```
 1: // $Id: expr-recdes.cc,v 1.29 2014-10-09 18:04:35-07 - - $
 2:
 3: //
 4: // A trivial hand-coded top-down recursive descent compiler for
 5: // a simple language.  No leak checking.  Just crash on any
 6: // syntax error.
 7: //
 8: // Context-free syntax.
 9: // ->, {, }, |, ... are metasymbols
10: //
11: // program -> { expr ; }...
12: // expr    -> term { { + | - } term }...
13: // term    -> factor { { * | / } factor }...
14: // factor  -> ( expr ) | IDENT | NUMBER
15: //
16: // Lexical syntax.
17: // Using flex notation.
18: //
19: // IDENT   -> [A-Za-z][A-Za-z0-9]*
20: // NUMBER  -> [0-9]+
21: // COMMENT -> #.*
22: // WHITE   -> [ \n]+
23: //
24:
25: #include <string>
26: #include <unordered_map>
27: #include <vector>
28: using namespace std;
29:
30: #include <assert.h>
31: #include <ctype.h>
32: #include <libgen.h>
33: #include <stdarg.h>
34: #include <stdio.h>
35: #include <stdlib.h>
36: #include <string.h>
37:
38: enum { ENDFILE = 256, IDENT = 257, NUMBER = 258, ROOT = 259,
39:        NOSYMBOL = 260, };
40:
41: unordered_map<unsigned,string> symbol_names {
42:    {ENDFILE,  "ENDFILE"},
43:    {IDENT,    "IDENT"},
44:    {NUMBER,   "NUMBER"},
45:    {ROOT,     "ROOT"},
46:    {NOSYMBOL, "NOSYMBOL"},
47: };
48:
```

```
49:
50: void print_symbol (unsigned symbol) {
51:    printf ("%d", symbol);
52:    const auto& isymbol = symbol_names.find (symbol);
53:    if (isymbol != symbol_names.cend()) {
54:       printf ("(%s)", isymbol->second.c_str());
55:    }else if (isgraph (symbol)) {
56:       printf ("('%c')", symbol);
57:    }
58: }
59:
60: struct astree {
61:    unsigned symbol;
62:    string lexeme;
63:    vector<astree*> children;
64:    void adopt (astree* child) { children.push_back (child); }
65: };
66:
67: astree* new_astree (int symbol, const string& lexeme) {
68:    astree* tree = new astree();
69:    tree->symbol = symbol;
70:    tree->lexeme = lexeme;
71:    return tree;
72: }
73:
74: void preorder_astree (size_t depth, astree* tree) {
75:    for (size_t count = 0; count < depth; ++count) printf ("|  ");
76:    printf ("\"%s\" ", tree->lexeme.c_str());
77:    print_symbol (tree->symbol);
78:    printf ("\n");
79:    for (size_t child = 0; child < tree->children.size(); ++child) {
80:       preorder_astree (depth + 1, tree->children[child]);
81:    }
82: }
83:
84: void print_astree (const char* func, int line, astree* tree) {
85:    printf ("%s[%d]:\n", func, line);
86:    preorder_astree (1, tree);
87: }
88: #define PRINT_ASTREE(TREE) print_astree (__func__, __LINE__, TREE)
89:
```

```
 90:
 91: int peekchar = 0;
 92: astree* lookahead_token = NULL;
 93:
 94: int isnt_nl (int achar) { return achar != '\n' && achar != EOF; }
 95:
 96: string scan_chars (int (*ischar) (int)) {
 97:    string lexeme;
 98:    do {
 99:       assert (peekchar != EOF);
100:       lexeme += peekchar;
101:       peekchar = getchar();
102:    }while (ischar && ischar (peekchar));
103:    return lexeme;
104: }
105:
106: void scan_new_token (int symbol, int (*ischar) (int)) {
107:    string lexeme = symbol != ENDFILE ? scan_chars (ischar) : "<<EOF>>";
108:    lookahead_token = new_astree (symbol, lexeme);
109:    PRINT_ASTREE (lookahead_token);
110: }
111:
112: void scan_lookahead_token (void) {
113:    for (;;) {
114:       if (isalpha (peekchar)) {
115:          scan_new_token (IDENT, isalnum);
116:          return;
117:       }else if (isdigit (peekchar)) {
118:          scan_new_token (NUMBER, isdigit);
119:          return;
120:       }else {
121:          switch (peekchar) {
122:             case ' ':
123:             case '\n':
124:                scan_chars (isspace);
125:                continue;
126:             case '#':
127:                scan_chars (isnt_nl);
128:                continue;
129:             case '+':
130:             case '-':
131:             case '*':
132:             case '/':
133:             case '(':
134:             case ')':
135:             case ';':
136:                scan_new_token (peekchar, NULL);
137:                return;
138:             case EOF:
139:                scan_new_token (ENDFILE, NULL);
140:                return;
141:          }
142:       }
143:       assert (false);
144:    }
145: }
146:
```

```
147:
148: astree* parse_expr (void);
149:
150: astree* parse_factor (void) {
151:    astree* tree = NULL;
152:    switch (lookahead_token->symbol) {
153:       case '(':
154:          scan_lookahead_token();
155:          tree = parse_expr();
156:          assert (lookahead_token->symbol == ')');
157:          scan_lookahead_token();
158:          break;
159:       case IDENT:
160:       case NUMBER:
161:          tree = lookahead_token;
162:          scan_lookahead_token();
163:          break;
164:       default:
165:          assert (lookahead_token->symbol == NOSYMBOL);
166:    }
167:    PRINT_ASTREE (tree);
168:    return tree;
169: }
170:
171: astree* parse_term (void) {
172:    astree* root = parse_factor();
173:    for (;;) {
174:       switch (lookahead_token->symbol) {
175:          case '*':
176:          case '/': {
177:             astree* oper = lookahead_token;
178:             scan_lookahead_token();
179:             oper->adopt (root);
180:             oper->adopt (parse_factor());
181:             root = oper;
182:             break;
183:          }
184:          default:
185:             PRINT_ASTREE (root);
186:             return root;
187:       }
188:    }
189: }
190:
```

```
191:
192: astree* parse_expr (void) {
193:    astree* root = parse_term();
194:    for (;;) {
195:       switch (lookahead_token->symbol) {
196:          case '+':
197:          case '-': {
198:             astree* oper = lookahead_token;
199:             scan_lookahead_token();
200:             oper->adopt (root);
201:             oper->adopt (parse_term());
202:             root = oper;
203:             break;
204:          }
205:          default:
206:             PRINT_ASTREE (root);
207:             return root;
208:       }
209:    }
210: }
211:
212: void parse_program (void) {
213:    astree* root = new_astree (ROOT, "<<ROOT>>");
214:    while (lookahead_token->symbol != ENDFILE) {
215:       astree* tree = parse_expr();
216:       printf ("\f\n");
217:       assert (lookahead_token->symbol == ';');
218:       scan_lookahead_token();
219:       root->adopt (tree);
220:    }
221:    PRINT_ASTREE (root);
222: }
223:
224: int main (void) {
225:    peekchar = getchar();
226:    scan_lookahead_token();
227:    parse_program();
228:    assert (lookahead_token->symbol == ENDFILE);
229:    return EXIT_SUCCESS;
230: }
231:
```

```
 1: # $Id: Makefile,v 1.36 2014-10-09 18:07:20-07 - - $
 2:
 3: GCC    = g++ -g -O0 -Wall -Wextra -std=gnu++0x
 4: EXEC   = expr-recdes
 5: LIST   = ${EXEC}.cc Makefile test.in test.out
 6:
 7: all : ${EXEC}
 8:
 9: % : %.cc
10:         ${GCC} $< -o $@
11:
12: test.out: ${EXEC} test.in
13:         ${EXEC} <test.in >test.out 2>&1
14:         if [ -f core ] ; then rm core ; fi
15:
16: ci :
17:         checksource ${filter-out test.out, ${LIST}}
18:         cid + expr-recdes.cc Makefile test.in
19:
20: lis : test.out
21:         killps gv
22:         mkpspdf Listing.ps ${LIST}
23:
```

```
1: # $Id: test.in,v 1.3 2009-10-01 17:51:46-07 - - $
2: foo * bar + baz;
3: 34 * 55 * 66;
4: add + this + and + this;
5: abc * (def - ghi);
6: abc + def / ghi - jkl;
7: foo * bar + baz / qux;
```

```
 1: scan_new_token[109]:
 2: |  "foo" 257(IDENT)
 3: scan_new_token[109]:
 4: |  "*" 42('*')
 5: parse_factor[167]:
 6: |  "foo" 257(IDENT)
 7: scan_new_token[109]:
 8: |  "bar" 257(IDENT)
 9: scan_new_token[109]:
10: |  "+" 43('+')
11: parse_factor[167]:
12: |  "bar" 257(IDENT)
13: parse_term[185]:
14: |  "*" 42('*')
15: |  |  "foo" 257(IDENT)
16: |  |  "bar" 257(IDENT)
17: scan_new_token[109]:
18: |  "baz" 257(IDENT)
19: scan_new_token[109]:
20: |  ";" 59(';')
21: parse_factor[167]:
22: |  "baz" 257(IDENT)
23: parse_term[185]:
24: |  "baz" 257(IDENT)
25: parse_expr[206]:
26: |  "+" 43('+')
27: |  |  "*" 42('*')
28: |  |  |  "foo" 257(IDENT)
29: |  |  |  "bar" 257(IDENT)
30: |  |  "baz" 257(IDENT)
```

```
31:
32: scan_new_token[109]:
33: |  "34" 258(NUMBER)
34: scan_new_token[109]:
35: |  "*" 42('*')
36: parse_factor[167]:
37: |  "34" 258(NUMBER)
38: scan_new_token[109]:
39: |  "55" 258(NUMBER)
40: scan_new_token[109]:
41: |  "*" 42('*')
42: parse_factor[167]:
43: |  "55" 258(NUMBER)
44: scan_new_token[109]:
45: |  "66" 258(NUMBER)
46: scan_new_token[109]:
47: |  ";" 59(';')
48: parse_factor[167]:
49: |  "66" 258(NUMBER)
50: parse_term[185]:
51: |  "*" 42('*')
52: |  |  "*" 42('*')
53: |  |  |  "34" 258(NUMBER)
54: |  |  |  "55" 258(NUMBER)
55: |  |  "66" 258(NUMBER)
56: parse_expr[206]:
57: |  "*" 42('*')
58: |  |  "*" 42('*')
59: |  |  |  "34" 258(NUMBER)
60: |  |  |  "55" 258(NUMBER)
61: |  |  "66" 258(NUMBER)
```

```
 62:
 63: scan_new_token[109]:
 64: |  "add" 257(IDENT)
 65: scan_new_token[109]:
 66: |  "+" 43('+')
 67: parse_factor[167]:
 68: |  "add" 257(IDENT)
 69: parse_term[185]:
 70: |  "add" 257(IDENT)
 71: scan_new_token[109]:
 72: |  "this" 257(IDENT)
 73: scan_new_token[109]:
 74: |  "+" 43('+')
 75: parse_factor[167]:
 76: |  "this" 257(IDENT)
 77: parse_term[185]:
 78: |  "this" 257(IDENT)
 79: scan_new_token[109]:
 80: |  "and" 257(IDENT)
 81: scan_new_token[109]:
 82: |  "+" 43('+')
 83: parse_factor[167]:
 84: |  "and" 257(IDENT)
 85: parse_term[185]:
 86: |  "and" 257(IDENT)
 87: scan_new_token[109]:
 88: |  "this" 257(IDENT)
 89: scan_new_token[109]:
 90: |  ";" 59(';')
 91: parse_factor[167]:
 92: |  "this" 257(IDENT)
 93: parse_term[185]:
 94: |  "this" 257(IDENT)
 95: parse_expr[206]:
 96: |  "+" 43('+')
 97: |  |  "+" 43('+')
 98: |  |  |  "+" 43('+')
 99: |  |  |  |  "add" 257(IDENT)
100: |  |  |  |  "this" 257(IDENT)
101: |  |  |  "and" 257(IDENT)
102: |  |  "this" 257(IDENT)
```

```
103:
104: scan_new_token[109]:
105: |  "abc" 257(IDENT)
106: scan_new_token[109]:
107: |  "*" 42('*')
108: parse_factor[167]:
109: |  "abc" 257(IDENT)
110: scan_new_token[109]:
111: |  "(" 40('(')
112: scan_new_token[109]:
113: |  "def" 257(IDENT)
114: scan_new_token[109]:
115: |  "-" 45('-')
116: parse_factor[167]:
117: |  "def" 257(IDENT)
118: parse_term[185]:
119: |  "def" 257(IDENT)
120: scan_new_token[109]:
121: |  "ghi" 257(IDENT)
122: scan_new_token[109]:
123: |  ")" 41(')')
124: parse_factor[167]:
125: |  "ghi" 257(IDENT)
126: parse_term[185]:
127: |  "ghi" 257(IDENT)
128: parse_expr[206]:
129: |  "-" 45('-')
130: |  |  "def" 257(IDENT)
131: |  |  "ghi" 257(IDENT)
132: scan_new_token[109]:
133: |  ";" 59(';')
134: parse_factor[167]:
135: |  "-" 45('-')
136: |  |  "def" 257(IDENT)
137: |  |  "ghi" 257(IDENT)
138: parse_term[185]:
139: |  "*" 42('*')
140: |  |  "abc" 257(IDENT)
141: |  |  "-" 45('-')
142: |  |  |  "def" 257(IDENT)
143: |  |  |  "ghi" 257(IDENT)
144: parse_expr[206]:
145: |  "*" 42('*')
146: |  |  "abc" 257(IDENT)
147: |  |  "-" 45('-')
148: |  |  |  "def" 257(IDENT)
149: |  |  |  "ghi" 257(IDENT)
```

```
150:
151: scan_new_token[109]:
152: |  "abc" 257(IDENT)
153: scan_new_token[109]:
154: |  "+" 43('+')
155: parse_factor[167]:
156: |  "abc" 257(IDENT)
157: parse_term[185]:
158: |  "abc" 257(IDENT)
159: scan_new_token[109]:
160: |  "def" 257(IDENT)
161: scan_new_token[109]:
162: |  "/" 47('/')
163: parse_factor[167]:
164: |  "def" 257(IDENT)
165: scan_new_token[109]:
166: |  "ghi" 257(IDENT)
167: scan_new_token[109]:
168: |  "-" 45('-')
169: parse_factor[167]:
170: |  "ghi" 257(IDENT)
171: parse_term[185]:
172: |  "/" 47('/')
173: |  |  "def" 257(IDENT)
174: |  |  "ghi" 257(IDENT)
175: scan_new_token[109]:
176: |  "jkl" 257(IDENT)
177: scan_new_token[109]:
178: |  ";" 59(';')
179: parse_factor[167]:
180: |  "jkl" 257(IDENT)
181: parse_term[185]:
182: |  "jkl" 257(IDENT)
183: parse_expr[206]:
184: |  "-" 45('-')
185: |  |  "+" 43('+')
186: |  |  |  "abc" 257(IDENT)
187: |  |  |  "/" 47('/')
188: |  |  |  |  "def" 257(IDENT)
189: |  |  |  |  "ghi" 257(IDENT)
190: |  |  "jkl" 257(IDENT)
```

```
191:
192: scan_new_token[109]:
193: |  "foo" 257(IDENT)
194: scan_new_token[109]:
195: |  "*" 42('*')
196: parse_factor[167]:
197: |  "foo" 257(IDENT)
198: scan_new_token[109]:
199: |  "bar" 257(IDENT)
200: scan_new_token[109]:
201: |  "+" 43('+')
202: parse_factor[167]:
203: |  "bar" 257(IDENT)
204: parse_term[185]:
205: |  "*" 42('*')
206: |  |  "foo" 257(IDENT)
207: |  |  "bar" 257(IDENT)
208: scan_new_token[109]:
209: |  "baz" 257(IDENT)
210: scan_new_token[109]:
211: |  "/" 47('/')
212: parse_factor[167]:
213: |  "baz" 257(IDENT)
214: scan_new_token[109]:
215: |  "qux" 257(IDENT)
216: scan_new_token[109]:
217: |  ";" 59(';')
218: parse_factor[167]:
219: |  "qux" 257(IDENT)
220: parse_term[185]:
221: |  "/" 47('/')
222: |  |  "baz" 257(IDENT)
223: |  |  "qux" 257(IDENT)
224: parse_expr[206]:
225: |  "+" 43('+')
226: |  |  "*" 42('*')
227: |  |  |  "foo" 257(IDENT)
228: |  |  |  "bar" 257(IDENT)
229: |  |  "/" 47('/')
230: |  |  |  "baz" 257(IDENT)
231: |  |  |  "qux" 257(IDENT)
```

```
232:
233: scan_new_token[109]:
234: |  "<<EOF>>" 256(ENDFILE)
235: parse_program[221]:
236: |  "<<ROOT>>" 259(ROOT)
237: |  |  "+" 43('+')
238: |  |  |  "*" 42('*')
239: |  |  |  |  "foo" 257(IDENT)
240: |  |  |  |  "bar" 257(IDENT)
241: |  |  |  "baz" 257(IDENT)
242: |  |  "*" 42('*')
243: |  |  |  "*" 42('*')
244: |  |  |  |  "34" 258(NUMBER)
245: |  |  |  |  "55" 258(NUMBER)
246: |  |  |  "66" 258(NUMBER)
247: |  |  "+" 43('+')
248: |  |  |  "+" 43('+')
249: |  |  |  |  "+" 43('+')
250: |  |  |  |  |  "add" 257(IDENT)
251: |  |  |  |  |  "this" 257(IDENT)
252: |  |  |  |  "and" 257(IDENT)
253: |  |  |  "this" 257(IDENT)
254: |  |  "*" 42('*')
255: |  |  |  "abc" 257(IDENT)
256: |  |  |  "-" 45('-')
257: |  |  |  |  "def" 257(IDENT)
258: |  |  |  |  "ghi" 257(IDENT)
259: |  |  "-" 45('-')
260: |  |  |  "+" 43('+')
261: |  |  |  |  "abc" 257(IDENT)
262: |  |  |  |  "/" 47('/')
263: |  |  |  |  |  "def" 257(IDENT)
264: |  |  |  |  |  "ghi" 257(IDENT)
265: |  |  |  "jkl" 257(IDENT)
266: |  |  "+" 43('+')
267: |  |  |  "*" 42('*')
268: |  |  |  |  "foo" 257(IDENT)
269: |  |  |  |  "bar" 257(IDENT)
270: |  |  |  "/" 47('/')
271: |  |  |  |  "baz" 257(IDENT)
272: |  |  |  |  "qux" 257(IDENT)
```