```
 1: // $Id: astree.h,v 1.7 2016-10-06 16:13:39-07 - - $
 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:
```

```cpp
 1: // $Id: astree.cpp,v 1.8 2016-09-21 17:13:03-07 - - $
 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:             this, parser::get_tname (symbol),
49:             lloc.filenr, lloc.linenr, lloc.offset,
50:             lexinfo->c_str());
51:    for (size_t child = 0; child < children.size(); ++child) {
52:       fprintf (outfile, " %p", children.at(child));
53:    }
54: }
55:
56: void astree::dump_tree (FILE* outfile, int depth) {
57:    fprintf (outfile, "%*s", depth * 3, "");
58:    dump_node (outfile);
59:    fprintf (outfile, "\n");
60:    for (astree* child: children) child->dump_tree (outfile, depth + 1);
61:    fflush (NULL);
62: }
63:
64: void astree::dump (FILE* outfile, astree* tree) {
65:    if (tree == nullptr) fprintf (outfile, "nullptr");
66:                  else tree->dump_node (outfile);
67: }
68:
69: void astree::print (FILE* outfile, astree* tree, int depth) {
70:    fprintf (outfile, "; %*s", depth * 3, "");
71:    fprintf (outfile, "%s \"%s\" (%zd.%zd.%zd)\n",
72:             parser::get_tname (tree->symbol), tree->lexinfo->c_str(),
73:             tree->lloc.filenr, tree->lloc.linenr, tree->lloc.offset);
74:    for (astree* child: tree->children) {
75:       astree::print (outfile, child, depth + 1);
76:    }
77: }
78:
79: void destroy (astree* tree1, astree* tree2) {
80:    if (tree1 != nullptr) delete tree1;
81:    if (tree2 != nullptr) delete tree2;
82: }
83:
84: void errllocprintf (const location& lloc, const char* format,
85:                     const char* arg) {
86:    static char buffer[0x1000];
87:    assert (sizeof buffer > strlen (format) + strlen (arg));
88:    snprintf (buffer, sizeof buffer, format, arg);
89:    errprintf ("%s:%zd.%zd: %s",
90:               lexer::filename (lloc.filenr), lloc.linenr, lloc.offset,
91:               buffer);
92: }
```

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

```
47:
48: //
49: // Support for stub messages.
50: //
51: #define STUBPRINTF(...) \
52:         __stubprintf (__FILE__, __LINE__, __func__, __VA_ARGS__)
53: void __stubprintf (const char* file, int line, const char* func,
54:                    const char* format, ...);
55:
56: //
57: // Debugging utility.
58: //
59:
60: void set_debugflags (const char* flags);
61: // Sets a string of debug flags to be used by DEBUGF statements.
62: // Uses the address of the string, and does not copy it, so
63: // it must not be dangling.  If a particular debug flag has
64: // been set, messages are printed.  The format is identical to
65: // printf format.  The flag "@" turns on all flags.
66:
67: bool is_debugflag (char flag);
68: // Checks to see if a debugflag is set.
69:
70: #ifdef NDEBUG
71: // Do not generate any code.
72: #define DEBUGF(FLAG,...)    /**/
73: #define DEBUGSTMT(FLAG,STMTS) /**/
74: #else
75: // Generate debugging code.
76: void __debugprintf (char flag, const char* file, int line,
77:                     const char* func, const char* format, ...);
78: #define DEBUGF(FLAG,...) \
79:         __debugprintf (FLAG, __FILE__, __LINE__, __func__, \
80:                        __VA_ARGS__)
81: #define DEBUGSTMT(FLAG,STMTS) \
82:         if (is_debugflag (FLAG)) { DEBUGF (FLAG, "\n"); STMTS }
83: #endif
84:
85: #endif
86:
```

```
 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: string exec::execname;
15: int exec::exit_status = EXIT_SUCCESS;
16:
17: const char* debugflags = "";
18: bool alldebugflags = false;
19:
20: static void eprint_signal (const char* kind, int signal) {
21:     eprintf (", %s %d", kind, signal);
22:     const char* sigstr = strsignal (signal);
23:     if (sigstr != NULL) fprintf (stderr, " %s", sigstr);
24: }
25:
26: void eprint_status (const char* command, int status) {
27:     if (status == 0) return;
28:     eprintf ("%s: status 0x%04X", command, status);
29:     if (WIFEXITED (status)) {
30:         eprintf (", exit %d", WEXITSTATUS (status));
31:     }
32:     if (WIFSIGNALED (status)) {
33:         eprint_signal ("Terminated", WTERMSIG (status));
34:         #ifdef WCOREDUMP
35:         if (WCOREDUMP (status)) eprintf (", core dumped");
36:         #endif
37:     }
38:     if (WIFSTOPPED (status)) {
39:         eprint_signal ("Stopped", WSTOPSIG (status));
40:     }
41:     if (WIFCONTINUED (status)) {
42:         eprintf (", Continued");
43:     }
44:     eprintf ("\n");
45: }
46:
47: void veprintf (const char* format, va_list args) {
48:     assert (exec::execname.size() != 0);
49:     assert (format != NULL);
50:     fflush (NULL);
51:     if (strstr (format, "%:") == format) {
52:         fprintf (stderr, "%s: ", exec::execname.c_str());
53:         format += 2;
54:     }
55:     vfprintf (stderr, format, args);
56:     fflush (NULL);
57: }
58:
```

```
59: void eprintf (const char* format, ...) {
60:    va_list args;
61:    va_start (args, format);
62:    veprintf (format, args);
63:    va_end (args);
64: }
65:
66: void errprintf (const char* format, ...) {
67:    va_list args;
68:    va_start (args, format);
69:    veprintf (format, args);
70:    va_end (args);
71:    exec::exit_status = EXIT_FAILURE;
72: }
73:
74: void syserrprintf (const char* object) {
75:    errprintf ("%:%s: %s\n", object, strerror (errno));
76: }
77:
78: void __stubprintf (const char* file, int line, const char* func,
79:                    const char* format, ...) {
80:    va_list args;
81:    fflush (NULL);
82:    printf ("%s: %s[%d] %s: ", exec::execname.c_str(), file, line, func);
83:    va_start (args, format);
84:    vprintf (format, args);
85:    va_end (args);
86:    fflush (NULL);
87: }
88:
```

```
 89:
 90: void set_debugflags (const char* flags) {
 91:     debugflags = flags;
 92:     if (strchr (debugflags, '@') != NULL) alldebugflags = true;
 93:     DEBUGF ('x', "Debugflags = \"%s\", all = %d\n",
 94:             debugflags, alldebugflags);
 95: }
 96:
 97: bool is_debugflag (char flag) {
 98:     return alldebugflags or strchr (debugflags, flag) != NULL;
 99: }
100:
101: void __debugprintf (char flag, const char* file, int line,
102:                     const char* func, const char* format, ...) {
103:     va_list args;
104:     if (not is_debugflag (flag)) return;
105:     fflush (NULL);
106:     va_start (args, format);
107:     fprintf (stderr, "DEBUGF(%c): %s[%d] %s():\n",
108:              flag, file, line, func);
109:     vfprintf (stderr, format, args);
110:     va_end (args);
111:     fflush (NULL);
112: }
113:
```

```
 1: // $Id: lyutils.h,v 1.10 2016-10-06 16:42:53-07 - - $
 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: extern FILE* yyin;
18: extern char* yytext;
19: extern int yy_flex_debug;
20: extern int yydebug;
21: extern size_t yyleng;
22:
23: int yylex();
24: int yylex_destroy();
25: int yyparse();
26: void yyerror (const char* message);
27:
28: struct lexer {
29:    static bool interactive;
30:    static location lloc;
31:    static size_t last_yyleng;
32:    static vector<string> filenames;
33:    static const string* filename (int filenr);
34:    static void newfilename (const string& filename);
35:    static void advance();
36:    static void newline();
37:    static void badchar (unsigned char bad);
38:    static void badtoken (char* lexeme);
39:    static void include();
40: };
41:
42: struct parser {
43:    static astree* root;
44:    static const char* get_tname (int symbol);
45: };
46:
47: #define YYSTYPE astree*
48: #include "yyparse.h"
49:
50: #endif
51:
```

```
 1: // $Id: lyutils.cpp,v 1.11 2016-10-06 16:42:53-07 - - $
 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, "# %zd \"%[^\"]\"", &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 1.1 2016-10-06 16:15:22-07 - - $
 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:
```

```cpp
 1: // $Id: string_set.cpp,v 1.2 2016-10-06 16:42:53-07 - - $
 2:
 3: #include <string>
 4: #include <unordered_set>
 5: using namespace std;
 6:
 7: #include "string_set.h"
 8:
 9: unordered_set<string> string_set::set;
10:
11: string_set::string_set() {
12:    set.max_load_factor (0.5);
13: }
14:
15: const string* string_set::intern (const char* string) {
16:    auto handle = set.insert (string);
17:    return &*handle.first;
18: }
19:
20: void string_set::dump (FILE* out) {
21:    static unordered_set<string>::hasher hash_fn
22:                = string_set::set.hash_function();
23:    size_t max_bucket_size = 0;
24:    for (size_t bucket = 0; bucket < set.bucket_count(); ++bucket) {
25:       bool need_index = true;
26:       size_t curr_size = set.bucket_size (bucket);
27:       if (max_bucket_size < curr_size) max_bucket_size = curr_size;
28:       for (auto itor = set.cbegin (bucket);
29:            itor != set.cend (bucket); ++itor) {
30:          if (need_index) fprintf (out, "string_set[%4zu]: ", bucket);
31:                    else fprintf (out, "           %4s   ", "");
32:          need_index = false;
33:          const string* str = &*itor;
34:          fprintf (out, "%22zu %p->\"%s\"\n", hash_fn(*str),
35:                   str, str->c_str());
36:       }
37:    }
38:    fprintf (out, "load_factor = %.3f\n", set.load_factor());
39:    fprintf (out, "bucket_count = %zu\n", set.bucket_count());
40:    fprintf (out, "max_bucket_size = %zu\n", max_bucket_size);
41: }
42:
```

```
 1: /* A Bison parser, made by GNU Bison 2.7.  */
 2:
 3: /* Bison interface for Yacc-like parsers in C
 4:
 5:       Copyright (C) 1984, 1989-1990, 2000-2012 Free Software Foundation,
Inc.
 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: /* Enabling traces.  */
36: #ifndef YYDEBUG
37: # define YYDEBUG 1
38: #endif
39: #if YYDEBUG
40: extern int yydebug;
41: #endif
42:
43: /* Tokens.  */
44: #ifndef YYTOKENTYPE
45: # define YYTOKENTYPE
46:    /* Put the tokens into the symbol table, so that GDB and other debugg
ers
47:      know about them.  */
48:    enum yytokentype {
49:      TOK_VOID = 258,
50:      TOK_CHAR = 259,
51:      TOK_INT = 260,
52:      TOK_STRING = 261,
53:      TOK_IF = 262,
54:      TOK_ELSE = 263,
55:      TOK_WHILE = 264,
```

```
 56:        TOK_RETURN = 265,
 57:        TOK_STRUCT = 266,
 58:        TOK_NULL = 267,
 59:        TOK_NEW = 268,
 60:        TOK_ARRAY = 269,
 61:        TOK_EQ = 270,
 62:        TOK_NE = 271,
 63:        TOK_LT = 272,
 64:        TOK_LE = 273,
 65:        TOK_GT = 274,
 66:        TOK_GE = 275,
 67:        TOK_IDENT = 276,
 68:        TOK_INTCON = 277,
 69:        TOK_CHARCON = 278,
 70:        TOK_STRINGCON = 279,
 71:        TOK_BLOCK = 280,
 72:        TOK_CALL = 281,
 73:        TOK_IFELSE = 282,
 74:        TOK_INITDECL = 283,
 75:        TOK_POS = 284,
 76:        TOK_NEG = 285,
 77:        TOK_NEWARRAY = 286,
 78:        TOK_TYPEID = 287,
 79:        TOK_FIELD = 288,
 80:        TOK_ORD = 289,
 81:        TOK_CHR = 290,
 82:        TOK_ROOT = 291
 83:     };
 84: #endif
 85:
 86:
 87: #if ! defined YYSTYPE && ! defined YYSTYPE_IS_DECLARED
 88: typedef int YYSTYPE;
 89: # define YYSTYPE_IS_TRIVIAL 1
 90: # define yystype YYSTYPE /* obsolescent; will be withdrawn */
 91: # define YYSTYPE_IS_DECLARED 1
 92: #endif
 93:
 94: extern YYSTYPE yylval;
 95:
 96: #ifdef YYPARSE_PARAM
 97: #if defined __STDC__ || defined __cplusplus
 98: int yyparse (void *YYPARSE_PARAM);
 99: #else
100: int yyparse ();
101: #endif
102: #else /* ! YYPARSE_PARAM */
103: #if defined __STDC__ || defined __cplusplus
104: int yyparse (void);
105: #else
106: int yyparse ();
107: #endif
108: #endif /* ! YYPARSE_PARAM */
109:
110: #endif /* !YY_YY_YYPARSE_H_INCLUDED  */
```

```
 1: %{
 2: // Dummy parser for scanner project.
 3:
 4: #include <cassert>
 5:
 6: #include "lyutils.h"
 7: #include "astree.h"
 8:
 9: %}
10:
11: %debug
12: %defines
13: %error-verbose
14: %token-table
15: %verbose
16:
17: %token TOK_VOID TOK_CHAR TOK_INT TOK_STRING
18: %token TOK_IF TOK_ELSE TOK_WHILE TOK_RETURN TOK_STRUCT
19: %token TOK_NULL TOK_NEW TOK_ARRAY
20: %token TOK_EQ TOK_NE TOK_LT TOK_LE TOK_GT TOK_GE
21: %token TOK_IDENT TOK_INTCON TOK_CHARCON TOK_STRINGCON
22:
23: %token TOK_BLOCK TOK_CALL TOK_IFELSE TOK_INITDECL
24: %token TOK_POS TOK_NEG TOK_NEWARRAY TOK_TYPEID TOK_FIELD
25: %token TOK_ORD TOK_CHR TOK_ROOT
26:
27: %start program
28:
29: %%
30:
31: program : program token | ;
32: token   : '(' | ')' | '[' | ']' | '{' | '}' | ';' | ',' | '.'
33:         | '=' | '+' | '-' | '*' | '/' | '%' | '!'
34:         | TOK_VOID | TOK_CHAR | TOK_INT | TOK_STRING
35:         | TOK_IF | TOK_ELSE | TOK_WHILE | TOK_RETURN | TOK_STRUCT
36:         | TOK_NULL | TOK_NEW | TOK_ARRAY
37:         | TOK_EQ | TOK_NE | TOK_LT | TOK_LE | TOK_GT | TOK_GE
38:         | TOK_IDENT | TOK_INTCON | TOK_CHARCON | TOK_STRINGCON
39:         | TOK_ORD | TOK_CHR | TOK_ROOT
40:         ;
41:
42: %%
```

```
43:
44:
45: const char *get_yytname (int symbol) {
46:    return yytname [YYTRANSLATE (symbol)];
47: }
48:
49:
50: bool is_defined_token (int symbol) {
51:    return YYTRANSLATE (symbol) > YYUNDEFTOK;
52: }
53:
54: /*
55: static void* yycalloc (size_t size) {
56:    void* result = calloc (1, size);
57:    assert (result != nullptr);
58:    return result;
59: }
60: */
61:
```