```
 1:
 2: Simple compiler:  Translate exprs to stack machine insns.
 3:
 4: Syntax:     the ETF grammar
 5: Lexical:    identifiers, numbers
 6: Comments:   // and /**/ C-style
 7: Directives: #-cpp style
 8: Activity:   Build AST
 9: Codegen:    Stack machine code
10:
11: $Id: README,v 1.2 2011-08-31 17:54:03-07 - - $
12:
```

```
 1: # $Id: Makefile,v 1.28 2013-08-22 13:59:59-07 - - $
 2:
 3: MKFILE    = Makefile
 4: DEPSFILE  = ${MKFILE}.deps
 5: NOINCLUDE = ci clean spotless
 6: NEEDINCL  = ${filter ${NOINCLUDE}, ${MAKECMDGOALS}}
 7:
 8: #
 9: # Definitions of list of files:
10: #
11: HSOURCES  = astree.h astree.rep.h emit.h lyutils.h auxlib.h
12: CSOURCES  = astree.c emit.c lyutils.c main.c auxlib.c
13: LSOURCES  = scanner.l
14: YSOURCES  = parser.y
15: ETCSRC    = README ${MKFILE} ${DEPSFILE}
16: CLGEN     = yylex.c
17: HYGEN     = yyparse.h
18: CYGEN     = yyparse.c
19: CGENS     = ${CLGEN} ${CYGEN}
20: ALLGENS   = ${HYGEN} ${CGENS}
21: EXECBIN   = zexprsm
22: ALLCSRC   = ${CSOURCES} ${CGENS}
23: OBJECTS   = ${ALLCSRC:.c=.o}
24: LREPORT   = yylex.output
25: YREPORT   = yyparse.output
26: IREPORT   = ident.output
27: REPORTS   = ${LREPORT} ${YREPORT} ${IREPORT}
28: ALLSRC    = ${ETCSRC} ${YSOURCES} ${LSOURCES} ${HSOURCES} ${CSOURCES}
29: TESTINS   = ${wildcard test?.in}
30: LISTSRC   = ${ALLSRC} ${HYGEN}
31:
32: #
33: # Definitions of the compiler and compilation options:
34: #
35: GCC       = gcc -g -O0 -Wall -Wextra -std=gnu99
36: MKDEPS    = gcc -MM
37:
38: #
39: # The first target is always ``all'', and hence the default,
40: # and builds the executable images
41: #
42: all : ${EXECBIN}
43:
44: #
45: # Build the executable image from the object files.
46: #
47: ${EXECBIN} : ${OBJECTS}
48:         ${GCC} -o${EXECBIN} ${OBJECTS}
49:         ident ${OBJECTS} ${EXECBIN} >${IREPORT}
50:
51: #
52: # Build an object file form a C source file.
53: #
54: %.o : %.c
55:         cid + $<
56:         ${GCC} -c $<
57:
```

```
58:
59: #
60: # Build the scanner.
61: #
62: ${CLGEN} : ${LSOURCES}
63:         flex -o${CLGEN} ${LSOURCES} 2>${LREPORT}
64:         - grep -v '^  ' ${LREPORT}
65:         - (perl -e 'print "="x65,"\n"'; cat lex.backup) >>${LREPORT}
66:         - rm lex.backup
67:
68: #
69: # Build the parser.
70: #
71: ${CYGEN} ${HYGEN} : ${YSOURCES}
72:         bison -dtv -o${CYGEN} ${YSOURCES}
73:
74: #
75: # Check sources into an RCS subdirectory.
76: #
77: ci : ${ALLSRC} ${TESTINS}
78:         cid + ${ALLSRC} ${TESTINS} test?.inh
79:
80: #
81: # Make a listing from all of the sources
82: #
83: lis : ${LISTSRC} tests
84:         mkpspdf List.source.ps ${LISTSRC}
85:         mkpspdf List.output.ps ${REPORTS} \
86:                 ${foreach test, ${TESTINS:.in=}, \
87:                 ${patsubst %, ${test}.%, in out err log}}
88:
89: #
90: # Clean and spotless remove generated files.
91: #
92: clean :
93:         - rm ${OBJECTS} ${ALLGENS} ${REPORTS} ${DEPSFILE} core
94:         - rm ${foreach test, ${TESTINS:.in=}, \
95:                 ${patsubst %, ${test}.%, out err log}}
96:
97: spotless : clean
98:         - rm ${EXECBIN} List.*.ps List.*.pdf
99:
```

```
100:
101: #
102: # Build the dependencies file using the C preprocessor
103: #
104: deps : ${ALLCSRC}
105:         @ echo "# ${DEPSFILE} created `date` by ${MAKE}" >${DEPSFILE}
106:         ${MKDEPS} ${ALLCSRC} >>${DEPSFILE}
107:
108: ${DEPSFILE} :
109:         @ touch ${DEPSFILE}
110:         ${MAKE} --no-print-directory deps
111:
112: #
113: # Test
114: #
115:
116: tests : ${EXECBIN} ${TESTINS:.in=.out}
117:
118: %.out %.err %.log : %.in ${EXECBIN}
119:         runprog -o$* ${EXECBIN} -@@ -ly -de $<
120:
121: #
122: # Everything
123: #
124: again :
125:         gmake --no-print-directory spotless deps ci all lis
126:
127: ifeq "${NEEDINCL}" ""
128: include ${DEPSFILE}
129: endif
130:
```

```
1: # Makefile.deps created Wed Sep 11 20:17:48 PDT 2013 by gmake
2: astree.o: astree.c astree.h auxlib.h astree.rep.h lyutils.h yyparse.h
3: emit.o: emit.c astree.h auxlib.h astree.rep.h emit.h lyutils.h yyparse.h
4: lyutils.o: lyutils.c astree.rep.h astree.h auxlib.h lyutils.h yyparse.h
5: main.o: main.c astree.h auxlib.h emit.h lyutils.h yyparse.h
6: auxlib.o: auxlib.c auxlib.h
7: yylex.o: yylex.c auxlib.h lyutils.h astree.h yyparse.h
8: yyparse.o: yyparse.c lyutils.h astree.h auxlib.h yyparse.h astree.rep.h
```

```
%{
// $Id: parser.y,v 1.9 2013-08-22 13:58:43-07 - - $

#include <assert.h>
#include <stdlib.h>
#include <string.h>

#include "lyutils.h"
#include "astree.h"
#include "astree.rep.h"

#define YYDEBUG 1
#define YYERROR_VERBOSE 1
#define YYPRINT yyprint
#define YYMALLOC yycalloc

static void *yycalloc (size_t size);

%}

%debug
%defines
%error-verbose
%token-table
%verbose

%token  ROOT IDENT NUMBER

%right  '='
%left   '+' '-'
%left   '*' '/'
%right  '^'
%right  POS "u+" NEG "u-"

%start  program

%%

program : stmtseq               { $$ = $1; }
        ;

stmtseq : stmtseq expr ';'      { freeast ($3); $$ = adopt1 ($1, $2); }
        | stmtseq error ';'     { freeast ($3); $$ = $1; }
        | stmtseq ';'           { freeast ($2); $$ = $1; }
        |                       { $$ = new_parseroot(); }
        ;

expr    : expr '=' expr         { $$ = adopt2 ($2, $1, $3); }
        | expr '+' expr         { $$ = adopt2 ($2, $1, $3); }
        | expr '-' expr         { $$ = adopt2 ($2, $1, $3); }
        | expr '*' expr         { $$ = adopt2 ($2, $1, $3); }
        | expr '/' expr         { $$ = adopt2 ($2, $1, $3); }
        | expr '^' expr         { $$ = adopt2 ($2, $1, $3); }
        | '+' expr %prec POS    { $$ = adopt1sym ($1, $2, POS); }
        | '-' expr %prec NEG    { $$ = adopt1sym ($1, $2, NEG); }
        | '(' expr ')'          { freeast2 ($1, $3); $$ = $2; }
        | IDENT                 { $$ = $1; }
        | NUMBER                { $$ = $1; }
        ;

```

```
61:
62: %%
63:
64: const char *get_yytname (int symbol) {
65:    return yytname [YYTRANSLATE (symbol)];
66: }
67:
68: static void *yycalloc (size_t size) {
69:    void *result = calloc (1, size);
70:    assert (result != NULL);
71:    return result;
72: }
73:
74: RCSC("$Id: parser.y,v 1.9 2013-08-22 13:58:43-07 - - $")
75:
```

```
%{
// $Id: scanner.l,v 1.11 2013-09-11 20:17:48-07 - - $

#include <stdlib.h>
#include <stdbool.h>

#include "auxlib.h"
#include "lyutils.h"

#define YY_USER_ACTION  { scanner_useraction (); }
#define IGNORE(THING)   { }

%}

%option 8bit
%option backup
%option debug
%option ecs
%option nodefault
%option nounput
%option noyywrap
%option perf-report
%option verbose
%option warn

LETTER          [A-Za-z_]
DIGIT           [0-9]
MANTISSA        ({DIGIT}+\.?{DIGIT}*|\.{DIGIT}+)
EXPONENT        ([Ee][+-]?{DIGIT}+)
NUMBER          ({MANTISSA}{EXPONENT}?)
NOTNUMBER       ({MANTISSA}[Ee][+-]?)
IDENT           ({LETTER}({LETTER}|{DIGIT})*)

%%

"#".*           { scanner_include(); }
[ \t]+          { IGNORE (white space) }
\n              { scanner_newline(); }

{NUMBER}        { return yylval_token (NUMBER); }
{IDENT}         { return yylval_token (IDENT); }

"="             { return yylval_token ('='); }
"+"             { return yylval_token ('+'); }
"-"             { return yylval_token ('-'); }
"*"             { return yylval_token ('*'); }
"/"             { return yylval_token ('/'); }
"^"             { return yylval_token ('^'); }
"("             { return yylval_token ('('); }
")"             { return yylval_token (')'); }
";"             { return yylval_token (';'); }

{NOTNUMBER}     { scanner_badtoken (yytext);
                  return yylval_token (NUMBER); }

.               { scanner_badchar (*yytext); }

%%
```

```
59:
60:
61: RCSC("$Id: scanner.l,v 1.11 2013-09-11 20:17:48-07 - - $")
62:
```

```
 1: #ifndef __ASTREE_H__
 2: #define __ASTREE_H__
 3:
 4: #include <stdbool.h>
 5:
 6: #include "auxlib.h"
 7:
 8: typedef struct astree_rep *astree;
 9:
10: bool is_astree (void *object);
11: astree new_astree (int symbol, int filenr, int linenr, int offset,
12:                    char *lexinfo);
13: astree adopt (astree root, /*ASTree*/ ... /*, NULL */);
14: astree adopt1 (astree root, astree child);
15: astree adopt2 (astree root, astree left, astree right);
16: astree adopt1sym (astree root, astree child, int symbol);
17: void dump_astree (FILE *outfile, astree root);
18: void yyprint (FILE *outfile, unsigned short toknum, astree yyvaluep);
19: void freeast (astree tree);
20:
21: #define freeast2(T1,T2) { freeast (T1); freeast (T2); }
22:
23: RCSH("$Id: astree.h,v 1.8 2013-08-22 13:58:43-07 - - $")
24: #endif
```

```
 1: #ifndef __ASTREEREP_H__
 2: #define __ASTREEREP_H__
 3:
 4: #include "astree.h"
 5:
 6: struct astree_rep {
 7:    char *tag;           // tag field to verify class membership
 8:    int symbol;          // token code
 9:    int filenr;          // index into filename stack
10:    int linenr;          // line number from source code
11:    int offset;          // offset of token with current line
12:    char *lexinfo;       // pointer to lexical information
13:    astree first;        // first child node of this node
14:    astree last;         // last child node of this node
15:    astree next;         // next younger sibling of this node
16: };
17:
18: RCSH("$Id: astree.rep.h,v 1.5 2013-08-22 13:58:43-07 - - $")
19: #endif
```

```
1: #ifndef __EMIT_H__
2: #define __EMIT_H__
3:
4: #include "astree.h"
5:
6: void emit_sm_code (astree);
7:
8: RCSH("$Id: emit.h,v 1.3 2013-08-22 13:58:43-07 - - $")
9: #endif
```

```
 1: #ifndef __LYUTILS_H__
 2: #define __LYUTILS_H__
 3:
 4: // Lex and Yacc interface utility.
 5:
 6: #include <stdio.h>
 7:
 8: #include "astree.h"
 9: #include "auxlib.h"
10:
11: #define YYEOF 0
12:
13: extern FILE *yyin;
14: extern astree yyparse_astree;
15: extern int yyin_linenr;
16: extern char *yytext;
17: extern int yy_flex_debug;
18: extern int yydebug;
19: extern int yyleng;
20:
21: int yylex (void);
22: int yyparse (void);
23: void yyerror (char *message);
24: const char *get_yytname (int symbol);
25:
26: char *scanner_filename (int filenr);
27: void scanner_newfilename (char *filename);
28: void scanner_badchar (unsigned char bad);
29: void scanner_badtoken (char *lexeme);
30: void scanner_newline (void);
31: void scanner_setecho (bool echoflag);
32: void scanner_useraction (void);
33:
34: astree new_parseroot (void);
35: int yylval_token (int symbol);
36:
37: void scanner_include (void);
38:
39: #define YYSTYPE astree
40: #include "yyparse.h"
41:
42: RCSH("$Id: lyutils.h,v 1.10 2013-08-22 13:58:43-07 - - $")
43: #endif
```

```
 1: #ifndef __AUXLIB_H__
 2: #define __AUXLIB_H__
 3:
 4: #include <stdarg.h>
 5:
 6: //
 7: // DESCRIPTION
 8: //    Auxiliary library containing miscellaneous useful things.
 9: //
10:
11: //
12: // Error message and exit status utility.
13: //
14:
15: void set_execname (char *argv0);
16:    //
17:    // Sets the program name for use by auxlib messages.
18:    // Must called from main before anything else is done,
19:    // passing in argv[0].
20:    //
21:
22: char *get_execname (void);
23:    //
24:    // Returns a read-only value previously stored by set_progname.
25:    //
26:
27: void eprint_status (char *command, int status);
28:    //
29:    // Print the status returned by wait(2) from a subprocess.
30:    //
31:
32: int get_exitstatus (void);
33:    //
34:    // Returns the exit status.  Default is EXIT_SUCCESS unless
35:    // set_exitstatus (int) is called.  The last statement in main
36:    // should be:  ``return get_exitstatus();''.
37:    //
38:
39: void set_exitstatus (int);
40:    //
41:    // Sets the exit status.  Remebers only the largest value passed in.
42:    //
43:
```

```
44:
45: void veprintf (char *format, va_list args);
46:    //
47:    // Prints a message to stderr using the vector form of
48:    // argument list.
49:    //
50:
51: void eprintf (char *format, ...);
52:    //
53:    // Print a message to stderr according to the printf format
54:    // specified.  Usually called for debug output.
55:    // Precedes the message by the program name if the format
56:    // begins with the characters `%:'.
57:    //
58:
59: void errprintf (char *format, ...);
60:    //
61:    // Print an error message according to the printf format
62:    // specified, using eprintf.  Sets the exitstatus to EXIT_FAILURE.
63:    //
64:
65: void syserrprintf (char *object);
66:    //
67:    // Print a message resulting from a bad system call.  The
68:    // object is the name of the object causing the problem and
69:    // the reason is taken from the external variable errno.
70:    //
71:
```

```
 72:
 73: //
 74: // Support for stub messages.
 75: //
 76: #define STUBPRINTF(...) \
 77:         __stubprintf (__FILE__, __LINE__, __func__, __VA_ARGS__)
 78: void __stubprintf (char *file, int line, const char *func,
 79:                    char *format, ...);
 80:
 81: //
 82: // Debugging utility.
 83: //
 84:
 85: void set_debugflags (char *flags);
 86:    //
 87:    // Sets a string of debug flags to be used by DEBUGF statements.
 88:    // Uses the address of the string, and does not copy it, so it
 89:    // must not be dangling.  If a particular debug flag has been set,
 90:    // messages are printed.  The format is identical to printf format.
 91:    // The flag "@" turns on all flags.
 92:    //
 93:
 94: bool is_debugflag (char flag);
 95:    //
 96:    // Checks to see if a debugflag is set.
 97:    //
 98:
 99: #ifdef NDEBUG
100: // Do not generate any code.
101: #define DEBUGF(FLAG,...)   /**/
102: #define DEBUGSTMT(FLAG,STMTS) /**/
103: #else
104: // Generate debugging code.
105: void __debugprintf (char flag, char *file, int line, const char *func,
106:                     char *format, ...);
107: #define DEBUGF(FLAG,...) \
108:         __debugprintf (FLAG, __FILE__, __LINE__, __func__, __VA_ARGS__)
109: #define DEBUGSTMT(FLAG,STMTS) \
110:         if (is_debugflag (FLAG)) { DEBUGF (FLAG, "\n"); STMTS }
111: #endif
112:
113: //
114: // Definition of RCSID macro to include RCS info in objs and execbin.
115: //
116:
117: #define RCS3(ID,N,X) static const char ID##N[] = X;
118: #define RCS2(N,X) RCS3(RCS_Id,N,X)
119: #define RCSH(X) RCS2(__COUNTER__,X)
120: #define RCSC(X) RCSH(X \
121: "\0$Compiled: " __FILE__ " " __DATE__ " " __TIME__ " $")
122: RCSH("$Id: auxlib.h,v 1.10 2013-08-22 13:58:43-07 - - $")
123: #endif
```

```
 1:
 2: #include <assert.h>
 3: #include <inttypes.h>
 4: #include <stdarg.h>
 5: #include <stdio.h>
 6: #include <stdlib.h>
 7: #include <string.h>
 8:
 9: #include "astree.h"
10: #include "astree.rep.h"
11: #include "lyutils.h"
12:
13: static char *astree_tag = "struct astree_rep";
14:
15: bool is_astree (void *object) {
16:    astree tree = (astree) object;
17:    return tree != NULL && tree->tag == astree_tag;
18: }
19:
20: astree new_astree (int symbol, int filenr, int linenr, int offset,
21:                    char *lexinfo) {
22:    size_t size = sizeof (struct astree_rep);
23:    astree tree = malloc (size);
24:    assert (tree != NULL);
25:    tree->tag = astree_tag;
26:    tree->symbol = symbol;
27:    tree->filenr = filenr;
28:    tree->linenr = linenr;
29:    tree->offset = offset;
30:    tree->lexinfo = strdup (lexinfo);
31:    assert (tree->lexinfo != NULL);
32:    tree->first = NULL;
33:    tree->last = NULL;
34:    tree->next = NULL;
35:    DEBUGF ('f', "malloc (%d) = %p-> %d:%d.%d: %s: %p->\"%s\"\n",
36:            size, tree, tree->filenr, tree->linenr, tree->offset,
37:            get_yytname (tree->symbol), tree->lexinfo, tree->lexinfo);
38:    return tree;
39: }
40:
```

```
41:
42: astree adopt (astree root, ...) {
43:    va_list children;
44:    assert (is_astree (root));
45:    va_start (children, root);
46:    for(;;) {
47:       astree child = va_arg (children, astree);
48:       if (child == NULL) break;
49:       assert (is_astree (child));
50:       if (root->last == NULL) root->first = child;
51:                          else root->last->next = child;
52:       root->last = child;
53:       DEBUGF ('a', "%p (%s) adopting %p (%s)\n",
54:               root, root->lexinfo,
55:               child, child->lexinfo);
56:    }
57:    va_end (children);
58:    return root;
59: }
60:
61: astree adopt2 (astree root, astree left, astree right) {
62:    return adopt (root, left, right, NULL);
63: }
64:
65: astree adopt1 (astree root, astree child) {
66:    return adopt (root, child, NULL);
67: }
68:
69: astree adopt1sym (astree root, astree child, int symbol) {
70:    root = adopt1 (root, child);
71:    root->symbol = symbol;
72:    return root;
73: }
74:
```

```
 75:
 76: static void dump_node (FILE *outfile, astree node, int depth) {
 77:    assert (is_astree (node));
 78:    fprintf (outfile, "%p-> astree {%s(%d), %d:%d.%03d, %p->\"%s\",\n",
 79:             (void*) node, get_yytname (node->symbol), node->symbol,
 80:             node->filenr, node->linenr, node->offset,
 81:             node->lexinfo, node->lexinfo);
 82:    fprintf (outfile, "%*sfirst=%p, last=%p, next=%p}",
 83:             depth * 3 + 12, "", (void*) node->first,
 84:             (void*) node->last, (void*) node->next);
 85: }
 86:
 87: static void dump_astree_rec (FILE *outfile, astree root, int depth) {
 88:    astree child = NULL;
 89:    if (root == NULL) return;
 90:    assert (is_astree (root));
 91:    fprintf (outfile, "%*s%s ", depth * 3, "", root->lexinfo);
 92:    dump_node (outfile, root, depth);
 93:    fprintf (outfile, "\n");
 94:    for (child = root->first; child != NULL; child = child->next) {
 95:       dump_astree_rec (outfile, child, depth + 1);
 96:    }
 97: }
 98:
 99: void dump_astree (FILE *outfile, astree root) {
100:    dump_astree_rec (outfile, root, 0);
101:    fflush (NULL);
102: }
103:
104: void yyprint (FILE *outfile, unsigned short toknum, astree yyvaluep) {
105:    fprintf (outfile, "%d=%s)\n%*s(",
106:             toknum, get_yytname (toknum), 9, "");
107:    if (is_astree (yyvaluep)) {
108:       dump_node (outfile, yyvaluep, 3);
109:    }else{
110:       fprintf (outfile, "yyvaluep = %p", (void*) yyvaluep);
111:    }
112:    fflush (NULL);
113: }
114:
115: void freeast (astree root) {
116:    astree child = NULL;
117:    if (root == NULL) return;
118:    assert (is_astree (root));
119:    for (child = root->first; child != NULL;) {
120:       astree asttofree = child;
121:       assert (is_astree (asttofree));
122:       child = child->next;
123:       freeast (asttofree);
124:    }
125:    DEBUGF ('f', "free [%X]-> %d:%d.%d: %s: %p->\"%s\")\n",
126:            (uintptr_t) root, root->filenr, root->linenr, root->offset,
127:             get_yytname (root->symbol), root->lexinfo, root->lexinfo);
128:    free (root->lexinfo);
129:    memset (root, 0, sizeof (struct astree_rep));
130:    free (root);
131: }
132:
133: RCSC("$Id: astree.c,v 1.12 2013-08-22 13:58:43-07 - - $")
134:
```

```
 1:
 2: #include <stdio.h>
 3: #include <assert.h>
 4:
 5: #include "astree.h"
 6: #include "astree.rep.h"
 7: #include "emit.h"
 8: #include "lyutils.h"
 9: #include "auxlib.h"
10:
11: void emit (astree);
12:
13: void emit_insn (char *opcode, char *operand, astree tree) {
14:    printf ("%-10s%-10s%-20s; %s %d.%d\n", "",
15:             opcode, operand, scanner_filename (tree->filenr),
16:             tree->linenr, tree->offset);
17: }
18:
19: void postorder (astree tree) {
20:    astree itor;
21:    assert (tree != NULL);
22:    for (itor = tree->first; itor != NULL; itor = itor->next) {
23:       emit (itor);
24:    }
25: }
26:
27: void postorder_emit_stmts (astree tree) {
28:    postorder (tree);
29: }
30:
31: void postorder_emit_oper (astree tree, char *opcode) {
32:    postorder (tree);
33:    emit_insn (opcode, "", tree);
34: }
35:
36: void postorder_emit_semi (astree tree) {
37:    postorder (tree);
38:    emit_insn ("", "", tree);
39: }
40:
41: void emit_push (astree tree, char *opcode) {
42:    assert (tree != NULL);
43:    assert (tree->first == NULL);
44:    emit_insn (opcode, tree->lexinfo, tree);
45: }
46:
47: void emit_assign (astree tree) {
48:    astree left = NULL;
49:    assert (tree != NULL);
50:    left = tree->first;
51:    assert (left != NULL);
52:    assert (left->next != NULL);
53:    assert (left->next->next == NULL);
54:    emit (left->next);
55:    if (left->symbol != IDENT) {
56:       eprintf ("%:%s: %d: left operand of `=' is not an identifier\n",
57:                scanner_filename (left->filenr), left->linenr);
58:    }else{
59:       emit_insn ("popvar", left->lexinfo, left);
60:    }
61: }
```

```
62:
```

```
63:
64: void emit (astree tree) {
65:    assert (is_astree (tree));
66:    switch (tree->symbol) {
67:       case ROOT  : postorder_emit_stmts (tree);       break;
68:       case ';'   : postorder_emit_semi (tree);        break;
69:       case '='   : emit_assign (tree);                break;
70:       case '+'   : postorder_emit_oper (tree, "add"); break;
71:       case '-'   : postorder_emit_oper (tree, "sub"); break;
72:       case '*'   : postorder_emit_oper (tree, "mul"); break;
73:       case '/'   : postorder_emit_oper (tree, "div"); break;
74:       case '^'   : postorder_emit_oper (tree, "pow"); break;
75:       case POS   : postorder_emit_oper (tree, "pos"); break;
76:       case NEG   : postorder_emit_oper (tree, "neg"); break;
77:       case IDENT : emit_push (tree, "pushvar");       break;
78:       case NUMBER: emit_push (tree, "pushnum");       break;
79:       default    : assert (! "emit default");         break;
80:    }
81: }
82:
83: void emit_sm_code (astree tree) {
84:    printf ("\n");
85:    if (tree) emit (tree);
86: }
87:
88: RCSC("$Id: emit.c,v 1.6 2013-08-22 13:58:43-07 - - $")
89:
```

```
 1:
 2: #include <assert.h>
 3: #include <ctype.h>
 4: #include <stdbool.h>
 5: #include <stdio.h>
 6: #include <stdlib.h>
 7: #include <string.h>
 8:
 9: #include "astree.rep.h"
10: #include "lyutils.h"
11: #include "auxlib.h"
12:
13: astree yyparse_astree = NULL;
14: int scan_linenr = 1;
15: int scan_offset = 0;
16: bool scan_echo = false;
17:
18: struct {
19:    char **filenames;
20:    int size;
21:    int last_filenr;
22: } filename_stack = {NULL, 0, -1};
23:
24: char *scanner_filename (int filenr) {
25:    assert (filename_stack.filenames != NULL);
26:    return filename_stack.filenames[filenr];
27: }
28:
29: void scanner_newfilename (char *filename) {
30:    assert (filename != NULL);
31:    if (filename_stack.filenames == NULL) {
32:       filename_stack.size = 16;
33:       filename_stack.last_filenr = -1;
34:       filename_stack.filenames
35:             = malloc (filename_stack.size * sizeof (char*));
36:       assert (filename_stack.filenames != NULL);
37:    }else if (filename_stack.last_filenr == filename_stack.size - 1) {
38:       filename_stack.size *= 2;
39:       filename_stack.filenames
40:             = realloc (filename_stack.filenames,
41:                        filename_stack.size * sizeof (char*));
42:       assert (filename_stack.filenames != NULL);
43:    }
44:    char *newfilename = strdup (filename);
45:    assert (newfilename != NULL);
46:    filename_stack.filenames[++filename_stack.last_filenr]
47:          = newfilename;
48: }
49:
50: void scanner_newline (void) {
51:    ++scan_linenr;
52:    scan_offset = 0;
53: }
54:
55: void scanner_setecho (bool echoflag) {
56:    scan_echo = echoflag;
57: }
58:
```

```
 59:
 60: void scanner_useraction (void) {
 61:    if (scan_echo) {
 62:       if (scan_offset == 0) printf (";%5d: ", scan_linenr);
 63:       printf ("%s", yytext);
 64:    }
 65:    scan_offset += yyleng;
 66: }
 67:
 68: void yyerror (char *message) {
 69:    assert (filename_stack.filenames != NULL);
 70:    errprintf ("%:%s: %d: %s\n",
 71:               filename_stack.filenames[filename_stack.last_filenr],
 72:               scan_linenr, message);
 73: }
 74:
 75: void scanner_badchar (unsigned char bad) {
 76:    char char_rep[16];
 77:    sprintf (char_rep, isgraph ((int) bad) ? "%c" : "\\%03o", bad);
 78:    errprintf ("%:%s: %d: invalid source character (%s)\n",
 79:               filename_stack.filenames[filename_stack.last_filenr],
 80:               scan_linenr, char_rep);
 81: }
 82:
 83: void scanner_badtoken (char *lexeme) {
 84:    errprintf ("%:%s: %d: invalid token (%s)\n",
 85:               filename_stack.filenames[filename_stack.last_filenr],
 86:               scan_linenr, lexeme);
 87: }
 88:
 89: int yylval_token (int symbol) {
 90:    int offset = scan_offset - yyleng;
 91:    yylval = new_astree (symbol, filename_stack.last_filenr,
 92:                         scan_linenr, offset, yytext);
 93:    return symbol;
 94: }
 95:
 96: astree new_parseroot (void) {
 97:    yyparse_astree = new_astree (ROOT, 0, 0, 0, "<<ROOT>>");
 98:    return yyparse_astree;
 99: }
100:
```

```
101:
102: void scanner_include (void) {
103:    scanner_newline();
104:    char *filename = alloca (strlen (yytext) + 1);
105:    int linenr;
106:    int scan_rc = sscanf (yytext, "# %d \"%[^\"]\"", &linenr, filename);
107:    if (scan_rc != 2) {
108:       errprintf ("%: %d: [%s]: invalid directive, ignored\n",
109:                  scan_rc, yytext);
110:    }else {
111:       char *newfilename = strdup (filename);
112:       assert (newfilename != NULL);
113:       printf (";# %d \"%s\"\n", linenr, newfilename);
114:       scanner_newfilename (newfilename);
115:       scan_linenr = linenr - 1;
116:       DEBUGF ('m', "filename=%s, scan_linenr=%d\n",
117:               filename_stack.filenames[filename_stack.last_filenr],
118:               scan_linenr);
119:    }
120: }
121:
122: RCSC("$Id: lyutils.c,v 1.12 2013-08-22 13:58:43-07 - - $")
123:
```

```
 1:
 2: #include <assert.h>
 3: #include <errno.h>
 4: #include <stdbool.h>
 5: #include <stdio.h>
 6: #include <stdlib.h>
 7: #include <string.h>
 8: #include <unistd.h>
 9:
10: #include "astree.h"
11: #include "emit.h"
12: #include "lyutils.h"
13: #include "auxlib.h"
14:
15: #define CPP "/usr/bin/cpp"
16:
17: struct options{
18:    bool dumptree;
19:    bool echoinput;
20: };
21:
22: // Open a pipe from the C preprocessor.
23: // Exit failure if can't.
24: // Assignes opened pipe to FILE *yyin.
25: char *yyin_cpp_command = NULL;
26: void yyin_cpp_popen (char *filename) {
27:    yyin_cpp_command = malloc (strlen (CPP) + strlen (filename) + 2);
28:    assert (yyin_cpp_command != NULL);
29:    strcpy (yyin_cpp_command, CPP);
30:    strcat (yyin_cpp_command, " ");
31:    strcat (yyin_cpp_command, filename);
32:    yyin = popen (yyin_cpp_command, "r");
33:    if (yyin == NULL) {
34:       syserrprintf (yyin_cpp_command);
35:       exit (get_exitstatus());
36:    }
37: }
38:
39: void yyin_cpp_pclose (void) {
40:    int pclose_rc = pclose (yyin);
41:    eprint_status (yyin_cpp_command, pclose_rc);
42:    if (pclose_rc != 0) set_exitstatus (EXIT_FAILURE);
43: }
44:
```

```
45:
46: void scan_opts (int argc, char **argv, struct options *options) {
47:    int option;
48:    opterr = 0;
49:    yy_flex_debug = 0;
50:    yydebug = 0;
51:    for(;;) {
52:       option = getopt (argc, argv, "@:ely");
53:       if (option == EOF) break;
54:       switch (option) {
55:          case '@': set_debugflags (optarg);   break;
56:          case 'e': options->echoinput = true; break;
57:          case 'l': yy_flex_debug = 1;         break;
58:          case 'y': yydebug = 1;               break;
59:          default:  errprintf ("%:bad option (%c)\n", optopt); break;
60:       }
61:    }
62:    if (optind > argc) {
63:       errprintf ("Usage: %s [-ly] [filename]\n", get_execname());
64:       exit (get_exitstatus());
65:    }
66:    char *filename = optind == argc ? "-" : argv[optind];
67:    yyin_cpp_popen (filename);
68:    DEBUGF ('m', "filename = %s, yyin = %p, fileno (yyin) = %d\n",
69:            filename, yyin, fileno (yyin));
70:    scanner_newfilename (filename);
71: }
72:
73: int main (int argc, char **argv) {
74:    struct options options = {false, false};
75:    int parsecode = 0;
76:    set_execname (argv[0]);
77:    DEBUGSTMT ('m',
78:       for (int argi = 0; argi < argc; ++argi) {
79:          eprintf ("%s%c", argv[argi], argi < argc - 1 ? ' ' : '\n');
80:       }
81:    );
82:    scan_opts (argc, argv, &options);
83:    scanner_setecho (options.echoinput);
84:    parsecode = yyparse();
85:    if (parsecode) {
86:       errprintf ("%:parse failed (%d)\n", parsecode);
87:    }else {
88:       DEBUGSTMT ('a', dump_astree (stderr, yyparse_astree); );
89:       emit_sm_code (yyparse_astree);
90:    }
91:    freeast (yyparse_astree);
92:    yyin_cpp_pclose();
93:    return get_exitstatus();
94: }
95:
96: RCSC("$Id: main.c,v 1.17 2013-08-22 13:58:43-07 - - $")
97:
```

```
 1:
 2: #define _GNU_SOURCE
 3: #define __USE_GNU
 4:
 5: #include <assert.h>
 6: #include <errno.h>
 7: #include <libgen.h>
 8: #include <limits.h>
 9: #include <stdarg.h>
10: #include <stdbool.h>
11: #include <stdio.h>
12: #include <stdlib.h>
13: #include <string.h>
14: #include <wait.h>
15:
16: #include "auxlib.h"
17:
18: static int exitstatus = EXIT_SUCCESS;
19: static char *execname = NULL;
20: static char *debugflags = "";
21: static bool alldebugflags = false;
22:
23: void set_execname (char *argv0) {
24:    execname = basename (argv0);
25: }
26:
27: char *get_execname (void) {
28:    assert (execname != NULL);
29:    return execname;
30: }
31:
32: static void eprint_signal (char *kind, int signal) {
33:    eprintf (", %s %d", kind, signal);
34:    char *sigstr = strsignal (signal);
35:    if (sigstr != NULL) fprintf (stderr, " %s", sigstr);
36: }
37:
38: void eprint_status (char *command, int status) {
39:    if (status == 0) return;
40:    eprintf ("%s: status 0x%04X", command, status);
41:    if (WIFEXITED (status)) {
42:       eprintf (", exit %d", WEXITSTATUS (status));
43:    }
44:    if (WIFSIGNALED (status)) {
45:       eprint_signal ("Terminated", WTERMSIG (status));
46:       #ifdef WCOREDUMP
47:       if (WCOREDUMP (status)) eprintf (", core dumped");
48:       #endif
49:    }
50:    if (WIFSTOPPED (status)) {
51:       eprint_signal ("Stopped", WSTOPSIG (status));
52:    }
53:    if (WIFCONTINUED (status)) {
54:       eprintf (", Continued");
55:    }
56:    eprintf ("\n");
57: }
58:
59: int get_exitstatus (void) {
60:    return exitstatus;
61: }
```

```
62:
```

```
 63:
 64: void veprintf (char *format, va_list args) {
 65:    assert (execname != NULL);
 66:    assert (format != NULL);
 67:    fflush (NULL);
 68:    if (strstr (format, "%:") == format) {
 69:       fprintf (stderr, "%s: ", get_execname ());
 70:       format += 2;
 71:    }
 72:    vfprintf (stderr, format, args);
 73:    fflush (NULL);
 74: }
 75:
 76: void eprintf (char *format, ...) {
 77:    va_list args;
 78:    va_start (args, format);
 79:    veprintf (format, args);
 80:    va_end (args);
 81: }
 82:
 83: void errprintf (char *format, ...) {
 84:    va_list args;
 85:    va_start (args, format);
 86:    veprintf (format, args);
 87:    va_end (args);
 88:    exitstatus = EXIT_FAILURE;
 89: }
 90:
 91: void syserrprintf (char *object) {
 92:    errprintf ("%:%s: %s\n", object, strerror (errno));
 93: }
 94:
 95: void set_exitstatus (int newexitstatus) {
 96:    newexitstatus &= 0xFF;
 97:    if (exitstatus < newexitstatus) exitstatus = newexitstatus;
 98:    DEBUGF ('x', "exitstatus = %d\n", exitstatus);
 99: }
100:
101: void __stubprintf (char *file, int line, const char *func,
102:                    char *format, ...) {
103:    va_list args;
104:    fflush (NULL);
105:    printf ("%s: %s[%d] %s: ", execname, file, line, func);
106:    va_start (args, format);
107:    vprintf (format, args);
108:    va_end (args);
109:    fflush (NULL);
110: }
111:
```

```
112:
113: void set_debugflags (char *flags) {
114:    debugflags = flags;
115:    if (strchr (debugflags, '@') != NULL) alldebugflags = true;
116:    DEBUGF ('x', "Debugflags = \"%s\", all = %d\n",
117:            debugflags, alldebugflags);
118: }
119:
120: bool is_debugflag (char flag) {
121:    return alldebugflags || strchr (debugflags, flag) != NULL;
122: }
123:
124: void __debugprintf (char flag, char *file, int line, const char *func,
125:                     char *format, ...) {
126:    va_list args;
127:    if (! is_debugflag (flag)) return;
128:    fflush (NULL);
129:    va_start (args, format);
130:    fprintf (stderr, "DEBUGF(%c): %s[%d] %s():\n",
131:             flag, file, line, func);
132:    vfprintf (stderr, format, args);
133:    va_end (args);
134:    fflush (NULL);
135: }
136:
137: RCSC("$Id: auxlib.c,v 1.16 2013-08-22 13:59:59-07 - - $")
138:
```

```
    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, 20
06
    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 debuggers
   40:       know about them.  */
   41:    enum yytokentype {
   42:      ROOT = 258,
   43:      IDENT = 259,
   44:      NUMBER = 260,
   45:      NEG = 263,
   46:      POS = 264
   47:    };
   48: #endif
   49:
   50:
   51:
   52: #if ! defined YYSTYPE && ! defined YYSTYPE_IS_DECLARED
   53: typedef int YYSTYPE;
   54: # define YYSTYPE_IS_TRIVIAL 1
   55: # define yystype YYSTYPE /* obsolescent; will be withdrawn */
   56: # define YYSTYPE_IS_DECLARED 1
   57: #endif
   58:
   59: extern YYSTYPE yylval;
   60:
```

```
61:
```