

```
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,
21:                    int offset, 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,
27:              astree* yyvaluep);
28: void free_ast (astree* tree);
29: void free_ast2 (astree* tree1, astree* tree2);
30:
31: RCSH("$Id: astree.h,v 1.2 2013-10-11 18:52:46-07 - - $")
32: #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,
14:                    int offset, 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: astree* adopt1 (astree* root, astree* child) {
28:     root->children.push_back (child);
29:     DEBUGF ('a', "%p (%s) adopting %p (%s)\n",
30:             root, root->lexinfo->c_str(),
31:             child, child->lexinfo->c_str());
32:     return root;
33: }
34:
35: astree* adopt2 (astree* root, astree* left, astree* right) {
36:     adopt1 (root, left);
37:     adopt1 (root, right);
38:     return root;
39: }
40:
41: astree* adopt1sym (astree* root, astree* child, int symbol) {
42:     root = adopt1 (root, child);
43:     root->symbol = symbol;
44:     return root;
45: }
46:
```

```
47:
48: static void dump_node (FILE* outfile, astree* node) {
49:     fprintf (outfile, "%p->{%s(%d) %ld:%ld.%03ld \"%s\" [",
50:             node, get_yytname (node->symbol), node->symbol,
51:             node->filenr, node->linenr, node->offset,
52:             node->lexinfo->c_str());
53:     bool need_space = false;
54:     for (size_t child = 0; child < node->children.size();
55:         ++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,
64:                             int depth) {
65:     if (root == NULL) return;
66:     fprintf (outfile, "%*s%s ", depth * 3, "",
67:             root->lexinfo->c_str());
68:     dump_node (outfile, root);
69:     fprintf (outfile, "\n");
70:     for (size_t child = 0; child < root->children.size();
71:         ++child) {
72:         dump_astree_rec (outfile, root->children[child],
73:                         depth + 1);
74:     }
75: }
76:
77: void dump_astree (FILE* outfile, astree* root) {
78:     dump_astree_rec (outfile, root, 0);
79:     fflush (NULL);
80: }
81:
82: void yyprint (FILE* outfile, unsigned short toknum,
83:              astree* yyvaluep) {
84:     if (is_defined_token (toknum)) {
85:         dump_node (outfile, yyvaluep);
86:     } else {
87:         fprintf (outfile, "%s(%d)\n",
88:                 get_yytname (toknum), toknum);
89:     }
90:     fflush (NULL);
91: }
92:
```

```
93:
94: void free_ast (astree* root) {
95:     while (not root->children.empty()) {
96:         astree* child = root->children.back();
97:         root->children.pop_back();
98:         free_ast (child);
99:     }
100:     DEBUGF ('f', "free [%p]-> %d:%d.%d: %s: \"%s\"")\n",
101:             root, root->filenr, root->linenr, root->offset,
102:             get_yytname (root->symbol), root->lexinfo->c_str());
103:     delete root;
104: }
105:
106: void free_ast2 (astree* tree1, astree* tree2) {
107:     free_ast (tree1);
108:     free_ast (tree2);
109: }
110:
111: RCSC("$Id: astree.cpp,v 1.1 2014-10-03 18:22:05-07 - - $")
112:
```

```
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 set 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. Remembers only the largest value.
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
63:     // EXIT_FAILURE.
64:     //
65:
66: void syserrprintf (const char* object);
67:     //
68:     // Print a message resulting from a bad system call. The
69:     // object is the name of the object causing the problem and
70:     // the reason is taken from the external variable errno.
71:     // Sets the exit status to EXIT_FAILURE.
72:     //
73:
```

```
74:
75: //
76: // Support for stub messages.
77: //
78: #define STUBPRINTF(...) \
79:     __stubprintf (__FILE__, __LINE__, __func__, __VA_ARGS__)
80: void __stubprintf (const char* file, int line, const char* func,
81:                  const char* format, ...);
82:
83: //
84: // Debugging utility.
85: //
86:
87: void set_debugflags (const char* flags);
88:
89:     // Sets a string of debug flags to be used by DEBUGF
90:     // statements. Uses the address of the string, and does
91:     // not copy it, so it must not be dangling. If a particular
92:     // debug flag has been set, messages are printed. The format
93:     // is identical to printf format. The flag "@" turns on
94:     // all flags.
95:
96:
97: bool is_debugflag (char flag);
98: //
99: // Checks to see if a debugflag is set.
100: //
101:
102: #ifndef NDEBUG
103: // Do not generate any code.
104: #define DEBUGF(FLAG,...) /**/
105: #define DEBUGSTMT(FLAG,STMTS) /**/
106: #else
107: // Generate debugging code.
108: void __debugprintf (char flag, const char* file, int line,
109:                   const char* func, const char* format, ...);
110: #define DEBUGF(FLAG,...) \
111:     __debugprintf (FLAG, __FILE__, __LINE__, __func__, \
112:                  __VA_ARGS__)
113: #define DEBUGSTMT(FLAG,STMTS) \
114:     if (is_debugflag (FLAG)) { DEBUGF (FLAG, "\n"); STMTS }
115: #endif
116:
117: //
118: // Definition of RCSID macro to include RCS info in obj and exec.
119: //
120:
121: #define RCS3(ID,N,X) static const char ID##N[] = X;
122: #define RCS2(N,X) RCS3(RCS_Id,N,X)
123: #define RCSH(X) RCS2(__COUNTER__,X)
124: #define RCSC(X) RCSH(X \
125: "\0$Compiled: " __FILE__ " " __DATE__ " " __TIME__ " $")
126: RCSH("$Id: auxlib.h,v 1.2 2013-10-11 18:53:00-07 - - $")
127: #endif
```

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

```
106:
107: void set_debugflags (const char* flags) {
108:     debugflags = flags;
109:     if (strchr (debugflags, '@') != NULL) alldebugflags = true;
110:     DEBUGF ('x', "Debugflags = \"%s\\", all = %d\\n",
111:             debugflags, alldebugflags);
112: }
113:
114: bool is_debugflag (char flag) {
115:     return alldebugflags or strchr (debugflags, flag) != NULL;
116: }
117:
118: void __debugprintf (char flag, const char* file, int line,
119:                    const char* func, const char* format, ...) {
120:     va_list args;
121:     if (not is_debugflag (flag)) return;
122:     fflush (NULL);
123:     va_start (args, format);
124:     fprintf (stderr, "DEBUGF(%c): %s[%d] %s():\\n",
125:             flag, file, line, func);
126:     vfprintf (stderr, format, args);
127:     va_end (args);
128:     fflush (NULL);
129: }
130:
131: RCSC("$Id: auxlib.cpp,v 1.1 2014-10-03 18:22:05-07 - - $")
132:
```

```
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_lineno;
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 linenr);
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:
39: void scanner_include (void);
40:
41: typedef astree* astree_pointer;
42: #define YYSTYPE astree_pointer
43: #include "yyparse.h"
44:
45: RCSH("$Id: lyutils.h,v 1.2 2013-10-11 18:53:00-07 - - $")
46: #endif
```

```
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:     fprintf ("%s: %d: %s\n",
50:             included_filenames.back().c_str(),
51:             scan_linenr, message);
52: }
53:
54: void scanner_badchar (unsigned char bad) {
55:     char char_rep[16];
56:     sprintf (char_rep, isgraph (bad) ? "%c" : "\\%03o", bad);
57:     fprintf ("%s: %d: invalid source character (%s)\n",
58:             included_filenames.back().c_str(),
59:             scan_linenr, char_rep);
60: }
61:
62: void scanner_badtoken (char* lexeme) {
63:     fprintf ("%s: %d: invalid token (%s)\n",
64:             included_filenames.back().c_str(),
65:             scan_linenr, lexeme);
66: }
67:
68: int yylval_token (int symbol) {
69:     int offset = scan_offset - yyleng;
70:     yylval = new_astree (symbol, included_filenames.size() - 1,
71:                         scan_linenr, offset, yytext);
72:     return symbol;
73: }
74:
75: astree* new_parserroot (void) {
76:     yyparse_astree = new_astree (ROOT, 0, 0, 0, "<<ROOT>>");
77:     return yyparse_astree;
78: }
79:
```

```
80:
81: void scanner_include (void) {
82:     scanner_newline();
83:     char filename[strlen (yytext) + 1];
84:     int linenr;
85:     int scan_rc = sscanf (yytext, "# %d \"%^[^\" ]\"",
86:                           &linenr, filename);
87:     if (scan_rc != 2) {
88:         errprintf ("%d: [%s]: invalid directive, ignored\n",
89:                   scan_rc, yytext);
90:     }else {
91:         printf (";# %d \"%s\"\n", linenr, filename);
92:         scanner_newfilename (filename);
93:         scan_linenr = linenr - 1;
94:         DEBUGF ('m', "filename=%s, scan_linenr=%d\n",
95:                 included_filenames.back().c_str(), scan_linenr);
96:     }
97: }
98:
99: RCSC("$Id: lyutils.cpp,v 1.1 2014-10-03 18:22:05-07 - - $")
100:
```

```
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 1.1 2013-10-11 18:53:00-07 - - $")
17: #endif
```

```
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();
22:         ++bucket) {
23:         bool need_index = true;
24:         size_t curr_size = set.bucket_size (bucket);
25:         if (max_bucket_size < curr_size)
26:             max_bucket_size = curr_size;
27:         for (stringset_bucket_citor itor = set.cbegin (bucket);
28:             itor != set.cend (bucket); ++itor) {
29:             if (need_index) fprintf (out, "string[%4lu]: ", bucket);
30:             else fprintf (out, "      %4s    ", "");
31:             need_index = false;
32:             const string* str = &*itor;
33:             fprintf (out, "%22lu %p->\"%s\"\\n",
34:                     set.hash_function() (*str), str, str->c_str());
35:         }
36:     }
37:     fprintf (out, "load_factor = %.3f\\n", set.load_factor());
38:     fprintf (out, "bucket_count = %lu\\n", set.bucket_count());
39:     fprintf (out, "max_bucket_size = %lu\\n", max_bucket_size);
40: }
41:
42: RCSC("$Id: stringset.cpp,v 1.1 2014-10-03 18:22:05-07 - - $")
```



```
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, 200
4, 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 publis
hed 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 Lice
nse
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 cont
ains
23:     part or all of the Bison parser skeleton and distribute that w
ork
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 ther
eof
26:     as a parser skeleton.  Alternatively, if you modify or redistr
ibute
27:     the parser skeleton itself, you may (at your option) remove th
is
28:     special exception, which will cause the skeleton and the resul
ting
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 Foundati
on in
33:     version 2.2 of Bison.  */
34:
35:
36: /* Tokens.  */
37: #ifndef YYTOKENTYPE
38: # define YYTOKENTYPE
39:     /* Put the tokens into the symbol table, so that GDB and other
```

debuggers

```
40:      know about them.  */
41:      enum yytokentype {
42:          TOK_VOID = 258,
43:          TOK_BOOL = 259,
44:          TOK_CHAR = 260,
45:          TOK_INT = 261,
46:          TOK_STRING = 262,
47:          TOK_IF = 263,
48:          TOK_ELSE = 264,
49:          TOK_WHILE = 265,
50:          TOK_RETURN = 266,
51:          TOK_STRUCT = 267,
52:          TOK_FALSE = 268,
53:          TOK_TRUE = 269,
54:          TOK_NULL = 270,
55:          TOK_NEW = 271,
56:          TOK_ARRAY = 272,
57:          TOK_EQ = 273,
58:          TOK_NE = 274,
59:          TOK_LT = 275,
60:          TOK_LE = 276,
61:          TOK_GT = 277,
62:          TOK_GE = 278,
63:          TOK_IDENT = 279,
64:          TOK_INTCON = 280,
65:          TOK_CHARCON = 281,
66:          TOK_STRINGCON = 282,
67:          TOK_BLOCK = 283,
68:          TOK_CALL = 284,
69:          TOK_IFELSE = 285,
70:          TOK_INITDECL = 286,
71:          TOK_POS = 287,
72:          TOK_NEG = 288,
73:          TOK_NEWARRAY = 289,
74:          TOK_TYPEID = 290,
75:          TOK_FIELD = 291,
76:          TOK_ORD = 292,
77:          TOK_CHR = 293,
78:          TOK_ROOT = 294
79:      };
80: #endif
81:
82:
83:
84: #if ! defined YYSTYPE && ! defined YYSTYPE_IS_DECLARED
85: typedef int YYSTYPE;
86: # define YYSTYPE_IS_TRIVIAL 1
87: # define YYSTYPE YYSTYPE /* obsolescent; will be withdrawn */
88: # define YYSTYPE_IS_DECLARED 1
89: #endif
90:
91: extern YYSTYPE yylval;
92:
```

93:

```
1: %{
2: // Dummy parser for scanner project.
3:
4: #include "lyutils.h"
5: #include "astree.h"
6:
7: %}
8:
9: %debug
10: %defines
11: %error-verbose
12: %token-table
13: %verbose
14:
15: %token TOK_VOID TOK_BOOL TOK_CHAR TOK_INT TOK_STRING
16: %token TOK_IF TOK_ELSE TOK_WHILE TOK_RETURN TOK_STRUCT
17: %token TOK_FALSE TOK_TRUE TOK_NULL TOK_NEW TOK_ARRAY
18: %token TOK_EQ TOK_NE TOK_LT TOK_LE TOK_GT TOK_GE
19: %token TOK_IDENT TOK_INTCON TOK_CHARCON TOK_STRINGCON
20:
21: %token TOK_BLOCK TOK_CALL TOK_IFELSE TOK_INITDECL
22: %token TOK_POS TOK_NEG TOK_NEWARRAY TOK_TYPEID TOK_FIELD
23: %token TOK_ORD TOK_CHR TOK_ROOT
24:
25: %start program
26:
27: %%
28:
29: program : program token | ;
30: token   : '(' | ')' | '[' | ']' | '{' | '}' | ';' | ',' | '.'
31:         | '=' | '+' | '-' | '*' | '/' | '%' | '!'
32:         | TOK_VOID | TOK_BOOL | TOK_CHAR | TOK_INT | TOK_STRING
33:         | TOK_IF | TOK_ELSE | TOK_WHILE | TOK_RETURN | TOK_STRUCT
34:         | TOK_FALSE | TOK_TRUE | TOK_NULL | TOK_NEW | TOK_ARRAY
35:         | TOK_EQ | TOK_NE | TOK_LT | TOK_LE | TOK_GT | TOK_GE
36:         | TOK_IDENT | TOK_INTCON | TOK_CHARCON | TOK_STRINGCON
37:         | TOK_ORD | TOK_CHR | TOK_ROOT
38:         ;
39:
40: %%
41:
42: const char *get_yytname (int symbol) {
43:     return yytname [YYTRANSLATE (symbol)];
44: }
45:
46:
47: bool is_defined_token (int symbol) {
48:     return YYTRANSLATE (symbol) > YYUNDEFTOK;
49: }
50:
51: static void* yycalloc (size_t size) {
52:     void* result = calloc (1, size);
53:     assert (result != NULL);
54:     return result;
```

10/16/13
11:22:22

\$cmps104a-wm/Assignments/code/utility-code/
parser.y

2/2

55: }
56: