```
 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:
```

```
 1: // $Id: astree.cpp,v 1.10 2019-04-03 17:17:43-07 - - $
 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: auxlib.h,v 1.10 2017-10-11 14:33:32-07 - - $
 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 1.5 2017-10-11 14:28:23-07 - - $
 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: void veprintf (const char* format, va_list args) {
49:    assert (exec::execname.size() != 0);
50:    assert (format != nullptr);
51:    fflush (nullptr);
52:    if (strstr (format, "%:") == format) {
53:       fprintf (stderr, "%s: ", exec::execname.c_str());
54:       format += 2;
55:    }
56:    vfprintf (stderr, format, args);
57:    fflush (nullptr);
58: }
```

```
59:
60: void eprintf (const char* format, ...) {
61:    va_list args;
62:    va_start (args, format);
63:    veprintf (format, args);
64:    va_end (args);
65: }
66:
67: void errprintf (const char* format, ...) {
68:    va_list args;
69:    va_start (args, format);
70:    veprintf (format, args);
71:    va_end (args);
72:    exec::exit_status = EXIT_FAILURE;
73: }
74:
75: void syserrprintf (const char* object) {
76:    errprintf ("%:%s: %s\n", object, strerror (errno));
77: }
78:
79: void __stubprintf (const char* file, int line, const char* func,
80:                    const char* format, ...) {
81:    va_list args;
82:    fflush (nullptr);
83:    printf ("%s: %s[%d] %s: ", exec::execname.c_str(), file, line, func);
84:    va_start (args, format);
85:    vprintf (format, args);
86:    va_end (args);
87:    fflush (nullptr);
88: }
89:
```

```
 90:
 91: void set_debugflags (const char* flags) {
 92:    debugflags = flags;
 93:    assert (debugflags != nullptr);
 94:    if (strchr (debugflags, '@') != nullptr) alldebugflags = true;
 95:    DEBUGF ('x', "Debugflags = \"%s\", all = %d\n",
 96:            debugflags, alldebugflags);
 97: }
 98:
 99: bool is_debugflag (char flag) {
100:    return alldebugflags or strchr (debugflags, flag) != nullptr;
101: }
102:
103: void __debugprintf (char flag, const char* file, int line,
104:                     const char* func, const char* format, ...) {
105:    va_list args;
106:    if (not is_debugflag (flag)) return;
107:    fflush (nullptr);
108:    va_start (args, format);
109:    fprintf (stderr, "DEBUGF(%c): %s[%d] %s():\n",
110:             flag, file, line, func);
111:    vfprintf (stderr, format, args);
112:    va_end (args);
113:    fflush (nullptr);
114: }
115:
```

```
 1: // $Id: lyutils.h,v 1.11 2017-10-11 14:19:04-07 - - $
 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 1.12 2019-04-03 17:17:43-07 - - $
 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 1.1 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:
```

```
 1: // $Id: string_set.cpp,v 1.4 2019-04-03 17:17:43-07 - - $
 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: /* 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:     TOK_VOID = 258,
   49:     TOK_INT = 259,
   50:     TOK_STRING = 260,
   51:     TOK_IF = 261,
   52:     TOK_ELSE = 262,
   53:     TOK_WHILE = 263,
   54:     TOK_RETURN = 264,
   55:     TOK_STRUCT = 265,
   56:     TOK_NULL = 266,
```

```
57:     TOK_NEW = 267,
58:     TOK_ARRAY = 268,
59:     TOK_ARROW = 269,
60:     TOK_EQ = 270,
61:     TOK_NE = 271,
62:     TOK_LT = 272,
63:     TOK_LE = 273,
64:     TOK_GT = 274,
65:     TOK_GE = 275,
66:     TOK_NOT = 276,
67:     TOK_IDENT = 277,
68:     TOK_INTCON = 278,
69:     TOK_CHARCON = 279,
70:     TOK_STRINGCON = 280,
71:     TOK_ROOT = 281,
72:     TOK_BLOCK = 282,
73:     TOK_CALL = 283,
74:     TOK_IFELSE = 284,
75:     TOK_INITDECL = 285,
76:     TOK_POS = 286,
77:     TOK_NEG = 287,
78:     TOK_NEWARRAY = 288,
79:     TOK_TYPEID = 289,
80:     TOK_FIELD = 290
81:   };
82: #endif
83:
84: /* Value type.  */
85: #if ! defined YYSTYPE && ! defined YYSTYPE_IS_DECLARED
86: typedef int YYSTYPE;
87: # define YYSTYPE_IS_TRIVIAL 1
88: # define YYSTYPE_IS_DECLARED 1
89: #endif
90:
91:
92: extern YYSTYPE yylval;
93:
94: int yyparse (void);
95:
96: #endif /* !YY_YY_YYPARSE_H_INCLUDED  */
```

```
%{
// $Id: parser.y,v 1.18 2018-10-25 13:08:35-07 - - $
// Dummy parser for scanner project.

#include <cassert>

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

%}

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

%token TOK_VOID TOK_INT TOK_STRING
%token TOK_IF TOK_ELSE TOK_WHILE TOK_RETURN TOK_STRUCT
%token TOK_NULL TOK_NEW TOK_ARRAY TOK_ARROW
%token TOK_EQ TOK_NE TOK_LT TOK_LE TOK_GT TOK_GE TOK_NOT
%token TOK_IDENT TOK_INTCON TOK_CHARCON TOK_STRINGCON

%token TOK_ROOT TOK_BLOCK TOK_CALL TOK_IFELSE TOK_INITDECL
%token TOK_POS TOK_NEG TOK_NEWARRAY TOK_TYPEID TOK_FIELD

%start program

%%

program : program token | ;
token   : '(' | ')' | '[' | ']' | '{' | '}' | ';' | ','
        | '=' | '+' | '-' | '*' | '/' | '%' | TOK_NOT
        | TOK_ROOT TOK_VOID | TOK_INT | TOK_STRING
        | TOK_IF | TOK_ELSE | TOK_WHILE | TOK_RETURN | TOK_STRUCT
        | TOK_NULL | TOK_NEW | TOK_ARRAY | TOK_ARROW
        | TOK_EQ | TOK_NE | TOK_LT | TOK_LE | TOK_GT | TOK_GE
        | TOK_IDENT | TOK_INTCON | TOK_CHARCON | TOK_STRINGCON
        ;

%%
```

```
42:
43:
44: const char *parser::get_tname (int symbol) {
45:    return yytname [YYTRANSLATE (symbol)];
46: }
47:
48:
49: bool is_defined_token (int symbol) {
50:    return YYTRANSLATE (symbol) > YYUNDEFTOK;
51: }
52:
```