```
 1: %{
 2: // $Id: etf.yy,v 1.5 2014-11-06 19:04:28-08 - - $
 3:
 4: #include <assert.h>
 5: #include <ctype.h>
 6: #include <stdio.h>
 7: #include <stdlib.h>
 8: #include <string.h>
 9:
10: #define YYDEBUG 1
11: #define YYERROR_VERBOSE 1
12: #define YYPRINT(FILE,TYPE,TREE) yyprint (FILE, TYPE, TREE)
13: #define YYSTYPE tree*
14:
15: struct tree {
16:    int nodenr;
17:    int Vt;
18:    char lexeme;
19:    tree* left;
20:    tree* right;
21: };
22:
23: void yyprint (FILE* file, int type, tree* tree);
24: tree* trace (tree* tree);
25: tree* adopt2 (tree* root, tree* left, tree* right);
26: int yylex (void);
27: void yyerror (const char* message);
28:
29: tree* root = nullptr;
30:
31: %}
32:
33: %token-table
34: %verbose
35:
36: %token IDENT ADD MUL LPAR RPAR
37: %start start
38:
39: %%
40:
41: start  : expr               {root = $$ = trace ($1); }
42:        ;
43:
44: expr   : expr ADD term       {$$ = trace (adopt2 ($2, $1, $3)); }
45:        | term               {$$ = trace ($1); }
46:        ;
47:
48: term   : term MUL factor     {$$ = trace (adopt2 ($2, $1, $3)); }
49:        | factor             {$$ = trace ($1); }
50:        ;
51:
52: factor : LPAR expr RPAR      {$$ = trace ($2); }
53:        | IDENT              {$$ = trace ($1); }
54:        ;
55:
56: %%
```

```
57:
58:
59: tree* adopt2 (tree* root, tree* left, tree* right) {
60:    root->left = left;
61:    root->right = right;
62:    return root;
63: }
64:
65: int setyylval (int Vt, char lexeme) {
66:    static int static_nodenr = 0;
67:    yylval = new tree();
68:    yylval->nodenr = ++static_nodenr;
69:    yylval->lexeme = lexeme;
70:    yylval->left = yylval->right = nullptr;
71:    yylval->Vt = Vt;
72:    return Vt;
73: }
74:
75: void print_char (FILE* file, char ch) {
76:    fprintf (file, isgraph (ch) ? "'%c'" : "'\\x%02X'", ch);
77: }
78:
79: char *input_string = nullptr;
80: char *nextchar = nullptr;
81: int yylex (void) {
82:    for (;;) {
83:       int ch = *nextchar++;
84:       if (ch == '\0') return setyylval (0, ch);
85:       if (isspace (ch)) continue;
86:       if (isalpha (ch)) return setyylval (IDENT, ch);
87:       switch (ch) {
88:          case '+': return setyylval (ADD, ch);
89:          case '*': return setyylval (MUL, ch);
90:          case '(': return setyylval (LPAR, ch);
91:          case ')': return setyylval (RPAR, ch);
92:       }
93:       fprintf (stderr, "Bad character ");
94:       print_char (stderr, ch);
95:       fprintf (stderr, "\n");
96:    }
97: }
98:
```

```
 99:
100: void yyprint_child_nr (FILE* file, tree* node) {
101:    if (node == nullptr) fprintf (file, " nil");
102:                         else fprintf (file, " node%d", node->nodenr);
103: }
104: void yyprint (FILE* file, int type, tree* tree){
105:    fprintf (file, "node%d: %s ",
106:             tree->nodenr, yytname[YYTRANSLATE(type)]);
107:    print_char (file, tree->lexeme);
108:    yyprint_child_nr (file, tree->left);
109:    yyprint_child_nr (file, tree->right);
110: }
111:
112: void preorder (tree* tree, int depth) {
113:    if (tree == nullptr) return;
114:    fprintf (stderr, "AST: %*s", depth * 3, "");
115:    yyprint (stderr, tree->Vt, tree);
116:    fprintf (stderr, "\n");
117:    preorder (tree->left, depth + 1);
118:    preorder (tree->right, depth + 1);
119: }
120:
121: tree* trace (tree* tree) {
122:    preorder (tree, 0);
123:    return tree;
124: }
125:
126: void yyerror (const char* message) {
127:    fprintf (stderr, "%s)\n", message);
128: }
129:
130: int main (int argc, char** argv) {
131:    assert (argc == 2);
132:    nextchar = input_string = argv[1];
133:    fprintf (stderr, "Scanning input: \"%s\"\n", input_string);
134:    yydebug = 1;
135:    int status = yyparse ();
136:    fprintf (stderr, "Finished parse: status %d\n", status);
137:    fprintf (stderr, "Scanned input: \"%s\"\n", input_string);
138:    fprintf (stderr, "Root of AST:\n");
139:    trace (root);
140:    return 0;
141: }
142:
```

```
Grammar

    0 $accept: start $end

    1 start: expr

    2 expr: expr ADD term
    3     | term

    4 term: term MUL factor
    5     | factor

    6 factor: LPAR expr RPAR
    7       | IDENT


Terminals, with rules where they appear

$end (0) 0
error (256)
IDENT (258) 7
ADD (259) 2
MUL (260) 4
LPAR (261) 6
RPAR (262) 6


Nonterminals, with rules where they appear

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


state 0

    0 $accept: . start $end

    IDENT  shift, and go to state 1
    LPAR   shift, and go to state 2

    start   go to state 3
    expr    go to state 4
    term    go to state 5
    factor  go to state 6


state 1

    7 factor: IDENT .
```

```
    $default  reduce using rule 7 (factor)


state 2

    6 factor: LPAR . expr RPAR

    IDENT  shift, and go to state 1
    LPAR   shift, and go to state 2

    expr    go to state 7
    term    go to state 5
    factor  go to state 6


state 3

    0 $accept: start . $end

    $end  shift, and go to state 8


state 4

    1 start: expr .
    2 expr: expr . ADD term

    ADD  shift, and go to state 9

    $default  reduce using rule 1 (start)


state 5

    3 expr: term .
    4 term: term . MUL factor

    MUL  shift, and go to state 10

    $default  reduce using rule 3 (expr)


state 6

    5 term: factor .

    $default  reduce using rule 5 (term)


state 7

    2 expr: expr . ADD term
    6 factor: LPAR expr . RPAR

    ADD   shift, and go to state 9
    RPAR  shift, and go to state 11
```

```
117: state 8
118:
119:     0 $accept: start $end .
120:
121:     $default  accept
122:
123:
124: state 9
125:
126:     2 expr: expr ADD . term
127:
128:     IDENT  shift, and go to state 1
129:     LPAR   shift, and go to state 2
130:
131:     term    go to state 12
132:     factor  go to state 6
133:
134:
135: state 10
136:
137:     4 term: term MUL . factor
138:
139:     IDENT  shift, and go to state 1
140:     LPAR   shift, and go to state 2
141:
142:     factor  go to state 13
143:
144:
145: state 11
146:
147:     6 factor: LPAR expr RPAR .
148:
149:     $default  reduce using rule 6 (factor)
150:
151:
152: state 12
153:
154:     2 expr: expr ADD term .
155:     4 term: term . MUL factor
156:
157:     MUL  shift, and go to state 10
158:
159:     $default  reduce using rule 2 (expr)
160:
161:
162: state 13
163:
164:     4 term: term MUL factor .
165:
166:     $default  reduce using rule 4 (term)
```

```
 1: Scanning input: "a*b+c*d"
 2: Starting parse
 3: Entering state 0
 4: Reading a token: Next token is token IDENT (node1: IDENT 'a' nil nil)
 5: Shifting token IDENT (node1: IDENT 'a' nil nil)
 6: Entering state 1
 7: Reducing stack by rule 7 (line 53):
 8:    $1 = token IDENT (node1: IDENT 'a' nil nil)
 9: AST: node1: IDENT 'a' nil nil
10: -> $$ = nterm factor ()
11: Stack now 0
12: Entering state 6
13: Reducing stack by rule 5 (line 49):
14:    $1 = nterm factor ()
15: AST: node1: IDENT 'a' nil nil
16: -> $$ = nterm term ()
17: Stack now 0
18: Entering state 5
19: Reading a token: Next token is token MUL (node2: MUL '*' nil nil)
20: Shifting token MUL (node2: MUL '*' nil nil)
21: Entering state 10
22: Reading a token: Next token is token IDENT (node3: IDENT 'b' nil nil)
23: Shifting token IDENT (node3: IDENT 'b' nil nil)
24: Entering state 1
25: Reducing stack by rule 7 (line 53):
26:    $1 = token IDENT (node3: IDENT 'b' nil nil)
27: AST: node3: IDENT 'b' nil nil
28: -> $$ = nterm factor ()
29: Stack now 0 5 10
30: Entering state 13
31: Reducing stack by rule 4 (line 48):
32:    $1 = nterm term ()
33:    $2 = token MUL (node2: MUL '*' nil nil)
34:    $3 = nterm factor ()
35: AST: node2: MUL '*' node1 node3
36: AST:    node1: IDENT 'a' nil nil
37: AST:    node3: IDENT 'b' nil nil
38: -> $$ = nterm term ()
39: Stack now 0
40: Entering state 5
41: Reading a token: Next token is token ADD (node4: ADD '+' nil nil)
42: Reducing stack by rule 3 (line 45):
43:    $1 = nterm term ()
44: AST: node2: MUL '*' node1 node3
45: AST:    node1: IDENT 'a' nil nil
46: AST:    node3: IDENT 'b' nil nil
47: -> $$ = nterm expr ()
48: Stack now 0
49: Entering state 4
50: Next token is token ADD (node4: ADD '+' nil nil)
51: Shifting token ADD (node4: ADD '+' nil nil)
52: Entering state 9
53: Reading a token: Next token is token IDENT (node5: IDENT 'c' nil nil)
54: Shifting token IDENT (node5: IDENT 'c' nil nil)
55: Entering state 1
56: Reducing stack by rule 7 (line 53):
57:    $1 = token IDENT (node5: IDENT 'c' nil nil)
58: AST: node5: IDENT 'c' nil nil
```

```
 59: -> $$ = nterm factor ()
 60: Stack now 0 4 9
 61: Entering state 6
 62: Reducing stack by rule 5 (line 49):
 63:    $1 = nterm factor ()
 64: AST: node5: IDENT 'c' nil nil
 65: -> $$ = nterm term ()
 66: Stack now 0 4 9
 67: Entering state 12
 68: Reading a token: Next token is token MUL (node6: MUL '*' nil nil)
 69: Shifting token MUL (node6: MUL '*' nil nil)
 70: Entering state 10
 71: Reading a token: Next token is token IDENT (node7: IDENT 'd' nil nil)
 72: Shifting token IDENT (node7: IDENT 'd' nil nil)
 73: Entering state 1
 74: Reducing stack by rule 7 (line 53):
 75:    $1 = token IDENT (node7: IDENT 'd' nil nil)
 76: AST: node7: IDENT 'd' nil nil
 77: -> $$ = nterm factor ()
 78: Stack now 0 4 9 12 10
 79: Entering state 13
 80: Reducing stack by rule 4 (line 48):
 81:    $1 = nterm term ()
 82:    $2 = token MUL (node6: MUL '*' nil nil)
 83:    $3 = nterm factor ()
 84: AST: node6: MUL '*' node5 node7
 85: AST:    node5: IDENT 'c' nil nil
 86: AST:    node7: IDENT 'd' nil nil
 87: -> $$ = nterm term ()
 88: Stack now 0 4 9
 89: Entering state 12
 90: Reading a token: Now at end of input.
 91: Reducing stack by rule 2 (line 44):
 92:    $1 = nterm expr ()
 93:    $2 = token ADD (node4: ADD '+' nil nil)
 94:    $3 = nterm term ()
 95: AST: node4: ADD '+' node2 node6
 96: AST:    node2: MUL '*' node1 node3
 97: AST:       node1: IDENT 'a' nil nil
 98: AST:       node3: IDENT 'b' nil nil
 99: AST:    node6: MUL '*' node5 node7
100: AST:       node5: IDENT 'c' nil nil
101: AST:       node7: IDENT 'd' nil nil
102: -> $$ = nterm expr ()
103: Stack now 0
104: Entering state 4
105: Now at end of input.
106: Reducing stack by rule 1 (line 41):
107:    $1 = nterm expr ()
108: AST: node4: ADD '+' node2 node6
109: AST:    node2: MUL '*' node1 node3
110: AST:       node1: IDENT 'a' nil nil
111: AST:       node3: IDENT 'b' nil nil
112: AST:    node6: MUL '*' node5 node7
113: AST:       node5: IDENT 'c' nil nil
114: AST:       node7: IDENT 'd' nil nil
115: -> $$ = nterm start ()
116: Stack now 0
```

```
117: Entering state 3
118: Now at end of input.
119: Shifting token $end (node8: $end '\x00' nil nil)
120: Entering state 8
121: Stack now 0 3 8
122: Cleanup: popping token $end (node8: $end '\x00' nil nil)
123: Cleanup: popping nterm start ()
124: Finished parse: status 0
125: Scanned input: "a*b+c*d"
126: Root of AST:
127: AST: node4: ADD '+' node2 node6
128: AST:    node2: MUL '*' node1 node3
129: AST:       node1: IDENT 'a' nil nil
130: AST:       node3: IDENT 'b' nil nil
131: AST:    node6: MUL '*' node5 node7
132: AST:       node5: IDENT 'c' nil nil
133: AST:       node7: IDENT 'd' nil nil
```

```
 1: Scanning input: "a*(b+c)"
 2: Starting parse
 3: Entering state 0
 4: Reading a token: Next token is token IDENT (node1: IDENT 'a' nil nil)
 5: Shifting token IDENT (node1: IDENT 'a' nil nil)
 6: Entering state 1
 7: Reducing stack by rule 7 (line 53):
 8:    $1 = token IDENT (node1: IDENT 'a' nil nil)
 9: AST: node1: IDENT 'a' nil nil
10: -> $$ = nterm factor ()
11: Stack now 0
12: Entering state 6
13: Reducing stack by rule 5 (line 49):
14:    $1 = nterm factor ()
15: AST: node1: IDENT 'a' nil nil
16: -> $$ = nterm term ()
17: Stack now 0
18: Entering state 5
19: Reading a token: Next token is token MUL (node2: MUL '*' nil nil)
20: Shifting token MUL (node2: MUL '*' nil nil)
21: Entering state 10
22: Reading a token: Next token is token LPAR (node3: LPAR '(' nil nil)
23: Shifting token LPAR (node3: LPAR '(' nil nil)
24: Entering state 2
25: Reading a token: Next token is token IDENT (node4: IDENT 'b' nil nil)
26: Shifting token IDENT (node4: IDENT 'b' nil nil)
27: Entering state 1
28: Reducing stack by rule 7 (line 53):
29:    $1 = token IDENT (node4: IDENT 'b' nil nil)
30: AST: node4: IDENT 'b' nil nil
31: -> $$ = nterm factor ()
32: Stack now 0 5 10 2
33: Entering state 6
34: Reducing stack by rule 5 (line 49):
35:    $1 = nterm factor ()
36: AST: node4: IDENT 'b' nil nil
37: -> $$ = nterm term ()
38: Stack now 0 5 10 2
39: Entering state 5
40: Reading a token: Next token is token ADD (node5: ADD '+' nil nil)
41: Reducing stack by rule 3 (line 45):
42:    $1 = nterm term ()
43: AST: node4: IDENT 'b' nil nil
44: -> $$ = nterm expr ()
45: Stack now 0 5 10 2
46: Entering state 7
47: Next token is token ADD (node5: ADD '+' nil nil)
48: Shifting token ADD (node5: ADD '+' nil nil)
49: Entering state 9
50: Reading a token: Next token is token IDENT (node6: IDENT 'c' nil nil)
51: Shifting token IDENT (node6: IDENT 'c' nil nil)
52: Entering state 1
53: Reducing stack by rule 7 (line 53):
54:    $1 = token IDENT (node6: IDENT 'c' nil nil)
55: AST: node6: IDENT 'c' nil nil
56: -> $$ = nterm factor ()
57: Stack now 0 5 10 2 7 9
58: Entering state 6
```

```
 59: Reducing stack by rule 5 (line 49):
 60:    $1 = nterm factor ()
 61: AST: node6: IDENT 'c' nil nil
 62: -> $$ = nterm term ()
 63: Stack now 0 5 10 2 7 9
 64: Entering state 12
 65: Reading a token: Next token is token RPAR (node7: RPAR ')' nil nil)
 66: Reducing stack by rule 2 (line 44):
 67:    $1 = nterm expr ()
 68:    $2 = token ADD (node5: ADD '+' nil nil)
 69:    $3 = nterm term ()
 70: AST: node5: ADD '+' node4 node6
 71: AST:    node4: IDENT 'b' nil nil
 72: AST:    node6: IDENT 'c' nil nil
 73: -> $$ = nterm expr ()
 74: Stack now 0 5 10 2
 75: Entering state 7
 76: Next token is token RPAR (node7: RPAR ')' nil nil)
 77: Shifting token RPAR (node7: RPAR ')' nil nil)
 78: Entering state 11
 79: Reducing stack by rule 6 (line 52):
 80:    $1 = token LPAR (node3: LPAR '(' nil nil)
 81:    $2 = nterm expr ()
 82:    $3 = token RPAR (node7: RPAR ')' nil nil)
 83: AST: node5: ADD '+' node4 node6
 84: AST:    node4: IDENT 'b' nil nil
 85: AST:    node6: IDENT 'c' nil nil
 86: -> $$ = nterm factor ()
 87: Stack now 0 5 10
 88: Entering state 13
 89: Reducing stack by rule 4 (line 48):
 90:    $1 = nterm term ()
 91:    $2 = token MUL (node2: MUL '*' nil nil)
 92:    $3 = nterm factor ()
 93: AST: node2: MUL '*' node1 node5
 94: AST:    node1: IDENT 'a' nil nil
 95: AST:    node5: ADD '+' node4 node6
 96: AST:       node4: IDENT 'b' nil nil
 97: AST:       node6: IDENT 'c' nil nil
 98: -> $$ = nterm term ()
 99: Stack now 0
100: Entering state 5
101: Reading a token: Now at end of input.
102: Reducing stack by rule 3 (line 45):
103:    $1 = nterm term ()
104: AST: node2: MUL '*' node1 node5
105: AST:    node1: IDENT 'a' nil nil
106: AST:    node5: ADD '+' node4 node6
107: AST:       node4: IDENT 'b' nil nil
108: AST:       node6: IDENT 'c' nil nil
109: -> $$ = nterm expr ()
110: Stack now 0
111: Entering state 4
112: Now at end of input.
113: Reducing stack by rule 1 (line 41):
114:    $1 = nterm expr ()
115: AST: node2: MUL '*' node1 node5
116: AST:    node1: IDENT 'a' nil nil
```

```
117: AST:    node5: ADD '+' node4 node6
118: AST:       node4: IDENT 'b' nil nil
119: AST:       node6: IDENT 'c' nil nil
120: -> $$ = nterm start ()
121: Stack now 0
122: Entering state 3
123: Now at end of input.
124: Shifting token $end (node8: $end '\x00' nil nil)
125: Entering state 8
126: Stack now 0 3 8
127: Cleanup: popping token $end (node8: $end '\x00' nil nil)
128: Cleanup: popping nterm start ()
129: Finished parse: status 0
130: Scanned input: "a*(b+c)"
131: Root of AST:
132: AST: node2: MUL '*' node1 node5
133: AST:    node1: IDENT 'a' nil nil
134: AST:    node5: ADD '+' node4 node6
135: AST:       node4: IDENT 'b' nil nil
136: AST:       node6: IDENT 'c' nil nil
```

```
 1: Scanning input: "f(c)"
 2: Starting parse
 3: Entering state 0
 4: Reading a token: Next token is token IDENT (node1: IDENT 'f' nil nil)
 5: Shifting token IDENT (node1: IDENT 'f' nil nil)
 6: Entering state 1
 7: Reducing stack by rule 7 (line 53):
 8:    $1 = token IDENT (node1: IDENT 'f' nil nil)
 9: AST: node1: IDENT 'f' nil nil
10: -> $$ = nterm factor ()
11: Stack now 0
12: Entering state 6
13: Reducing stack by rule 5 (line 49):
14:    $1 = nterm factor ()
15: AST: node1: IDENT 'f' nil nil
16: -> $$ = nterm term ()
17: Stack now 0
18: Entering state 5
19: Reading a token: Next token is token LPAR (node2: LPAR '(' nil nil)
20: Reducing stack by rule 3 (line 45):
21:    $1 = nterm term ()
22: AST: node1: IDENT 'f' nil nil
23: -> $$ = nterm expr ()
24: Stack now 0
25: Entering state 4
26: Next token is token LPAR (node2: LPAR '(' nil nil)
27: Reducing stack by rule 1 (line 41):
28:    $1 = nterm expr ()
29: AST: node1: IDENT 'f' nil nil
30: -> $$ = nterm start ()
31: Stack now 0
32: Entering state 3
33: Next token is token LPAR (node2: LPAR '(' nil nil)
34: syntax error, unexpected LPAR, expecting $end)
35: Error: popping nterm start ()
36: Stack now 0
37: Cleanup: discarding lookahead token LPAR (node2: LPAR '(' nil nil)
38: Stack now 0
39: Finished parse: status 1
40: Scanned input: "f(c)"
41: Root of AST:
42: AST: node1: IDENT 'f' nil nil
```

```
 1: # $Id: Makefile,v 1.3 2014-11-06 19:04:28-08 - - $
 2:
 3: GCC = g++ -g -O0 -Wall -Wextra -std=gnu++11
 4:
 5: all : etf
 6:
 7: etf : etf.tab.cc
 8:         ${GCC} etf.tab.cc -o etf
 9:
10: etf.tab.cc : etf.yy
11:         bison etf.yy
12:
13: ci :
14:         cid + Makefile etf.yy
15:
16: spotless : clean
17:         - rm Listing.{ps,pdf} test?.log
18:
19: clean :
20:         - rm etf.tab.cc etf.output etf
21:
22: test : etf
23:         ./etf "a*b+c*d" >test1.log 2>&1
24:         ./etf "a*(b+c)" >test2.log 2>&1
25:         ./etf "f(c)"  >test3.log 2>&1
26:
27: lis : test
28:         mkpspdf Listing.ps etf.yy etf.output test?.log Makefile
```