```
 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 2015-07-08 13:29:32-07 - - $
12:
```

```
/* $Id: lexer.l,v 1.8 2019-04-10 14:51:55-07 - - $ */

%{

#include "lyutils.h"

#define YY_USER_ACTION  { lexer::advance(); }

int yylval_token (int symbol) {
   yylval = new astree (symbol, lexer::lloc, yytext);
   return symbol;
}

%}

%option 8bit
%option debug
%option nobackup
%option nodefault
%option noinput
%option nounput
%option noyywrap
%option warn
/*%option verbose*/

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

%%

"#".*           { lexer::include(); }
[ \t]+          { }
\n              { lexer::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}     { lexer::badtoken (yytext);
                  return yylval_token (NUMBER); }
.               { lexer::badchar (*yytext); }

%%
```

```
   1: // $Id: parser.y,v 1.14 2016-10-06 16:26:41-07 - - $
   2:
   3: %{
   4:
   5: #include <assert.h>
   6: #include <stdlib.h>
   7: #include <string.h>
   8:
   9: #include "astree.h"
  10: #include "lyutils.h"
  11:
  12: %}
  13:
  14: %debug
  15: %defines
  16: %error-verbose
  17: %token-table
  18: %verbose
  19:
  20: %destructor { destroy ($$); } <>
  21: %printer { astree::dump (yyoutput, $$); } <>
  22:
  23: %initial-action {
  24:    parser::root = new astree (ROOT, {0, 0, 0}, "<<ROOT>>");
  25: }
  26:
  27: %token  ROOT IDENT NUMBER
  28:
  29: %right  '='
  30: %left   '+' '-'
  31: %left   '*' '/'
  32: %right  '^'
  33: %right  POS NEG
  34:
  35: %start  program
  36:
```

```
37:
38: %%
39:
40: program : stmtseq               { $$ = $1 = nullptr; }
41:         ;
42:
43: stmtseq : stmtseq expr ';'      { destroy ($3); $$ = $1->adopt ($2); }
44:         | stmtseq error ';'     { destroy ($3); $$ = $1; }
45:         | stmtseq ';'           { destroy ($2); $$ = $1; }
46:         |                       { $$ = parser::root; }
47:         ;
48:
49: expr    : expr '=' expr         { $$ = $2->adopt ($1, $3); }
50:         | expr '+' expr         { $$ = $2->adopt ($1, $3); }
51:         | expr '-' expr         { $$ = $2->adopt ($1, $3); }
52:         | expr '*' expr         { $$ = $2->adopt ($1, $3); }
53:         | expr '/' expr         { $$ = $2->adopt ($1, $3); }
54:         | expr '^' expr         { $$ = $2->adopt ($1, $3); }
55:         | '+' expr %prec POS    { $$ = $1->adopt_sym ($2, POS); }
56:         | '-' expr %prec NEG    { $$ = $1->adopt_sym ($2, NEG); }
57:         | '(' expr ')'          { destroy ($1, $3); $$ = $2; }
58:         | IDENT                 { $$ = $1; }
59:         | NUMBER                { $$ = $1; }
60:         ;
61:
62: %%
63:
64: const char* parser::get_tname (int symbol) {
65:    return yytname [YYTRANSLATE (symbol)];
66: }
67:
```

```
 1: // $Id: astree.h,v 1.10 2016-10-06 16:42:35-07 - - $
 2:
 3: #ifndef __ASTREE_H__
 4: #define __ASTREE_H__
 5:
 6: #include <string>
 7: #include <vector>
 8: using namespace std;
 9:
10: #include "auxlib.h"
11:
12: struct location {
13:    size_t filenr;
14:    size_t linenr;
15:    size_t offset;
16: };
17:
18: struct astree {
19:
20:    // Fields.
21:    int symbol;               // token code
22:    location lloc;            // source location
23:    const string* lexinfo;    // pointer to lexical information
24:    vector<astree*> children; // children of this n-way node
25:
26:    // Functions.
27:    astree (int symbol, const location&, const char* lexinfo);
28:    ~astree();
29:    astree* adopt (astree* child1, astree* child2 = nullptr);
30:    astree* adopt_sym (astree* child, int symbol);
31:    void dump_node (FILE*);
32:    void dump_tree (FILE*, int depth = 0);
33:    static void dump (FILE* outfile, astree* tree);
34:    static void print (FILE* outfile, astree* tree, int depth = 0);
35: };
36:
37: void destroy (astree* tree1, astree* tree2 = nullptr);
38:
39: void errllocprintf (const location&, const char* format, const char*);
40:
41: #endif
42:
```

```
 1: // $Id: astree.cpp,v 1.17 2019-03-15 14:32:40-07 - - $
 2:
 3: #include <assert.h>
 4: #include <inttypes.h>
 5: #include <stdarg.h>
 6: #include <stdio.h>
 7: #include <stdlib.h>
 8: #include <string.h>
 9:
10: #include "astree.h"
11: #include "string_set.h"
12: #include "lyutils.h"
13:
14: astree::astree (int symbol_, const location& lloc_, const char* info) {
15:    symbol = symbol_;
16:    lloc = lloc_;
17:    lexinfo = string_set::intern (info);
18:    // vector defaults to empty -- no children
19: }
20:
21: astree::~astree() {
22:    while (not children.empty()) {
23:       astree* child = children.back();
24:       children.pop_back();
25:       delete child;
26:    }
27:    if (yydebug) {
28:       fprintf (stderr, "Deleting astree (");
29:       astree::dump (stderr, this);
30:       fprintf (stderr, ")\n");
31:    }
32: }
33:
34: astree* astree::adopt (astree* child1, astree* child2) {
35:    if (child1 != nullptr) children.push_back (child1);
36:    if (child2 != nullptr) children.push_back (child2);
37:    return this;
38: }
39:
40: astree* astree::adopt_sym (astree* child, int symbol_) {
41:    symbol = symbol_;
42:    return adopt (child);
43: }
44:
```

```
45:
46: void astree::dump_node (FILE* outfile) {
47:    fprintf (outfile, "%p->{%s %zd.%zd.%zd \"%s\":",
48:             static_cast<const void*> (this),
49:             parser::get_tname (symbol),
50:             lloc.filenr, lloc.linenr, lloc.offset,
51:             lexinfo->c_str());
52:    for (size_t child = 0; child < children.size(); ++child) {
53:       fprintf (outfile, " %p",
54:                static_cast<const void*> (children.at(child)));
55:    }
56: }
57:
58: void astree::dump_tree (FILE* outfile, int depth) {
59:    fprintf (outfile, "%*s", depth * 3, "");
60:    dump_node (outfile);
61:    fprintf (outfile, "\n");
62:    for (astree* child: children) child->dump_tree (outfile, depth + 1);
63:    fflush (nullptr);
64: }
65:
66: void astree::dump (FILE* outfile, astree* tree) {
67:    if (tree == nullptr) fprintf (outfile, "nullptr");
68:                    else tree->dump_node (outfile);
69: }
70:
71: void astree::print (FILE* outfile, astree* tree, int depth) {
72:    fprintf (outfile, "; %*s", depth * 3, "");
73:    fprintf (outfile, "%s \"%s\" (%zd.%zd.%zd)\n",
74:             parser::get_tname (tree->symbol), tree->lexinfo->c_str(),
75:             tree->lloc.filenr, tree->lloc.linenr, tree->lloc.offset);
76:    for (astree* child: tree->children) {
77:       astree::print (outfile, child, depth + 1);
78:    }
79: }
80:
81: void destroy (astree* tree1, astree* tree2) {
82:    if (tree1 != nullptr) delete tree1;
83:    if (tree2 != nullptr) delete tree2;
84: }
85:
86: void errllocprintf (const location& lloc, const char* format,
87:                     const char* arg) {
88:    static char buffer[0x1000];
89:    assert (sizeof buffer > strlen (format) + strlen (arg));
90:    snprintf (buffer, sizeof buffer, format, arg);
91:    errprintf ("%s:%zd.%zd: %s",
92:               lexer::filename (lloc.filenr), lloc.linenr, lloc.offset,
93:               buffer);
94: }
```

```
 1: // $Id: lyutils.h,v 1.6 2017-10-05 16:39:39-07 - - $
 2:
 3: #ifndef __UTILS_H__
 4: #define __UTILS_H__
 5:
 6: // Lex and Yacc interface utility.
 7:
 8: #include <string>
 9: #include <vector>
10: using namespace std;
11:
12: #include <stdio.h>
13:
14: #include "astree.h"
15: #include "auxlib.h"
16:
17: #define YYEOF 0
18:
19: extern FILE* yyin;
20: extern char* yytext;
21: extern int yy_flex_debug;
22: extern int yydebug;
23: extern size_t yyleng;
24:
25: int yylex();
26: int yylex_destroy();
27: int yyparse();
28: void yyerror (const char* message);
29:
30: struct lexer {
31:    static bool interactive;
32:    static location lloc;
33:    static size_t last_yyleng;
34:    static vector<string> filenames;
35:    static const string* filename (int filenr);
36:    static void newfilename (const string& filename);
37:    static void advance();
38:    static void newline();
39:    static void badchar (unsigned char bad);
40:    static void badtoken (char* lexeme);
41:    static void include();
42: };
43:
44: struct parser {
45:    static astree* root;
46:    static const char* get_tname (int symbol);
47: };
48:
49: #define YYSTYPE_IS_DECLARED
50: typedef astree* YYSTYPE;
51: #include "yyparse.h"
52:
53: #endif
54:
```

```
 1: // $Id: lyutils.cpp,v 1.4 2019-03-15 14:32:40-07 - - $
 2:
 3: #include <assert.h>
 4: #include <ctype.h>
 5: #include <stdio.h>
 6: #include <stdlib.h>
 7: #include <string.h>
 8:
 9: #include "auxlib.h"
10: #include "lyutils.h"
11:
12: bool lexer::interactive = true;
13: location lexer::lloc = {0, 1, 0};
14: size_t lexer::last_yyleng = 0;
15: vector<string> lexer::filenames;
16:
17: astree* parser::root = nullptr;
18:
19: const string* lexer::filename (int filenr) {
20:    return &lexer::filenames.at(filenr);
21: }
22:
23: void lexer::newfilename (const string& filename) {
24:    lexer::lloc.filenr = lexer::filenames.size();
25:    lexer::filenames.push_back (filename);
26: }
27:
28: void lexer::advance() {
29:    if (not interactive) {
30:       if (lexer::lloc.offset == 0) {
31:          printf (";%2zd.%3zd: ",
32:                  lexer::lloc.filenr, lexer::lloc.linenr);
33:       }
34:       printf ("%s", yytext);
35:    }
36:    lexer::lloc.offset += last_yyleng;
37:    last_yyleng = yyleng;
38: }
39:
40: void lexer::newline() {
41:    ++lexer::lloc.linenr;
42:    lexer::lloc.offset = 0;
43: }
44:
45: void lexer::badchar (unsigned char bad) {
46:    char buffer[16];
47:    snprintf (buffer, sizeof buffer,
48:              isgraph (bad) ? "%c" : "\\%03o", bad);
49:    errllocprintf (lexer::lloc, "invalid source character (%s)\n",
50:                   buffer);
51: }
52:
```

```
53:
54: void lexer::badtoken (char* lexeme) {
55:    errllocprintf (lexer::lloc, "invalid token (%s)\n", lexeme);
56: }
57:
58: void lexer::include() {
59:    size_t linenr;
60:    static char filename[0x1000];
61:    assert (sizeof filename > strlen (yytext));
62:    int scan_rc = sscanf (yytext, "# %zu \"%[^\"]\"", &linenr, filename);
63:    if (scan_rc != 2) {
64:       errprintf ("%s: invalid directive, ignored\n", yytext);
65:    }else {
66:       if (yy_flex_debug) {
67:          fprintf (stderr, "--included # %zd \"%s\"\n",
68:                   linenr, filename);
69:       }
70:       lexer::lloc.linenr = linenr - 1;
71:       lexer::newfilename (filename);
72:    }
73: }
74:
75: void yyerror (const char* message) {
76:    assert (not lexer::filenames.empty());
77:    errllocprintf (lexer::lloc, "%s\n", message);
78: }
79:
```

```
 1: // $Id: string_set.h,v 1.2 2016-08-18 15:12:57-07 - - $
 2:
 3: #ifndef __STRING_SET__
 4: #define __STRING_SET__
 5:
 6: #include <string>
 7: #include <unordered_set>
 8: using namespace std;
 9:
10: #include <stdio.h>
11:
12: struct string_set {
13:    string_set();
14:    static unordered_set<string> set;
15:    static const string* intern (const char*);
16:    static void dump (FILE*);
17: };
18:
19: #endif
20:
```

```
 1: // $Id: string_set.cpp,v 1.5 2019-03-15 14:32:40-07 - - $
 2:
 3: #include <string>
 4: #include <unordered_set>
 5: using namespace std;
 6:
 7: #include "auxlib.h"
 8: #include "string_set.h"
 9:
10: unordered_set<string> string_set::set;
11:
12: string_set::string_set() {
13:    set.max_load_factor (0.5);
14: }
15:
16: const string* string_set::intern (const char* string) {
17:    auto handle = set.insert (string);
18:    DEBUGF ('s', "inserted \"%s\" %s\n", handle.first->c_str(),
19:            handle.second ? "newly inserted" : "already there");
20:    return &*handle.first;
21: }
22:
23: void string_set::dump (FILE* out) {
24:    static unordered_set<string>::hasher hash_fn
25:                = string_set::set.hash_function();
26:    size_t max_bucket_size = 0;
27:    for (size_t bucket = 0; bucket < set.bucket_count(); ++bucket) {
28:       bool need_index = true;
29:       size_t curr_size = set.bucket_size (bucket);
30:       if (max_bucket_size < curr_size) max_bucket_size = curr_size;
31:       for (auto itor = set.cbegin (bucket);
32:            itor != set.cend (bucket); ++itor) {
33:          if (need_index) fprintf (out, "string_set[%4zu]: ", bucket);
34:                     else fprintf (out, "          %4s   ", "");
35:          need_index = false;
36:          const string* str = &*itor;
37:          fprintf (out, "%22zu %p->\"%s\"\n", hash_fn(*str),
38:                   reinterpret_cast<const void*> (str), str->c_str());
39:       }
40:    }
41:    fprintf (out, "load_factor = %.3f\n", set.load_factor());
42:    fprintf (out, "bucket_count = %zu\n", set.bucket_count());
43:    fprintf (out, "max_bucket_size = %zu\n", max_bucket_size);
44: }
45:
```

```
 1: // $Id: emitter.h,v 1.1 2015-07-09 14:08:38-07 - - $
 2:
 3: #ifndef __EMIT_H__
 4: #define __EMIT_H__
 5:
 6: #include "astree.h"
 7:
 8: void emit_sm_code (astree*);
 9:
10: #endif
11:
```

```
 1: // $Id: emitter.cpp,v 1.5 2017-10-05 16:39:39-07 - - $
 2:
 3: #include <assert.h>
 4: #include <stdio.h>
 5:
 6: #include "astree.h"
 7: #include "emitter.h"
 8: #include "auxlib.h"
 9: #include "lyutils.h"
10:
11: void emit (astree* root);
12:
13: void emit_insn (const char* opcode, const char* operand, astree* tree) {
14:    printf ("%-10s%-10s%-20s; %s %zd.%zd\n", "",
15:             opcode, operand,
16:             lexer::filename (tree->lloc.filenr)->c_str(),
17:             tree->lloc.linenr, tree->lloc.offset);
18: }
19:
20: void postorder (astree* tree) {
21:    assert (tree != nullptr);
22:    for (size_t child = 0; child < tree->children.size(); ++child) {
23:       emit (tree->children.at(child));
24:    }
25: }
26:
27: void postorder_emit_stmts (astree* tree) {
28:    postorder (tree);
29: }
30:
31: void postorder_emit_oper (astree* tree, const 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, const char* opcode) {
42:    emit_insn (opcode, tree->lexinfo->c_str(), tree);
43: }
44:
45: void emit_assign (astree* tree) {
46:    assert (tree->children.size() == 2);
47:    astree* left = tree->children.at(0);
48:    emit (tree->children.at(1));
49:    if (left->symbol != IDENT) {
50:       errllocprintf (left->lloc, "%s\n",
51:                      "left operand of '=' not an identifier");
52:    }else{
53:       emit_insn ("popvar", left->lexinfo->c_str(), left);
54:    }
55: }
56:
```

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

```
 1: // $Id: auxlib.h,v 1.5 2017-10-11 14:33:45-07 - - $
 2:
 3: #ifndef __AUXLIB_H__
 4: #define __AUXLIB_H__
 5:
 6: #include <string>
 7: using namespace std;
 8:
 9: #include <stdarg.h>
10:
11: //
12: // DESCRIPTION
13: //    Auxiliary library containing miscellaneous useful things.
14: //
15:
16: //
17: // Error message and exit status utility.
18: //
19:
20: struct exec {
21:    static string execname;
22:    static int exit_status;
23: };
24:
25: void veprintf (const char* format, va_list args);
26: // Prints a message to stderr using the vector form of
27: // argument list.
28:
29: void eprintf (const char* format, ...);
30: // Print a message to stderr according to the printf format
31: // specified.  Usually called for debug output.
32: // Precedes the message by the program name if the format
33: // begins with the characters `%:'.
34:
35: void errprintf (const char* format, ...);
36: // Print an error message according to the printf format
37: // specified, using eprintf.
38: // Sets the exitstatus to EXIT_FAILURE.
39:
40: void syserrprintf (const char* object);
41: // Print a message resulting from a bad system call.  The
42: // object is the name of the object causing the problem and
43: // the reason is taken from the external variable errno.
44: // Sets the exit status to EXIT_FAILURE.
45:
46: void eprint_status (const char* command, int status);
47: // Print the status returned by wait(2) from a subprocess.
48:
```

```
49:
50: //
51: // Support for stub messages.
52: //
53: #define STUBPRINTF(...) \
54:         __stubprintf (__FILE__, __LINE__, __PRETTY_FUNCTION__, \
55:                       __VA_ARGS__)
56: void __stubprintf (const char* file, int line, const char* func,
57:                    const char* format, ...);
58:
59: //
60: // Debugging utility.
61: //
62:
63: void set_debugflags (const char* flags);
64: // Sets a string of debug flags to be used by DEBUGF statements.
65: // Uses the address of the string, and does not copy it, so
66: // it must not be dangling.  If a particular debug flag has
67: // been set, messages are printed.  The format is identical to
68: // printf format.  The flag "@" turns on all flags.
69:
70: bool is_debugflag (char flag);
71: // Checks to see if a debugflag is set.
72:
73: #ifdef NDEBUG
74: // Do not generate any code.
75: #define DEBUGF(FLAG,...)   /**/
76: #define DEBUGSTMT(FLAG,STMTS) /**/
77: #else
78: // Generate debugging code.
79: void __debugprintf (char flag, const char* file, int line,
80:                     const char* func, const char* format, ...);
81: #define DEBUGF(FLAG,...) \
82:         __debugprintf (FLAG, __FILE__, __LINE__, __PRETTY_FUNCTION__, \
83:                        __VA_ARGS__)
84: #define DEBUGSTMT(FLAG,STMTS) \
85:         if (is_debugflag (FLAG)) { DEBUGF (FLAG, "\n"); STMTS }
86: #endif
87:
88: #endif
89:
```

```
 1: // $Id: auxlib.cpp,v 1.4 2019-04-10 15:50:29-07 - - $
 2:
 3: #include <assert.h>
 4: #include <errno.h>
 5: #include <libgen.h>
 6: #include <limits.h>
 7: #include <stdarg.h>
 8: #include <stdio.h>
 9: #include <stdlib.h>
10: #include <string.h>
11: #include <wait.h>
12:
13: #include "auxlib.h"
14:
15: string exec::execname;
16: int exec::exit_status = EXIT_SUCCESS;
17:
18: const char* debugflags = "";
19: bool alldebugflags = false;
20:
21: static void eprint_signal (const char* kind, int signal) {
22:    eprintf (", %s %d", kind, signal);
23:    const char* sigstr = strsignal (signal);
24:    if (sigstr != nullptr) fprintf (stderr, " %s", sigstr);
25: }
26:
27: void eprint_status (const char* command, int status) {
28:    if (status == 0) return;
29:    eprintf ("%s: status 0x%04X", command, status);
30:    if (WIFEXITED (status)) {
31:       eprintf (", exit %d", WEXITSTATUS (status));
32:    }
33:    if (WIFSIGNALED (status)) {
34:       eprint_signal ("Terminated", WTERMSIG (status));
35:       #ifdef WCOREDUMP
36:       if (WCOREDUMP (status)) eprintf (", core dumped");
37:       #endif
38:    }
39:    if (WIFSTOPPED (status)) {
40:       eprint_signal ("Stopped", WSTOPSIG (status));
41:    }
42:    if (WIFCONTINUED (status)) {
43:       eprintf (", Continued");
44:    }
45:    eprintf ("\n");
46: }
47:
```

```
48:
49: void veprintf (const char* format, va_list args) {
50:    assert (exec::execname.size() != 0);
51:    assert (format != nullptr);
52:    fflush (nullptr);
53:    if (strstr (format, "%:") == format) {
54:       fprintf (stderr, "%s: ", exec::execname.c_str());
55:       format += 2;
56:    }
57:    vfprintf (stderr, format, args);
58:    fflush (nullptr);
59: }
60:
61: void eprintf (const char* format, ...) {
62:    va_list args;
63:    va_start (args, format);
64:    veprintf (format, args);
65:    va_end (args);
66: }
67:
68: void errprintf (const char* format, ...) {
69:    va_list args;
70:    va_start (args, format);
71:    veprintf (format, args);
72:    va_end (args);
73:    exec::exit_status = EXIT_FAILURE;
74: }
75:
76: void syserrprintf (const char* object) {
77:    errprintf ("%:%s: %s\n", object, strerror (errno));
78: }
79:
80: void __stubprintf (const char* file, int line, const char* func,
81:                    const char* format, ...) {
82:    va_list args;
83:    fflush (nullptr);
84:    printf ("%s: %s[%d] %s: ", exec::execname.c_str(), file, line, func);
85:    va_start (args, format);
86:    vprintf (format, args);
87:    va_end (args);
88:    fflush (nullptr);
89: }
90:
```

```
 91:
 92: void set_debugflags (const char* flags) {
 93:    debugflags = flags;
 94:    assert (debugflags != nullptr);
 95:    if (strchr (debugflags, '@') != nullptr) alldebugflags = true;
 96:    DEBUGF ('x', "Debugflags = \"%s\", all = %d\n",
 97:            debugflags, alldebugflags);
 98: }
 99:
100: bool is_debugflag (char flag) {
101:    return alldebugflags or strchr (debugflags, flag) != nullptr;
102: }
103:
104: void __debugprintf (char flag, const char* file, int line,
105:                     const char* func, const char* format, ...) {
106:    va_list args;
107:    if (not is_debugflag (flag)) return;
108:    fflush (nullptr);
109:    va_start (args, format);
110:    fprintf (stderr, "DEBUGF(%c): %s[%d] %s():\n",
111:             flag, file, line, func);
112:    vfprintf (stderr, format, args);
113:    va_end (args);
114:    fflush (nullptr);
115: }
116:
```

```
 1: // $Id: main.cpp,v 1.18 2017-10-19 16:02:14-07 - - $
 2:
 3: #include <string>
 4: #include <vector>
 5: using namespace std;
 6:
 7: #include <assert.h>
 8: #include <errno.h>
 9: #include <libgen.h>
10: #include <stdio.h>
11: #include <stdlib.h>
12: #include <string.h>
13: #include <unistd.h>
14:
15: #include "astree.h"
16: #include "auxlib.h"
17: #include "emitter.h"
18: #include "lyutils.h"
19: #include "string_set.h"
20:
21: const string cpp_name = "/usr/bin/cpp";
22: string cpp_command;
23:
24: // Open a pipe from the C preprocessor.
25: // Exit failure if can't.
26: // Assigns opened pipe to FILE* yyin.
27: void cpp_popen (const char* filename) {
28:    cpp_command = cpp_name + " " + filename;
29:    yyin = popen (cpp_command.c_str(), "r");
30:    if (yyin == nullptr) {
31:       syserrprintf (cpp_command.c_str());
32:       exit (exec::exit_status);
33:    }else {
34:       if (yy_flex_debug) {
35:          fprintf (stderr, "-- popen (%s), fileno(yyin) = %d\n",
36:                   cpp_command.c_str(), fileno (yyin));
37:       }
38:       lexer::newfilename (cpp_command);
39:    }
40: }
41:
42: void cpp_pclose() {
43:    int pclose_rc = pclose (yyin);
44:    eprint_status (cpp_command.c_str(), pclose_rc);
45:    if (pclose_rc != 0) exec::exit_status = EXIT_FAILURE;
46: }
47:
```

```
 48:
 49: void scan_opts (int argc, char** argv) {
 50:    opterr = 0;
 51:    yy_flex_debug = 0;
 52:    yydebug = 0;
 53:    lexer::interactive = isatty (fileno (stdin))
 54:                         and isatty (fileno (stdout));
 55:    for(;;) {
 56:       int opt = getopt (argc, argv, "@:ly");
 57:       if (opt == EOF) break;
 58:       switch (opt) {
 59:          case '@': set_debugflags (optarg);   break;
 60:          case 'l': yy_flex_debug = 1;         break;
 61:          case 'y': yydebug = 1;               break;
 62:          default:  errprintf ("bad option (%c)\n", optopt); break;
 63:       }
 64:    }
 65:    if (optind > argc) {
 66:       errprintf ("Usage: %s [-ly] [filename]\n",
 67:                  exec::execname.c_str());
 68:       exit (exec::exit_status);
 69:    }
 70:    const char* filename = optind == argc ? "-" : argv[optind];
 71:    cpp_popen (filename);
 72: }
 73:
 74: int main (int argc, char** argv) {
 75:    exec::execname = basename (argv[0]);
 76:    if (yydebug or yy_flex_debug) {
 77:       fprintf (stderr, "Command:");
 78:       for (char** arg = &argv[0]; arg < &argv[argc]; ++arg) {
 79:             fprintf (stderr, " %s", *arg);
 80:       }
 81:       fprintf (stderr, "\n");
 82:    }
 83:    scan_opts (argc, argv);
 84:    int parse_rc = yyparse();
 85:    cpp_pclose();
 86:    yylex_destroy();
 87:    if (yydebug or yy_flex_debug) {
 88:       fprintf (stderr, "Dumping parser::root:\n");
 89:       if (parser::root != nullptr) parser::root->dump_tree (stderr);
 90:       fprintf (stderr, "Dumping string_set:\n");
 91:       string_set::dump (stderr);
 92:    }
 93:    if (parse_rc) {
 94:       errprintf ("parse failed (%d)\n", parse_rc);
 95:    }else {
 96:       astree::print (stdout, parser::root);
 97:       emit_sm_code (parser::root);
 98:       delete parser::root;
 99:    }
100:    return exec::exit_status;
101: }
102:
```

```
# $Id: Makefile,v 1.38 2019-04-10 15:51:41-07 - - $

DEPSFILE  = Makefile.deps
NOINCLUDE = ci clean spotless
NEEDINCL  = ${filter ${NOINCLUDE}, ${MAKECMDGOALS}}
WARNING   = -Wall -Wextra -Wpedantic -Wshadow -Wold-style-cast
CPP       = g++ -std=gnu++17 -g -O0
CPPWARN   = ${CPP} ${WARNING} -fdiagnostics-color=never
CPPYY     = ${CPP} -Wno-sign-compare -Wno-register
MKDEPS    = g++ -std=gnu++17 -MM
GRIND     = valgrind --leak-check=full --show-reachable=yes
FLEX      = flex --outfile=${LEXCPP}
BISON     = bison --defines=${PARSEHDR} --output=${PARSECPP}

MODULES   = astree lyutils string_set emitter auxlib
HDRSRC    = ${MODULES:=.h}
CPPSRC    = ${MODULES:=.cpp} main.cpp
FLEXSRC   = lexer.l
BISONSRC  = parser.y
PARSEHDR  = yyparse.h
LEXCPP    = yylex.cpp
PARSECPP  = yyparse.cpp
CGENS     = ${LEXCPP} ${PARSECPP}
ALLGENS   = ${PARSEHDR} ${CGENS}
EXECBIN   = zexprsm
ALLCSRC   = ${CPPSRC} ${CGENS}
OBJECTS   = ${ALLCSRC:.cpp=.o}
LEXOUT    = yylex.output
PARSEOUT  = yyparse.output
REPORTS   = ${LEXOUT} ${PARSEOUT}
MODSRC    = ${foreach MOD, ${MODULES}, ${MOD}.h ${MOD}.cpp}
MISCSRC   = ${filter-out ${MODSRC}, ${HDRSRC} ${CPPSRC}}
ALLSRC    = README ${FLEXSRC} ${BISONSRC} ${MODSRC} ${MISCSRC} Makefile
TESTINS   = ${wildcard test*.in}
EXECTEST  = ${EXECBIN} -ly
LISTSRC   = ${ALLSRC} ${DEPSFILE} ${PARSEHDR}

all : ${EXECBIN}

${EXECBIN} : ${OBJECTS}
        ${CPPWARN} -o${EXECBIN} ${OBJECTS}

yylex.o : yylex.cpp
        ${CPPYY} -c $<

yyparse.o : yyparse.cpp
        ${CPPYY} -c $<

%.o : %.cpp
        - cpplint.py.perl $<
        - checksource $<
        ${CPPWARN} -c $<

```

```
 54:
 55: ${LEXCPP} : ${FLEXSRC}
 56:         ${FLEX} ${FLEXSRC}
 57:
 58: ${PARSECPP} ${PARSEHDR} : ${BISONSRC}
 59:         ${BISON} ${BISONSRC}
 60:
 61: ci : ${ALLSRC} ${TESTINS}
 62:         - checksource ${ALLSRC}
 63:         cid + ${ALLSRC} ${TESTINS} test?.inh
 64:
 65: lis : ${LISTSRC} tests
 66:         mkpspdf List.source.ps ${LISTSRC}
 67:         mkpspdf List.output.ps ${REPORTS} \
 68:                 ${foreach test, ${TESTINS:.in=}, \
 69:                 ${patsubst %, ${test}.%, in out err log}}
 70:
 71: clean :
 72:         - rm ${OBJECTS} ${ALLGENS} ${REPORTS} ${DEPSFILE} core
 73:         - rm ${foreach test, ${TESTINS:.in=}, \
 74:                 ${patsubst %, ${test}.%, out err log}}
 75:
 76: spotless : clean
 77:         - rm ${EXECBIN} List.*.ps List.*.pdf
 78:
 79: deps : ${ALLCSRC}
 80:         @ echo "# ${DEPSFILE} created `date` by ${MAKE}" >${DEPSFILE}
 81:         ${MKDEPS} ${ALLCSRC} >>${DEPSFILE}
 82:
 83: ${DEPSFILE} :
 84:         @ touch ${DEPSFILE}
 85:         ${MAKE} --no-print-directory deps
 86:
 87: tests : ${EXECBIN}
 88:         touch ${TESTINS}
 89:         make --no-print-directory ${TESTINS:.in=.out}
 90:
 91: %.out %.err : %.in
 92:         ${GRIND} --log-file=$*.log ${EXECTEST} $< 1>$*.out 2>$*.err; \
 93:         echo EXIT STATUS = $$? >>$*.log
 94:
 95: again :
 96:         gmake --no-print-directory spotless deps ci all lis
 97:
 98: ifeq "${NEEDINCL}" ""
 99: include ${DEPSFILE}
100: endif
101:
```

```
 1: # Makefile.deps created Wed Apr 10 15:51:41 PDT 2019 by gmake
 2: astree.o: astree.cpp astree.h auxlib.h string_set.h lyutils.h yyparse.h
 3: lyutils.o: lyutils.cpp auxlib.h lyutils.h astree.h yyparse.h
 4: string_set.o: string_set.cpp auxlib.h string_set.h
 5: emitter.o: emitter.cpp astree.h auxlib.h emitter.h lyutils.h yyparse.h
 6: auxlib.o: auxlib.cpp auxlib.h
 7: main.o: main.cpp astree.h auxlib.h emitter.h lyutils.h yyparse.h \
 8:  string_set.h
 9: yylex.o: yylex.cpp lyutils.h astree.h auxlib.h yyparse.h
10: yyparse.o: yyparse.cpp astree.h auxlib.h lyutils.h yyparse.h
```

```
    1: /* A Bison parser, made by GNU Bison 3.0.4.  */
    2:
    3: /* Bison interface for Yacc-like parsers in C
    4:
    5:    Copyright (C) 1984, 1989-1990, 2000-2015 Free Software Foundation, In
c.
    6:
    7:    This program is free software: you can redistribute it and/or modify
    8:    it under the terms of the GNU General Public License as published by
    9:    the Free Software Foundation, either version 3 of the License, or
   10:    (at your option) any later version.
   11:
   12:    This program is distributed in the hope that it will be useful,
   13:    but WITHOUT ANY WARRANTY; without even the implied warranty of
   14:    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   15:    GNU General Public License for more details.
   16:
   17:    You should have received a copy of the GNU General Public License
   18:    along with this program.  If not, see <http://www.gnu.org/licenses/>.
  */
   19:
   20: /* As a special exception, you may create a larger work that contains
   21:    part or all of the Bison parser skeleton and distribute that work
   22:    under terms of your choice, so long as that work isn't itself a
   23:    parser generator using the skeleton or a modified version thereof
   24:    as a parser skeleton.  Alternatively, if you modify or redistribute
   25:    the parser skeleton itself, you may (at your option) remove this
   26:    special exception, which will cause the skeleton and the resulting
   27:    Bison output files to be licensed under the GNU General Public
   28:    License without this special exception.
   29:
   30:    This special exception was added by the Free Software Foundation in
   31:    version 2.2 of Bison.  */
   32:
   33: #ifndef YY_YY_YYPARSE_H_INCLUDED
   34: # define YY_YY_YYPARSE_H_INCLUDED
   35: /* Debug traces.  */
   36: #ifndef YYDEBUG
   37: # define YYDEBUG 1
   38: #endif
   39: #if YYDEBUG
   40: extern int yydebug;
   41: #endif
   42:
   43: /* Token type.  */
   44: #ifndef YYTOKENTYPE
   45: # define YYTOKENTYPE
   46:   enum yytokentype
   47:   {
   48:     ROOT = 258,
   49:     IDENT = 259,
   50:     NUMBER = 260,
   51:     POS = 261,
   52:     NEG = 262
   53:   };
   54: #endif
   55:
   56: /* Value type.  */
```

```
57: #if ! defined YYSTYPE && ! defined YYSTYPE_IS_DECLARED
58: typedef int YYSTYPE;
59: # define YYSTYPE_IS_TRIVIAL 1
60: # define YYSTYPE_IS_DECLARED 1
61: #endif
62:
63:
64: extern YYSTYPE yylval;
65:
66: int yyparse (void);
67:
68: #endif /* !YY_YY_YYPARSE_H_INCLUDED  */
```