```
 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 1.1 2013-09-19 16:38:25-07 - - $
12:
```

```
 1: # $Id: Makefile,v 1.11 2013-10-15 16:37:56-07 - - $
 2:
 3: MKFILE    = Makefile
 4: DEPSFILE  = ${MKFILE}.deps
 5: NOINCLUDE = ci clean spotless
 6: NEEDINCL  = ${filter ${NOINCLUDE}, ${MAKECMDGOALS}}
 7: VALGRIND  = valgrind --leak-check=full --show-reachable=yes
 8:
 9: #
10: # Definitions of list of files:
11: #
12: HSOURCES  = astree.h  emit.h  lyutils.h  auxlib.h  stringset.h
13: CSOURCES  = astree.cc emit.cc lyutils.cc auxlib.cc stringset.cc main.cc
14: LSOURCES  = scanner.l
15: YSOURCES  = parser.y
16: ETCSRC    = README ${MKFILE} ${DEPSFILE}
17: CLGEN     = yylex.cc
18: HYGEN     = yyparse.h
19: CYGEN     = yyparse.cc
20: CGENS     = ${CLGEN} ${CYGEN}
21: ALLGENS   = ${HYGEN} ${CGENS}
22: EXECBIN   = zexprsm
23: ALLCSRC   = ${CSOURCES} ${CGENS}
24: OBJECTS   = ${ALLCSRC:.cc=.o}
25: LREPORT   = yylex.output
26: YREPORT   = yyparse.output
27: IREPORT   = ident.output
28: REPORTS   = ${LREPORT} ${YREPORT} ${IREPORT}
29: ALLSRC    = ${ETCSRC} ${YSOURCES} ${LSOURCES} ${HSOURCES} ${CSOURCES}
30: TESTINS   = ${wildcard test?.in}
31: LISTSRC   = ${ALLSRC} ${HYGEN}
32:
33: #
34: # Definitions of the compiler and compilation options:
35: #
36: GCC       = g++ -g -O0 -Wall -Wextra -std=gnu++0x
37: MKDEPS    = g++ -MM -std=gnu++0x
38:
39: #
40: # The first target is always ``all'', and hence the default,
41: # and builds the executable images
42: #
43: all : ${EXECBIN}
44:
45: #
46: # Build the executable image from the object files.
47: #
48: ${EXECBIN} : ${OBJECTS}
49:         ${GCC} -o${EXECBIN} ${OBJECTS}
50:         ident ${OBJECTS} ${EXECBIN} >${IREPORT}
51:
52: #
53: # Build an object file form a C source file.
54: #
55: %.o : %.cc
56:         ${GCC} -c $<
57:
```

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

```
 98:
 99: #
100: # Build the dependencies file using the C preprocessor
101: #
102: deps : ${ALLCSRC}
103:         @ echo "# ${DEPSFILE} created `date` by ${MAKE}" >${DEPSFILE}
104:         ${MKDEPS} ${ALLCSRC} >>${DEPSFILE}
105:
106: ${DEPSFILE} :
107:         @ touch ${DEPSFILE}
108:         ${MAKE} --no-print-directory deps
109:
110: #
111: # Test
112: #
113:
114: tests : ${EXECBIN}
115:         touch ${TESTINS}
116:         make --no-print-directory ${TESTINS:.in=.out}
117:
118: %.out %.err : %.in ${EXECBIN}
119:         ( ${VALGRIND} ${EXECBIN} -ly -@@ $< \
120:         ;  echo EXIT STATUS $$? 1>&2 \
121:         ) 1>$*.out 2>$*.err
122:
123: #
124: # Everything
125: #
126: again :
127:         gmake --no-print-directory spotless deps ci all lis
128:
129: ifeq "${NEEDINCL}" ""
130: include ${DEPSFILE}
131: endif
132:
```

```
1: # Makefile.deps created Tue Oct 15 16:37:56 PDT 2013 by gmake
2: astree.o: astree.cc astree.h auxlib.h stringset.h lyutils.h yyparse.h
3: emit.o: emit.cc astree.h auxlib.h emit.h lyutils.h yyparse.h
4: lyutils.o: lyutils.cc lyutils.h astree.h auxlib.h yyparse.h
5: auxlib.o: auxlib.cc auxlib.h
6: stringset.o: stringset.cc stringset.h auxlib.h
7: main.o: main.cc astree.h auxlib.h emit.h lyutils.h yyparse.h stringset.h
8: yylex.o: yylex.cc auxlib.h lyutils.h astree.h yyparse.h
9: yyparse.o: yyparse.cc lyutils.h astree.h auxlib.h yyparse.h
```

```
 1: %{
 2: // $Id: parser.y,v 1.5 2013-10-10 18:48:18-07 - - $
 3:
 4: #include <assert.h>
 5: #include <stdlib.h>
 6: #include <string.h>
 7:
 8: #include "lyutils.h"
 9: #include "astree.h"
10:
11: #define YYDEBUG 1
12: #define YYERROR_VERBOSE 1
13: #define YYPRINT yyprint
14: #define YYMALLOC yycalloc
15:
16: static void* yycalloc (size_t size);
17:
18: %}
19:
20: %debug
21: %defines
22: %error-verbose
23: %token-table
24: %verbose
25:
26: %destructor { error_destructor ($$); } <>
27:
28: %token  ROOT IDENT NUMBER
29:
30: %right  '='
31: %left   '+' '-'
32: %left   '*' '/'
33: %right  '^'
34: %right  POS "u+" NEG "u-"
35:
36: %start  program
37:
```

```
38:
39: %%
40:
41: program : stmtseq               { $$ = $1; }
42:         ;
43:
44: stmtseq : stmtseq expr ';'      { free_ast ($3); $$ = adopt1 ($1, $2); }
45:         | stmtseq error ';'     { free_ast ($3); $$ = $1; }
46:         | stmtseq ';'           { free_ast ($2); $$ = $1; }
47:         |                       { $$ = new_parseroot(); }
48:         ;
49:
50: expr    : expr '=' expr         { $$ = adopt2 ($2, $1, $3); }
51:         | expr '+' expr         { $$ = adopt2 ($2, $1, $3); }
52:         | expr '-' expr         { $$ = adopt2 ($2, $1, $3); }
53:         | expr '*' expr         { $$ = adopt2 ($2, $1, $3); }
54:         | expr '/' expr         { $$ = adopt2 ($2, $1, $3); }
55:         | expr '^' expr         { $$ = adopt2 ($2, $1, $3); }
56:         | '+' expr %prec POS    { $$ = adopt1sym ($1, $2, POS); }
57:         | '-' expr %prec NEG    { $$ = adopt1sym ($1, $2, NEG); }
58:         | '(' expr ')'          { free_ast2 ($1, $3); $$ = $2; }
59:         | IDENT                 { $$ = $1; }
60:         | NUMBER                { $$ = $1; }
61:         ;
62:
63: %%
64:
65: const char* get_yytname (int symbol) {
66:    return yytname [YYTRANSLATE (symbol)];
67: }
68:
69: bool is_defined_token (int symbol) {
70:    return YYTRANSLATE (symbol) > YYUNDEFTOK;
71: }
72:
73: static void* yycalloc (size_t size) {
74:    void* result = calloc (1, size);
75:    assert (result != NULL);
76:    return result;
77: }
78:
79: RCSC("$Id: parser.y,v 1.5 2013-10-10 18:48:18-07 - - $")
80:
```

```
%{
// $Id: scanner.l,v 1.3 2013-10-15 16:37:56-07 - - $

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

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

%}

%option 8bit
%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); }

%%

RCSC("$Id: scanner.l,v 1.3 2013-10-15 16:37:56-07 - - $")

```

```
 1: #ifndef __ASTREE_H__
 2: #define __ASTREE_H__
 3:
 4: #include <string>
 5: #include <vector>
 6: using namespace std;
 7:
 8: #include "auxlib.h"
 9:
10: struct astree {
11:    int symbol;               // token code
12:    size_t filenr;            // index into filename stack
13:    size_t linenr;            // line number from source code
14:    size_t offset;            // offset of token with current line
15:    const string* lexinfo;    // pointer to lexical information
16:    vector<astree*> children; // children of this n-way node
17: };
18:
19:
20: astree* new_astree (int symbol, int filenr, int linenr, int offset,
21:                     const char* lexinfo);
22: astree* adopt1 (astree* root, astree* child);
23: astree* adopt2 (astree* root, astree* left, astree* right);
24: astree* adopt1sym (astree* root, astree* child, int symbol);
25: void dump_astree (FILE* outfile, astree* root);
26: void yyprint (FILE* outfile, unsigned short toknum, astree* yyvaluep);
27: void free_ast (astree* tree);
28: void free_ast2 (astree* tree1, astree* tree2);
29:
30: RCSH("$Id: astree.h,v 1.3 2013-09-20 12:23:31-07 - - $")
31: #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 1.1 2013-09-19 16:38:25-07 - - $")
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 (const char* message);
24: int yylex_destroy (void);
25: const char* get_yytname (int symbol);
26: bool is_defined_token (int symbol);
27:
28: const string* scanner_filename (int filenr);
29: void scanner_newfilename (const char* filename);
30: void scanner_badchar (unsigned char bad);
31: void scanner_badtoken (char* lexeme);
32: void scanner_newline (void);
33: void scanner_setecho (bool echoflag);
34: void scanner_useraction (void);
35:
36: astree* new_parseroot (void);
37: int yylval_token (int symbol);
38: void error_destructor (astree*);
39:
40: void scanner_include (void);
41:
42: typedef astree* astree_pointer;
43: #define YYSTYPE astree_pointer
44: #include "yyparse.h"
45:
46: RCSH("$Id: lyutils.h,v 1.5 2013-10-10 18:17:45-07 - - $")
47: #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: const char* get_execname (void);
23:    //
24:    // Returns a read-only value previously stored by set_progname.
25:    //
26:
27: void eprint_status (const 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 (const 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 (const 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 (const 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 (const 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:    // Sets the exit status to EXIT_FAILURE.
71:    //
72:
```

```
 73:
 74: //
 75: // Support for stub messages.
 76: //
 77: #define STUBPRINTF(...) \
 78:         __stubprintf (__FILE__, __LINE__, __func__, __VA_ARGS__)
 79: void __stubprintf (const char* file, int line, const char* func,
 80:                    const char* format, ...);
 81:
 82: //
 83: // Debugging utility.
 84: //
 85:
 86: void set_debugflags (const char* flags);
 87:    //
 88:    // Sets a string of debug flags to be used by DEBUGF statements.
 89:    // Uses the address of the string, and does not copy it, so it
 90:    // must not be dangling.  If a particular debug flag has been set,
 91:    // messages are printed.  The format is identical to printf format.
 92:    // The flag "@" turns on all flags.
 93:    //
 94:
 95: bool is_debugflag (char flag);
 96:    //
 97:    // Checks to see if a debugflag is set.
 98:    //
 99:
100: #ifdef NDEBUG
101: // Do not generate any code.
102: #define DEBUGF(FLAG,...)   /**/
103: #define DEBUGSTMT(FLAG,STMTS) /**/
104: #else
105: // Generate debugging code.
106: void __debugprintf (char flag, const char* file, int line,
107:                     const char* func, const char* format, ...);
108: #define DEBUGF(FLAG,...) \
109:         __debugprintf (FLAG, __FILE__, __LINE__, __func__, __VA_ARGS__)
110: #define DEBUGSTMT(FLAG,STMTS) \
111:         if (is_debugflag (FLAG)) { DEBUGF (FLAG, "\n"); STMTS }
112: #endif
113:
114: //
115: // Definition of RCSID macro to include RCS info in objs and execbin.
116: //
117:
118: #define RCS3(ID,N,X) static const char ID##N[] = X;
119: #define RCS2(N,X) RCS3(RCS_Id,N,X)
120: #define RCSH(X) RCS2(__COUNTER__,X)
121: #define RCSC(X) RCSH(X \
122: "\0$Compiled: " __FILE__ " " __DATE__ " " __TIME__ " $")
123: RCSH("$Id: auxlib.h,v 1.2 2013-09-19 19:55:32-07 - - $")
124: #endif
```

```
 1: #ifndef __STRINGSET__
 2: #define __STRINGSET__
 3:
 4: #include <string>
 5: #include <unordered_set>
 6: using namespace std;
 7:
 8: #include <stdio.h>
 9:
10: #include "auxlib.h"
11:
12: const string* intern_stringset (const char*);
13:
14: void dump_stringset (FILE*);
15:
16: RCSH("$Id: stringset.h,v 1.5 2013-09-23 14:16:09-07 - - $")
17: #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 "stringset.h"
11: #include "lyutils.h"
12:
13: astree* new_astree (int symbol, int filenr, int linenr, int offset,
14:                     const char* lexinfo) {
15:    astree* tree = new astree();
16:    tree->symbol = symbol;
17:    tree->filenr = filenr;
18:    tree->linenr = linenr;
19:    tree->offset = offset;
20:    tree->lexinfo = intern_stringset (lexinfo);
21:    DEBUGF ('f', "astree %p->{%d:%d.%d: %s: \"%s\"}\n",
22:            tree, tree->filenr, tree->linenr, tree->offset,
23:            get_yytname (tree->symbol), tree->lexinfo->c_str());
24:    return tree;
25: }
26:
```

```
27:
28: astree* adopt1 (astree* root, astree* child) {
29:    root->children.push_back (child);
30:    DEBUGF ('a', "%p (%s) adopting %p (%s)\n",
31:            root, root->lexinfo->c_str(),
32:            child, child->lexinfo->c_str());
33:    return root;
34: }
35:
36: astree* adopt2 (astree* root, astree* left, astree* right) {
37:    adopt1 (root, left);
38:    adopt1 (root, right);
39:    return root;
40: }
41:
42: astree* adopt1sym (astree* root, astree* child, int symbol) {
43:    root = adopt1 (root, child);
44:    root->symbol = symbol;
45:    return root;
46: }
47:
```

```
48:
49: static void dump_node (FILE* outfile, astree* node) {
50:    fprintf (outfile, "%p->{%s(%d) %ld:%ld.%03ld \"%s\" [",
51:             node, get_yytname (node->symbol), node->symbol,
52:             node->filenr, node->linenr, node->offset,
53:             node->lexinfo->c_str());
54:    bool need_space = false;
55:    for (size_t child = 0; child < node->children.size(); ++child) {
56:       if (need_space) fprintf (outfile, " ");
57:       need_space = true;
58:       fprintf (outfile, "%p", node->children.at(child));
59:    }
60:    fprintf (outfile, "]}");
61: }
62:
63: static void dump_astree_rec (FILE* outfile, astree* root, int depth) {
64:    if (root == NULL) return;
65:    fprintf (outfile, "%*s%s ", depth * 3, "", root->lexinfo->c_str());
66:    dump_node (outfile, root);
67:    fprintf (outfile, "\n");
68:    for (size_t child = 0; child < root->children.size(); ++child) {
69:       dump_astree_rec (outfile, root->children[child], depth + 1);
70:    }
71: }
72:
73: void dump_astree (FILE* outfile, astree* root) {
74:    dump_astree_rec (outfile, root, 0);
75:    fflush (NULL);
76: }
77:
78: void yyprint (FILE* outfile, unsigned short toknum, astree* yyvaluep) {
79:    DEBUGF ('f', "toknum = %d, yyvaluep = %p\n", toknum, yyvaluep);
80:    if (is_defined_token (toknum)) {
81:       dump_node (outfile, yyvaluep);
82:    }else {
83:       fprintf (outfile, "%s(%d)\n", get_yytname (toknum), toknum);
84:    }
85:    fflush (NULL);
86: }
87:
```

```
 88:
 89: void free_ast (astree* root) {
 90:    while (not root->children.empty()) {
 91:       astree* child = root->children.back();
 92:       root->children.pop_back();
 93:       free_ast (child);
 94:    }
 95:    DEBUGF ('f', "free [%X]-> %d:%d.%d: %s: \"%s\")\n",
 96:            (uintptr_t) root, root->filenr, root->linenr, root->offset,
 97:            get_yytname (root->symbol), root->lexinfo->c_str());
 98:    delete root;
 99: }
100:
101: void free_ast2 (astree* tree1, astree* tree2) {
102:    free_ast (tree1);
103:    free_ast (tree2);
104: }
105:
106: RCSC("$Id: astree.cc,v 1.14 2013-10-10 18:48:18-07 - - $")
107:
```

```
 1:
 2: #include <stdio.h>
 3: #include <assert.h>
 4:
 5: #include "astree.h"
 6: #include "emit.h"
 7: #include "lyutils.h"
 8: #include "auxlib.h"
 9:
10: void emit (astree*);
11:
12: void emit_insn (const char* opcode, const char* operand, astree* tree) {
13:    printf ("%-10s%-10s%-20s; %s %ld.%ld\n", "",
14:             opcode, operand, scanner_filename (tree->filenr)->c_str(),
15:             tree->linenr, tree->offset);
16: }
17:
18: void postorder (astree* tree) {
19:    assert (tree != NULL);
20:    for (size_t child = 0; child < tree->children.size(); ++child) {
21:       emit (tree->children.at(child));
22:    }
23: }
24:
25: void postorder_emit_stmts (astree* tree) {
26:    postorder (tree);
27: }
28:
29: void postorder_emit_oper (astree* tree, const char* opcode) {
30:    postorder (tree);
31:    emit_insn (opcode, "", tree);
32: }
33:
34: void postorder_emit_semi (astree* tree) {
35:    postorder (tree);
36:    emit_insn ("", "", tree);
37: }
38:
39: void emit_push (astree* tree, const char* opcode) {
40:    emit_insn (opcode, tree->lexinfo->c_str(), tree);
41: }
42:
43: void emit_assign (astree* tree) {
44:    assert (tree->children.size() == 2);
45:    astree* left = tree->children.at(0);
46:    emit (tree->children.at(1));
47:    if (left->symbol != IDENT) {
48:       eprintf ("%:%s: %d: left operand of `=' is not an identifier\n",
49:                scanner_filename (left->filenr)->c_str(), left->linenr);
50:    }else{
51:       emit_insn ("popvar", left->lexinfo->c_str(), left);
52:    }
53: }
54:
```

```
55:
56: void emit (astree* tree) {
57:    switch (tree->symbol) {
58:       case ROOT  : postorder_emit_stmts (tree);       break;
59:       case ';'   : postorder_emit_semi (tree);        break;
60:       case '='   : emit_assign (tree);                break;
61:       case '+'   : postorder_emit_oper (tree, "add"); break;
62:       case '-'   : postorder_emit_oper (tree, "sub"); break;
63:       case '*'   : postorder_emit_oper (tree, "mul"); break;
64:       case '/'   : postorder_emit_oper (tree, "div"); break;
65:       case '^'   : postorder_emit_oper (tree, "pow"); break;
66:       case POS   : postorder_emit_oper (tree, "pos"); break;
67:       case NEG   : postorder_emit_oper (tree, "neg"); break;
68:       case IDENT : emit_push (tree, "pushvar");       break;
69:       case NUMBER: emit_push (tree, "pushnum");       break;
70:       default    : assert (false);                    break;
71:    }
72: }
73:
74: void emit_sm_code (astree* tree) {
75:    printf ("\n");
76:    if (tree) emit (tree);
77: }
78:
79: RCSC("$Id: emit.cc,v 1.3 2013-09-20 17:52:13-07 - - $")
80:
```

```
 1:
 2: #include <vector>
 3: #include <string>
 4: using namespace std;
 5:
 6: #include <assert.h>
 7: #include <ctype.h>
 8: #include <stdio.h>
 9: #include <stdlib.h>
10: #include <string.h>
11:
12: #include "lyutils.h"
13: #include "auxlib.h"
14:
15: astree* yyparse_astree = NULL;
16: int scan_linenr = 1;
17: int scan_offset = 0;
18: bool scan_echo = false;
19: vector<string> included_filenames;
20:
21: const string* scanner_filename (int filenr) {
22:    return &included_filenames.at(filenr);
23: }
24:
25: void scanner_newfilename (const char* filename) {
26:    included_filenames.push_back (filename);
27: }
28:
29: void scanner_newline (void) {
30:    ++scan_linenr;
31:    scan_offset = 0;
32: }
33:
34: void scanner_setecho (bool echoflag) {
35:    scan_echo = echoflag;
36: }
37:
```

```
38:
39: void scanner_useraction (void) {
40:    if (scan_echo) {
41:       if (scan_offset == 0) printf (";%5d: ", scan_linenr);
42:       printf ("%s", yytext);
43:    }
44:    scan_offset += yyleng;
45: }
46:
47: void yyerror (const char* message) {
48:    assert (not included_filenames.empty());
49:    errprintf ("%:%s: %d: %s\n",
50:               included_filenames.back().c_str(), scan_linenr, message);
51: }
52:
53: void scanner_badchar (unsigned char bad) {
54:    char char_rep[16];
55:    sprintf (char_rep, isgraph ((int) bad) ? "%c" : "\\%03o", bad);
56:    errprintf ("%:%s: %d: invalid source character (%s)\n",
57:               included_filenames.back().c_str(), scan_linenr, char_rep);
58: }
59:
60: void scanner_badtoken (char* lexeme) {
61:    errprintf ("%:%s: %d: invalid token (%s)\n",
62:               included_filenames.back().c_str(), scan_linenr, lexeme);
63: }
64:
65: int yylval_token (int symbol) {
66:    int offset = scan_offset - yyleng;
67:    yylval = new_astree (symbol, included_filenames.size() - 1,
68:                         scan_linenr, offset, yytext);
69:    return symbol;
70: }
71:
72: void error_destructor (astree* tree) {
73:    if (tree == yyparse_astree) return;
74:    DEBUGSTMT ('a', dump_astree (stderr, tree); );
75:    free_ast (tree);
76: }
77:
78: astree* new_parseroot (void) {
79:    yyparse_astree = new_astree (ROOT, 0, 0, 0, "<<ROOT>>");
80:    return yyparse_astree;
81: }
82:
```

```
 83:
 84: void scanner_include (void) {
 85:    scanner_newline();
 86:    char filename[strlen (yytext) + 1];
 87:    int linenr;
 88:    int scan_rc = sscanf (yytext, "# %d \"%[^\"]\"", &linenr, filename);
 89:    if (scan_rc != 2) {
 90:       errprintf ("%: %d: [%s]: invalid directive, ignored\n",
 91:                  scan_rc, yytext);
 92:    }else {
 93:       printf (";# %d \"%s\"\n", linenr, filename);
 94:       scanner_newfilename (filename);
 95:       scan_linenr = linenr - 1;
 96:       DEBUGF ('m', "filename=%s, scan_linenr=%d\n",
 97:               included_filenames.back().c_str(), scan_linenr);
 98:    }
 99: }
100:
101: RCSC("$Id: lyutils.cc,v 1.3 2013-10-10 18:17:45-07 - - $")
102:
```

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

```
 59:
 60: void veprintf (const char* format, va_list args) {
 61:    assert (execname != NULL);
 62:    assert (format != NULL);
 63:    fflush (NULL);
 64:    if (strstr (format, "%:") == format) {
 65:       fprintf (stderr, "%s: ", get_execname ());
 66:       format += 2;
 67:    }
 68:    vfprintf (stderr, format, args);
 69:    fflush (NULL);
 70: }
 71:
 72: void eprintf (const char* format, ...) {
 73:    va_list args;
 74:    va_start (args, format);
 75:    veprintf (format, args);
 76:    va_end (args);
 77: }
 78:
 79: void errprintf (const char* format, ...) {
 80:    va_list args;
 81:    va_start (args, format);
 82:    veprintf (format, args);
 83:    va_end (args);
 84:    exitstatus = EXIT_FAILURE;
 85: }
 86:
 87: void syserrprintf (const char* object) {
 88:    errprintf ("%:%s: %s\n", object, strerror (errno));
 89: }
 90:
 91: void set_exitstatus (int newexitstatus) {
 92:    if (exitstatus < newexitstatus) exitstatus = newexitstatus;
 93:    DEBUGF ('x', "exitstatus = %d\n", exitstatus);
 94: }
 95:
 96: void __stubprintf (const char* file, int line, const char* func,
 97:                    const char* format, ...) {
 98:    va_list args;
 99:    fflush (NULL);
100:    printf ("%s: %s[%d] %s: ", execname, file, line, func);
101:    va_start (args, format);
102:    vprintf (format, args);
103:    va_end (args);
104:    fflush (NULL);
105: }
106:
```

```
107:
108: void set_debugflags (const char* flags) {
109:    debugflags = flags;
110:    if (strchr (debugflags, '@') != NULL) alldebugflags = true;
111:    DEBUGF ('x', "Debugflags = \"%s\", all = %d\n",
112:            debugflags, alldebugflags);
113: }
114:
115: bool is_debugflag (char flag) {
116:    return alldebugflags or strchr (debugflags, flag) != NULL;
117: }
118:
119: void __debugprintf (char flag, const char* file, int line,
120:                     const char* func, const char* format, ...) {
121:    va_list args;
122:    if (not is_debugflag (flag)) return;
123:    fflush (NULL);
124:    va_start (args, format);
125:    fprintf (stderr, "DEBUGF(%c): %s[%d] %s():\n",
126:             flag, file, line, func);
127:    vfprintf (stderr, format, args);
128:    va_end (args);
129:    fflush (NULL);
130: }
131:
132: RCSC("$Id: auxlib.cc,v 1.3 2013-09-20 17:52:13-07 - - $")
133:
```

```
 1:
 2: #include <string>
 3: #include <unordered_set>
 4: using namespace std;
 5:
 6: #include "stringset.h"
 7:
 8: typedef unordered_set<string> stringset;
 9: typedef stringset::const_iterator stringset_citor;
10: typedef stringset::const_local_iterator stringset_bucket_citor;
11:
12: stringset set;
13:
14: const string* intern_stringset (const char* string) {
15:    pair<stringset_citor,bool> handle = set.insert (string);
16:    return &*handle.first;
17: }
18:
19: void dump_stringset (FILE* out) {
20:    size_t max_bucket_size = 0;
21:    for (size_t bucket = 0; bucket < set.bucket_count(); ++bucket) {
22:       bool need_index = true;
23:       size_t curr_size = set.bucket_size (bucket);
24:       if (max_bucket_size < curr_size) max_bucket_size = curr_size;
25:       for (stringset_bucket_citor itor = set.cbegin (bucket);
26:            itor != set.cend (bucket); ++itor) {
27:          if (need_index) fprintf (out, "stringset[%4lu]: ", bucket);
28:                     else fprintf (out, "          %4s   ", "");
29:          need_index = false;
30:          const string* str = &*itor;
31:          fprintf (out, "%22lu %p->\"%s\"\n", set.hash_function()(*str),
32:                   str, str->c_str());
33:       }
34:    }
35:    fprintf (out, "load_factor = %.3f\n", set.load_factor());
36:    fprintf (out, "bucket_count = %lu\n", set.bucket_count());
37:    fprintf (out, "max_bucket_size = %lu\n", max_bucket_size);
38: }
39:
40: RCSC("$Id: stringset.cc,v 1.6 2013-10-10 17:44:18-07 - - $")
```

```
 1:
 2: #include <string>
 3: #include <vector>
 4: using namespace std;
 5:
 6: #include <assert.h>
 7: #include <errno.h>
 8: #include <stdio.h>
 9: #include <stdlib.h>
10: #include <string.h>
11: #include <unistd.h>
12:
13: #include "astree.h"
14: #include "auxlib.h"
15: #include "emit.h"
16: #include "lyutils.h"
17: #include "stringset.h"
18:
19: const string cpp_name = "/usr/bin/cpp";
20: string yyin_cpp_command;
21:
22: // Open a pipe from the C preprocessor.
23: // Exit failure if can't.
24: // Assignes opened pipe to FILE* yyin.
25: void yyin_cpp_popen (const char* filename) {
26:    yyin_cpp_command = cpp_name;
27:    yyin_cpp_command += " ";
28:    yyin_cpp_command += filename;
29:    yyin = popen (yyin_cpp_command.c_str(), "r");
30:    if (yyin == NULL) {
31:       syserrprintf (yyin_cpp_command.c_str());
32:       exit (get_exitstatus());
33:    }
34: }
35:
36: void yyin_cpp_pclose (void) {
37:    int pclose_rc = pclose (yyin);
38:    eprint_status (yyin_cpp_command.c_str(), pclose_rc);
39:    if (pclose_rc != 0) set_exitstatus (EXIT_FAILURE);
40: }
41:
42: bool want_echo () {
43:    return not (isatty (fileno (stdin)) and isatty (fileno (stdout)));
44: }
45:
```

```
46:
47: void scan_opts (int argc, char** argv) {
48:    int option;
49:    opterr = 0;
50:    yy_flex_debug = 0;
51:    yydebug = 0;
52:    for(;;) {
53:       option = getopt (argc, argv, "@:ely");
54:       if (option == EOF) break;
55:       switch (option) {
56:          case '@': set_debugflags (optarg);   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:    const 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:    int parsecode = 0;
75:    set_execname (argv[0]);
76:    DEBUGSTMT ('m',
77:       for (int argi = 0; argi < argc; ++argi) {
78:          eprintf ("%s%c", argv[argi], argi < argc - 1 ? ' ' : '\n');
79:       }
80:    );
81:    scan_opts (argc, argv);
82:    scanner_setecho (want_echo());
83:    parsecode = yyparse();
84:    if (parsecode) {
85:       errprintf ("%:parse failed (%d)\n", parsecode);
86:    }else {
87:       DEBUGSTMT ('a', dump_astree (stderr, yyparse_astree); );
88:       emit_sm_code (yyparse_astree);
89:    }
90:    free_ast (yyparse_astree);
91:    yyin_cpp_pclose();
92:    DEBUGSTMT ('s', dump_stringset (stderr); );
93:    yylex_destroy();
94:    return get_exitstatus();
95: }
96:
97: RCSC("$Id: main.cc,v 1.4 2013-09-20 17:52:13-07 - - $")
98:
```

```
    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:      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:
```