42-07 - - $
 2:
 3: #include <ctype.h>
 4: #include <stdio.h>
 5: #include <stdlib.h>
 6: #include <unistd.h>
 7:
 8: #include "extern.h"
 9:
10: int status = EXIT_SUCCESS;
11:
12: void scan_options (int argc, char** argv) {
13:    yy_flex_debug = yydebug = 0;
14:    for (;;) {
15:       int opt = getopt (argc, argv, "ly");
16:       if (opt == EOF) break;
17:       switch (opt) {
18:          case 'l': yy_flex_debug = 1; break;
19:          case 'y': yydebug = 1; break;
20:          default : status = EXIT_FAILURE;
21:       }
22:    }
23: }
24:
25: void yyerror (const char *message) {
26:    status = EXIT_FAILURE;
27:    fflush (NULL);
28:    fprintf (stderr, "%s\n", message);
29:    fflush (NULL);
30: }
31:
32: void printchar (char byte) {
33:    putchar (byte);
34:    fprintf (stderr, "Debug: printchar ('");
35:    fprintf (stderr, isprint (byte) ? "%c" : "\\x%02X", byte);
36:    fprintf (stderr, "')\n");
37: }
38:
39: void scanerror (void) {
40:    static char message[] = "Invalid input character ";
41:    static char buffer[sizeof message + 16];
42:    sprintf (buffer, isprint (*yytext) ? "%s'%c'\n" : "%s'\\%03o'\n",
43:             message, *yytext);
44:    yyerror (buffer);
45: }
46:
47: int main (int argc, char** argv){
48:    scan_options (argc, argv);
49:    yyparse();
50:    return status;
51: }
52:
```

```
 1: // $Id: extern.h,v 1.2 2013-09-03 20:17:41-07 - - $
 2:
 3: #ifndef __EXTERN_H__
 4: #define __EXTERN_H__
 5:
 6: extern int yy_flex_debug;
 7: extern int yydebug;
 8: extern char *yytext;
 9:
10: void yyerror (const char *);
11: int yylex (void);
12: int yyparse (void);
13:
14: void printchar (char);
15: void scanerror (void);
16:
17: #endif
18:
```

```
    1:
    2: /* A Bison parser, made by GNU Bison 2.4.1.  */
    3:
    4: /* Skeleton interface for Bison's Yacc-like parsers in C
    5:
    6:       Copyright (C) 1984, 1989, 1990, 2000, 2001, 2002, 2003, 2004, 2005
, 2006
    7:    Free Software Foundation, Inc.
    8:
    9:    This program is free software: you can redistribute it and/or modify
   10:    it under the terms of the GNU General Public License as published by
   11:    the Free Software Foundation, either version 3 of the License, or
   12:    (at your option) any later version.
   13:
   14:    This program is distributed in the hope that it will be useful,
   15:    but WITHOUT ANY WARRANTY; without even the implied warranty of
   16:    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   17:    GNU General Public License for more details.
   18:
   19:    You should have received a copy of the GNU General Public License
   20:    along with this program.  If not, see <http://www.gnu.org/licenses/>.
  */
   21:
   22: /* As a special exception, you may create a larger work that contains
   23:    part or all of the Bison parser skeleton and distribute that work
   24:    under terms of your choice, so long as that work isn't itself a
   25:    parser generator using the skeleton or a modified version thereof
   26:    as a parser skeleton.  Alternatively, if you modify or redistribute
   27:    the parser skeleton itself, you may (at your option) remove this
   28:    special exception, which will cause the skeleton and the resulting
   29:    Bison output files to be licensed under the GNU General Public
   30:    License without this special exception.
   31:
   32:    This special exception was added by the Free Software Foundation in
   33:    version 2.2 of Bison.  */
   34:
   35:
   36: /* Tokens.  */
   37: #ifndef YYTOKENTYPE
   38: # define YYTOKENTYPE
   39:    /* Put the tokens into the symbol table, so that GDB and other debugg
ers
   40:       know about them.  */
   41:    enum yytokentype {
   42:      IDENT = 258
   43:    };
   44: #endif
   45:
   46:
   47:
   48: #if ! defined YYSTYPE && ! defined YYSTYPE_IS_DECLARED
   49: typedef int YYSTYPE;
   50: # define YYSTYPE_IS_TRIVIAL 1
   51: # define yystype YYSTYPE /* obsolescent; will be withdrawn */
   52: # define YYSTYPE_IS_DECLARED 1
   53: #endif
   54:
   55: extern YYSTYPE yylval;
```

```
56:
57:
```

```
  1: # $Id: Makefile,v 1.13 2014-10-10 14:53:15-07 - - $
  2:
  3: #
  4: # Define programs and options to be used.
  5: #
  6:
  7: MKFILE   = Makefile
  8: DEPFILE  = ${MKFILE}.dep
  9: NOINCL   = ci clean spotless
 10: NEEDINCL = ${filter ${NOINCL}, ${MAKECMDGOALS}}
 11: GMAKE    = ${MAKE} --no-print-directory
 12:
 13: GCC      = g++ -g -O0 -Wall -Wextra -std=gnu++11
 14: GCCDEP   = g++ -MM
 15:
 16: #
 17: # Define set of files to be processed.
 18: #
 19:
 20: SOURCES  = parser.y scanner.l main.cc extern.h parser.h ${MKFILE}
 21: GENS     = parser.h parser.cc parser.log scanner.cc scanner.log
 22: OBJECTS  = parser.o scanner.o main.o
 23: EXECBIN  = infixtorpn
 24: CSOURCES = ${filter %.cc, ${SOURCES} ${GENS}}
 25: OUTPUTS  = scanner.log parser.log test1.lis test2.lis
 26:
 27: #
 28: # General recipes for building software.
 29: #
 30:
 31: all : ${EXECBIN}
 32:
 33: infixtorpn : ${OBJECTS}
 34:         ${GCC} -o infixtorpn ${OBJECTS}
 35:
 36: %.cc : %.l
 37:         flex -o$@ $< >$*.log 2>&1
 38:         - cat lex.backup >> $*.log
 39:         - rm lex.backup
 40:
 41: %.cc : %.y
 42:         bison -dtv -o$@ $<
 43:         mv parser.hh parser.h
 44:         - mv $*.output $*.log
 45:
 46: %.o : %.cc
 47:         ${GCC} -c $<
 48:
```

```
49:
50: #
51: # Other miscellaneous actions.
52: #
53:
54: test1.lis : test1.in ${DEPFILE} ${EXECBIN}
55:         infixtorpn -ly <test1.in >test1.out 2>test1.err
56:         morecat ${DEPFILE} test1.in test1.out test1.err >test1.lis
57:         rm test1.out test1.err
58:
59: test2.lis : test2.in ${DEPFILE} ${EXECBIN}
60:         infixtorpn -ly <test2.in >test2.out 2>test2.err
61:         morecat ${DEPFILE} test2.in test2.out test2.err >test2.lis
62:         rm test2.out test2.err
63:
64: lis : ${SOURCES} ${OUTPUTS}
65:         mkpspdf Listing.ps ${SOURCES} ${OUTPUTS}
66:
67: ci : ${SOURCES}
68:         cid + ${SOURCES}
69:
70: clean :
71:         - rm ${OBJECTS} ${GENS} core
72:
73: spotless : clean
74:         - rm ${EXECBIN} Listing.ps Listing.pdf ${DEPFILE}
75:
76: again :
77:         ${GMAKE} spotless ci all lis
78:
79: deps : ${CSOURCES}
80:         @ echo "# ${DEPFILE} created `LC_TIME=C date`" >${DEPFILE}
81:         ${GCCDEP} ${CSOURCES} >>${DEPFILE}
82:
83: ${DEPFILE} :
84:         @ touch ${DEPFILE}
85:         ${GMAKE} deps
86:
87: ifeq (${NEEDINCL},)
88: include ${DEPFILE}
89: endif
```

```
flex version 2.5.35 usage statistics:
  scanner options: -dsvI8 -Cem -oscanner.cc
  50/2000 NFA states
  16/1000 DFA states (65 words)
  11 rules
  Compressed tables always back-up
  1/40 start conditions
  37 epsilon states, 15 double epsilon states
  5/100 character classes needed 56/500 words of storage, 0 reused
  46 state/nextstate pairs created
  26/20 unique/duplicate transitions
  18/1000 base-def entries created
  27/2000 (peak 24) nxt-chk entries created
  4/2500 (peak 24) template nxt-chk entries created
  0 empty table entries
  2 protos created
  2 templates created, 2 uses
  12/256 equivalence classes created
  2/256 meta-equivalence classes created
  0 (0 saved) hash collisions, 13 DFAs equal
  0 sets of reallocations needed
  358 total table entries needed
```

```
Grammar

    0 $accept: program $end

    1 program: program expr ';'
    2        | program error ';'
    3        | /* empty */

    4 expr: expr '+' term
    5     | expr '-' term
    6     | term

    7 term: term '*' factor
    8     | term '/' factor
    9     | factor

   10 factor: '(' expr ')'
   11       | IDENT


Terminals, with rules where they appear

$end (0) 0
'(' (40) 10
')' (41) 10
'*' (42) 7
'+' (43) 4
'-' (45) 5
'/' (47) 8
';' (59) 1 2
error (256) 2
IDENT (258) 11


Nonterminals, with rules where they appear

$accept (11)
    on left: 0
program (12)
    on left: 1 2 3, on right: 0 1 2
expr (13)
    on left: 4 5 6, on right: 1 4 5 10
term (14)
    on left: 7 8 9, on right: 4 5 6 7 8
factor (15)
    on left: 10 11, on right: 7 8 9


state 0

    0 $accept: . program $end

    $default  reduce using rule 3 (program)

    program  go to state 1


state 1
```

```
 59:
 60:     0 $accept: program . $end
 61:     1 program: program . expr ';'
 62:     2        | program . error ';'
 63:
 64:     $end   shift, and go to state 2
 65:     error  shift, and go to state 3
 66:     IDENT  shift, and go to state 4
 67:     '('    shift, and go to state 5
 68:
 69:     expr    go to state 6
 70:     term    go to state 7
 71:     factor  go to state 8
 72:
 73:
 74: state 2
 75:
 76:     0 $accept: program $end .
 77:
 78:     $default  accept
 79:
 80:
 81: state 3
 82:
 83:     2 program: program error . ';'
 84:
 85:     ';'  shift, and go to state 9
 86:
 87:
 88: state 4
 89:
 90:    11 factor: IDENT .
 91:
 92:     $default  reduce using rule 11 (factor)
 93:
 94:
 95: state 5
 96:
 97:    10 factor: '(' . expr ')'
 98:
 99:     IDENT  shift, and go to state 4
100:     '('    shift, and go to state 5
101:
102:     expr    go to state 10
103:     term    go to state 7
104:     factor  go to state 8
105:
106:
107: state 6
108:
109:     1 program: program expr . ';'
110:     4 expr: expr . '+' term
111:     5     | expr . '-' term
112:
113:     ';'  shift, and go to state 11
114:     '+'  shift, and go to state 12
115:     '-'  shift, and go to state 13
116:
```

```
117:
118: state 7
119:
120:     6 expr: term .
121:     7 term: term . '*' factor
122:     8     | term . '/' factor
123:
124:     '*'  shift, and go to state 14
125:     '/'  shift, and go to state 15
126:
127:     $default  reduce using rule 6 (expr)
128:
129:
130: state 8
131:
132:     9 term: factor .
133:
134:     $default  reduce using rule 9 (term)
135:
136:
137: state 9
138:
139:     2 program: program error ';' .
140:
141:     $default  reduce using rule 2 (program)
142:
143:
144: state 10
145:
146:     4 expr: expr . '+' term
147:     5     | expr . '-' term
148:    10 factor: '(' expr . ')'
149:
150:     '+'  shift, and go to state 12
151:     '-'  shift, and go to state 13
152:     ')'  shift, and go to state 16
153:
154:
155: state 11
156:
157:     1 program: program expr ';' .
158:
159:     $default  reduce using rule 1 (program)
160:
161:
162: state 12
163:
164:     4 expr: expr '+' . term
165:
166:     IDENT  shift, and go to state 4
167:     '('    shift, and go to state 5
168:
169:     term    go to state 17
170:     factor  go to state 8
171:
172:
173: state 13
174:
```

```
175:     5 expr: expr '-' . term
176:
177:     IDENT  shift, and go to state 4
178:     '('    shift, and go to state 5
179:
180:     term    go to state 18
181:     factor  go to state 8
182:
183:
184: state 14
185:
186:     7 term: term '*' . factor
187:
188:     IDENT  shift, and go to state 4
189:     '('    shift, and go to state 5
190:
191:     factor  go to state 19
192:
193:
194: state 15
195:
196:     8 term: term '/' . factor
197:
198:     IDENT  shift, and go to state 4
199:     '('    shift, and go to state 5
200:
201:     factor  go to state 20
202:
203:
204: state 16
205:
206:    10 factor: '(' expr ')' .
207:
208:     $default  reduce using rule 10 (factor)
209:
210:
211: state 17
212:
213:     4 expr: expr '+' term .
214:     7 term: term . '*' factor
215:     8     | term . '/' factor
216:
217:     '*'  shift, and go to state 14
218:     '/'  shift, and go to state 15
219:
220:     $default  reduce using rule 4 (expr)
221:
222:
223: state 18
224:
225:     5 expr: expr '-' term .
226:     7 term: term . '*' factor
227:     8     | term . '/' factor
228:
229:     '*'  shift, and go to state 14
230:     '/'  shift, and go to state 15
231:
232:     $default  reduce using rule 5 (expr)
```

```
233:
234:
235: state 19
236:
237:     7 term: term '*' factor .
238:
239:     $default  reduce using rule 7 (term)
240:
241:
242: state 20
243:
244:     8 term: term '/' factor .
245:
246:     $default  reduce using rule 8 (term)
```

```
  1: ::::::::::::::::
  2: Makefile.dep
  3: ::::::::::::::::
  4:      1  # Makefile.dep created Fri Oct 10 14:48:42 PDT 2014
  5:      2  main.o: main.cc extern.h
  6:      3  parser.o: parser.cc extern.h
  7:      4  scanner.o: scanner.cc extern.h parser.h
  8: ::::::::::::::::
  9: test1.in
 10: ::::::::::::::::
 11:      1  a*b+c*d;
 12: ::::::::::::::::
 13: test1.out
 14: ::::::::::::::::
 15:      1  ab*cd*+
 16: ::::::::::::::::
 17: test1.err
 18: ::::::::::::::::
 19:      1  Starting parse
 20:      2  Entering state 0
 21:      3  Reducing stack by rule 3 (line 20):
 22:      4  -> $$ = nterm program ()
 23:      5  Stack now 0
 24:      6  Entering state 1
 25:      7  Reading a token: --(end of buffer or a NUL)
 26:      8  --accepting rule at line 21 ("a")
 27:      9  Next token is token IDENT ()
 28:     10  Shifting token IDENT ()
 29:     11  Entering state 4
 30:     12  Reducing stack by rule 11 (line 34):
 31:     13     $1 = token IDENT ()
 32:     14  Debug: printchar ('a')
 33:     15  -> $$ = nterm factor ()
 34:     16  Stack now 0 1
 35:     17  Entering state 8
 36:     18  Reducing stack by rule 9 (line 30):
 37:     19     $1 = nterm factor ()
 38:     20  -> $$ = nterm term ()
 39:     21  Stack now 0 1
 40:     22  Entering state 7
 41:     23  Reading a token: --accepting rule at line 24 ("*")
 42:     24  Next token is token '*' ()
 43:     25  Shifting token '*' ()
 44:     26  Entering state 14
 45:     27  Reading a token: --accepting rule at line 21 ("b")
 46:     28  Next token is token IDENT ()
 47:     29  Shifting token IDENT ()
 48:     30  Entering state 4
 49:     31  Reducing stack by rule 11 (line 34):
 50:     32     $1 = token IDENT ()
 51:     33  Debug: printchar ('b')
 52:     34  -> $$ = nterm factor ()
 53:     35  Stack now 0 1 7 14
 54:     36  Entering state 19
 55:     37  Reducing stack by rule 7 (line 28):
 56:     38     $1 = nterm term ()
 57:     39     $2 = token '*' ()
 58:     40     $3 = nterm factor ()
```

```
 59:     41  Debug: printchar ('*')
 60:     42  -> $$ = nterm term ()
 61:     43  Stack now 0 1
 62:     44  Entering state 7
 63:     45  Reading a token: --accepting rule at line 22 ("+")
 64:     46  Next token is token '+' ()
 65:     47  Reducing stack by rule 6 (line 25):
 66:     48     $1 = nterm term ()
 67:     49  -> $$ = nterm expr ()
 68:     50  Stack now 0 1
 69:     51  Entering state 6
 70:     52  Next token is token '+' ()
 71:     53  Shifting token '+' ()
 72:     54  Entering state 12
 73:     55  Reading a token: --accepting rule at line 21 ("c")
 74:     56  Next token is token IDENT ()
 75:     57  Shifting token IDENT ()
 76:     58  Entering state 4
 77:     59  Reducing stack by rule 11 (line 34):
 78:     60     $1 = token IDENT ()
 79:     61  Debug: printchar ('c')
 80:     62  -> $$ = nterm factor ()
 81:     63  Stack now 0 1 6 12
 82:     64  Entering state 8
 83:     65  Reducing stack by rule 9 (line 30):
 84:     66     $1 = nterm factor ()
 85:     67  -> $$ = nterm term ()
 86:     68  Stack now 0 1 6 12
 87:     69  Entering state 17
 88:     70  Reading a token: --accepting rule at line 24 ("*")
 89:     71  Next token is token '*' ()
 90:     72  Shifting token '*' ()
 91:     73  Entering state 14
 92:     74  Reading a token: --accepting rule at line 21 ("d")
 93:     75  Next token is token IDENT ()
 94:     76  Shifting token IDENT ()
 95:     77  Entering state 4
 96:     78  Reducing stack by rule 11 (line 34):
 97:     79     $1 = token IDENT ()
 98:     80  Debug: printchar ('d')
 99:     81  -> $$ = nterm factor ()
100:     82  Stack now 0 1 6 12 17 14
101:     83  Entering state 19
102:     84  Reducing stack by rule 7 (line 28):
103:     85     $1 = nterm term ()
104:     86     $2 = token '*' ()
105:     87     $3 = nterm factor ()
106:     88  Debug: printchar ('*')
107:     89  -> $$ = nterm term ()
108:     90  Stack now 0 1 6 12
109:     91  Entering state 17
110:     92  Reading a token: --accepting rule at line 28 (";")
111:     93  Next token is token ';' ()
112:     94  Reducing stack by rule 4 (line 23):
113:     95     $1 = nterm expr ()
114:     96     $2 = token '+' ()
115:     97     $3 = nterm term ()
116:     98  Debug: printchar ('+')
```

```
117:     99  -> $$ = nterm expr ()
118:    100  Stack now 0 1
119:    101  Entering state 6
120:    102  Next token is token ';' ()
121:    103  Shifting token ';' ()
122:    104  Entering state 11
123:    105  Reducing stack by rule 1 (line 18):
124:    106     $1 = nterm program ()
125:    107     $2 = nterm expr ()
126:    108     $3 = token ';' ()
127:    109  Debug: printchar ('\x0A')
128:    110  -> $$ = nterm program ()
129:    111  Stack now 0
130:    112  Entering state 1
131:    113  Reading a token: --accepting rule at line 29 ("
132:    114  ")
133:    115  --(end of buffer or a NUL)
134:    116  --EOF (start condition 0)
135:    117  Now at end of input.
136:    118  Shifting token $end ()
137:    119  Entering state 2
138:    120  Stack now 0 1 2
139:    121  Cleanup: popping token $end ()
140:    122  Cleanup: popping nterm program ()
```

```
  1: ::::::::::::::::
  2: Makefile.dep
  3: ::::::::::::::::
  4:      1  # Makefile.dep created Fri Oct 10 14:48:42 PDT 2014
  5:      2  main.o: main.cc extern.h
  6:      3  parser.o: parser.cc extern.h
  7:      4  scanner.o: scanner.cc extern.h parser.h
  8: ::::::::::::::::
  9: test2.in
 10: ::::::::::::::::
 11:      1  (a+b)*c;
 12: ::::::::::::::::
 13: test2.out
 14: ::::::::::::::::
 15:      1  ab+c*
 16: ::::::::::::::::
 17: test2.err
 18: ::::::::::::::::
 19:      1  Starting parse
 20:      2  Entering state 0
 21:      3  Reducing stack by rule 3 (line 20):
 22:      4  -> $$ = nterm program ()
 23:      5  Stack now 0
 24:      6  Entering state 1
 25:      7  Reading a token: --(end of buffer or a NUL)
 26:      8  --accepting rule at line 26 ("(")
 27:      9  Next token is token '(' ()
 28:     10  Shifting token '(' ()
 29:     11  Entering state 5
 30:     12  Reading a token: --accepting rule at line 21 ("a")
 31:     13  Next token is token IDENT ()
 32:     14  Shifting token IDENT ()
 33:     15  Entering state 4
 34:     16  Reducing stack by rule 11 (line 34):
 35:     17     $1 = token IDENT ()
 36:     18  Debug: printchar ('a')
 37:     19  -> $$ = nterm factor ()
 38:     20  Stack now 0 1 5
 39:     21  Entering state 8
 40:     22  Reducing stack by rule 9 (line 30):
 41:     23     $1 = nterm factor ()
 42:     24  -> $$ = nterm term ()
 43:     25  Stack now 0 1 5
 44:     26  Entering state 7
 45:     27  Reading a token: --accepting rule at line 22 ("+")
 46:     28  Next token is token '+' ()
 47:     29  Reducing stack by rule 6 (line 25):
 48:     30     $1 = nterm term ()
 49:     31  -> $$ = nterm expr ()
 50:     32  Stack now 0 1 5
 51:     33  Entering state 10
 52:     34  Next token is token '+' ()
 53:     35  Shifting token '+' ()
 54:     36  Entering state 12
 55:     37  Reading a token: --accepting rule at line 21 ("b")
 56:     38  Next token is token IDENT ()
 57:     39  Shifting token IDENT ()
 58:     40  Entering state 4
```

```
 59:     41  Reducing stack by rule 11 (line 34):
 60:     42     $1 = token IDENT ()
 61:     43  Debug: printchar ('b')
 62:     44  -> $$ = nterm factor ()
 63:     45  Stack now 0 1 5 10 12
 64:     46  Entering state 8
 65:     47  Reducing stack by rule 9 (line 30):
 66:     48     $1 = nterm factor ()
 67:     49  -> $$ = nterm term ()
 68:     50  Stack now 0 1 5 10 12
 69:     51  Entering state 17
 70:     52  Reading a token: --accepting rule at line 27 (")")
 71:     53  Next token is token ')' ()
 72:     54  Reducing stack by rule 4 (line 23):
 73:     55     $1 = nterm expr ()
 74:     56     $2 = token '+' ()
 75:     57     $3 = nterm term ()
 76:     58  Debug: printchar ('+')
 77:     59  -> $$ = nterm expr ()
 78:     60  Stack now 0 1 5
 79:     61  Entering state 10
 80:     62  Next token is token ')' ()
 81:     63  Shifting token ')' ()
 82:     64  Entering state 16
 83:     65  Reducing stack by rule 10 (line 33):
 84:     66     $1 = token '(' ()
 85:     67     $2 = nterm expr ()
 86:     68     $3 = token ')' ()
 87:     69  -> $$ = nterm factor ()
 88:     70  Stack now 0 1
 89:     71  Entering state 8
 90:     72  Reducing stack by rule 9 (line 30):
 91:     73     $1 = nterm factor ()
 92:     74  -> $$ = nterm term ()
 93:     75  Stack now 0 1
 94:     76  Entering state 7
 95:     77  Reading a token: --accepting rule at line 24 ("*")
 96:     78  Next token is token '*' ()
 97:     79  Shifting token '*' ()
 98:     80  Entering state 14
 99:     81  Reading a token: --accepting rule at line 21 ("c")
100:     82  Next token is token IDENT ()
101:     83  Shifting token IDENT ()
102:     84  Entering state 4
103:     85  Reducing stack by rule 11 (line 34):
104:     86     $1 = token IDENT ()
105:     87  Debug: printchar ('c')
106:     88  -> $$ = nterm factor ()
107:     89  Stack now 0 1 7 14
108:     90  Entering state 19
109:     91  Reducing stack by rule 7 (line 28):
110:     92     $1 = nterm term ()
111:     93     $2 = token '*' ()
112:     94     $3 = nterm factor ()
113:     95  Debug: printchar ('*')
114:     96  -> $$ = nterm term ()
115:     97  Stack now 0 1
116:     98  Entering state 7
```

```
117:      99  Reading a token: --accepting rule at line 28 (";")
118:     100  Next token is token ';' ()
119:     101  Reducing stack by rule 6 (line 25):
120:     102     $1 = nterm term ()
121:     103  -> $$ = nterm expr ()
122:     104  Stack now 0 1
123:     105  Entering state 6
124:     106  Next token is token ';' ()
125:     107  Shifting token ';' ()
126:     108  Entering state 11
127:     109  Reducing stack by rule 1 (line 18):
128:     110     $1 = nterm program ()
129:     111     $2 = nterm expr ()
130:     112     $3 = token ';' ()
131:     113  Debug: printchar ('\x0A')
132:     114  -> $$ = nterm program ()
133:     115  Stack now 0
134:     116  Entering state 1
135:     117  Reading a token: --accepting rule at line 29 ("
136:     118  ")
137:     119  --(end of buffer or a NUL)
138:     120  --EOF (start condition 0)
139:     121  Now at end of input.
140:     122  Shifting token $end ()
141:     123  Entering state 2
142:     124  Stack now 0 1 2
143:     125  Cleanup: popping token $end ()
144:     126  Cleanup: popping nterm program ()
```