

```
1: #ifndef __AUXLIB_H__
2: #define __AUXLIB_H__
3:
4: #include <stdarg.h>
5:
6: //
7: // DESCRIPTION
8: //     Auxiliary library containing miscellaneous useful things.
9: //
10:
11: //
12: // Error message and exit status utility.
13: //
14:
15: void set_execname (char* argv0);
16:     //
17:     // Sets the program name for use by auxlib messages.
18:     // Must called from main before anything else is done,
19:     // passing in argv[0].
20:     //
21:
22: const char* get_execname (void);
23:     //
24:     // Returns a read-only value previously stored by set_progname.
25:     //
26:
27: void eprint_status (const char* command, int status);
28:     //
29:     // Print the status returned by wait(2) from a subprocess.
30:     //
31:
32: int get_exitstatus (void);
33:     //
34:     // Returns the exit status. Default is EXIT_SUCCESS unless
35:     // set_exitstatus (int) is called. The last statement in main
36:     // should be: ``return get_exitstatus();''.
37:     //
38:
39: void set_exitstatus (int);
40:     //
41:     // Sets the exit status. Remembers only the largest value passed in.
42:     //
43:
```

```
44:
45: void veprintf (const char* format, va_list args);
46:     //
47:     // Prints a message to stderr using the vector form of
48:     // argument list.
49:     //
50:
51: void eprintf (const char* format, ...);
52:     //
53:     // Print a message to stderr according to the printf format
54:     // specified. Usually called for debug output.
55:     // Precedes the message by the program name if the format
56:     // begins with the characters `%:'.
57:     //
58:
59: void errprintf (const char* format, ...);
60:     //
61:     // Print an error message according to the printf format
62:     // specified, using eprintf. Sets the exitstatus to EXIT_FAILURE.
63:     //
64:
65: void syserrprintf (const char* object);
66:     //
67:     // Print a message resulting from a bad system call. The
68:     // object is the name of the object causing the problem and
69:     // the reason is taken from the external variable errno.
70:     // Sets the exit status to EXIT_FAILURE.
71:     //
72:
```

```
73:
74: //
75: // Support for stub messages.
76: //
77: #define STUBPRINTF(...) \
78:     __stubprintf (__FILE__, __LINE__, __func__, __VA_ARGS__)
79: void __stubprintf (const char* file, int line, const char* func,
80:                  const char* format, ...);
81:
82: //
83: // Debugging utility.
84: //
85:
86: void set_debugflags (const char* flags);
87:     //
88:     // Sets a string of debug flags to be used by DEBUGF statements.
89:     // Uses the address of the string, and does not copy it, so it
90:     // must not be dangling. If a particular debug flag has been set,
91:     // messages are printed. The format is identical to printf format.
92:     // The flag "@" turns on all flags.
93:     //
94:
95: bool is_debugflag (char flag);
96:     //
97:     // Checks to see if a debugflag is set.
98:     //
99:
100: #ifndef NDEBUG
101: // Do not generate any code.
102: #define DEBUGF(FLAG,...) /**/
103: #define DEBUGSTMT(FLAG,STMTS) /**/
104: #else
105: // Generate debugging code.
106: void __debugprintf (char flag, const char* file, int line,
107:                   const char* func, const char* format, ...);
108: #define DEBUGF(FLAG,...) \
109:     __debugprintf (FLAG, __FILE__, __LINE__, __func__, __VA_ARGS__)
110: #define DEBUGSTMT(FLAG,STMTS) \
111:     if (is_debugflag (FLAG)) { DEBUGF (FLAG, "\n"); STMTS }
112: #endif
113:
114: //
115: // Definition of RCSID macro to include RCS info in objs and execbin.
116: //
117:
118: #define RCS3(ID,N,X) static const char ID##N[] = X;
119: #define RCS2(N,X) RCS3(RCS_Id,N,X)
120: #define RCSH(X) RCS2(__COUNTER__,X)
121: #define RCSC(X) RCSH(X \
122: "\0$Compiled: " __FILE__ " " __DATE__ " " __TIME__ " $")
123: RCSH("$Id: auxlib.h,v 1.1 2013-09-20 19:38:26-07 - - $")
124: #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:
56: void veprintf (const char* format, va_list args) {
57:     assert (execname != NULL);
58:     assert (format != NULL);
59:     fflush (NULL);
60:     if (strstr (format, "%:") == format) {
61:         fprintf (stderr, "%s: ", get_execname ());
62:         format += 2;
63:     }
64:     vfprintf (stderr, format, args);
65:     fflush (NULL);
66: }
67:
68: void eprintf (const char* format, ...) {
69:     va_list args;
70:     va_start (args, format);
71:     veprintf (format, args);
72:     va_end (args);
73: }
74:
75: void errprintf (const char* format, ...) {
76:     va_list args;
77:     va_start (args, format);
78:     veprintf (format, args);
79:     va_end (args);
80:     exitstatus = EXIT_FAILURE;
81: }
82:
83: void syserrprintf (const char* object) {
84:     errprintf ("%s: %s\n", object, strerror (errno));
85: }
86:
87: int get_exitstatus (void) {
88:     return exitstatus;
89: }
90:
91: void set_exitstatus (int newexitstatus) {
92:     if (exitstatus < newexitstatus) exitstatus = newexitstatus;
93:     DEBUGF ('x', "exitstatus = %d\n", exitstatus);
94: }
95:
96: void __stubprintf (const char* file, int line, const char* func,
97:                   const char* format, ...) {
98:     va_list args;
99:     fflush (NULL);
100:    printf ("%s: %s[%d] %s: ", execname, file, line, func);
101:    va_start (args, format);
102:    vprintf (format, args);
103:    va_end (args);
104:    fflush (NULL);
105: }
106:
```

```
107:
108: void set_debugflags (const char* flags) {
109:     debugflags = flags;
110:     if (strchr (debugflags, '@') != NULL) alldebugflags = true;
111:     DEBUGF ('x', "Debugflags = \"%s\"", all = %d\n",
112:             debugflags, alldebugflags);
113: }
114:
115: bool is_debugflag (char flag) {
116:     return alldebugflags or strchr (debugflags, flag) != NULL;
117: }
118:
119: void __debugprintf (char flag, const char* file, int line,
120:                    const char* func, const char* format, ...) {
121:     va_list args;
122:     if (not is_debugflag (flag)) return;
123:     fflush (NULL);
124:     va_start (args, format);
125:     fprintf (stderr, "DEBUGF(%c): %s[%d] %s():\n",
126:             flag, file, line, func);
127:     vfprintf (stderr, format, args);
128:     va_end (args);
129:     fflush (NULL);
130: }
131:
132: RCSC("$Id: auxlib.cpp,v 1.2 2014-10-07 18:07:29-07 - - $")
133:
```

```
1: // $Id: cppstrtok.cpp,v 1.3 2014-10-07 18:09:11-07 - - $
2:
3: // Use cpp to scan a file and print line numbers.
4: // Print out each input line read in, then strtok it for
5: // tokens.
6:
7: #include <string>
8: using namespace std;
9:
10: #include <errno.h>
11: #include <libgen.h>
12: #include <stdio.h>
13: #include <stdlib.h>
14: #include <string.h>
15: #include <wait.h>
16:
17: #include "auxlib.h"
18:
19: const string CPP = "/usr/bin/cpp";
20: const size_t LINESIZE = 1024;
21:
22: // Chomp the last character from a buffer if it is delim.
23: void chomp (char* string, char delim) {
24:     size_t len = strlen (string);
25:     if (len == 0) return;
26:     char* nlpos = string + len - 1;
27:     if (*nlpos == delim) *nlpos = '\\0';
28: }
29:
```

```
30:
31: // Run cpp against the lines of the file.
32: void cpplines (FILE* pipe, char* filename) {
33:     int linenr = 1;
34:     char inputname[LINESIZE];
35:     strcpy (inputname, filename);
36:     for (;;) {
37:         char buffer[LINESIZE];
38:         char* fgets_rc = fgets (buffer, LINESIZE, pipe);
39:         if (fgets_rc == NULL) break;
40:         chomp (buffer, '\n');
41:         printf ("%s:line %d: [%s]\n", filename, linenr, buffer);
42:         // http://gcc.gnu.org/onlinedocs/cpp/Preprocessor-Output.html
43:         int sscanf_rc = sscanf (buffer, "# %d \"%^[^\"]\"",
44:                                &linenr, filename);
45:         if (sscanf_rc == 2) {
46:             printf ("DIRECTIVE: line %d file \"%s\"\n", linenr, filename);
47:             continue;
48:         }
49:         char* savepos = NULL;
50:         char* bufptr = buffer;
51:         for (int tokenct = 1; ++tokenct) {
52:             char* token = strtok_r (bufptr, " \\t\\n", &savepos);
53:             bufptr = NULL;
54:             if (token == NULL) break;
55:             printf ("token %d.%d: [%s]\n",
56:                    linenr, tokenct, token);
57:         }
58:         ++linenr;
59:     }
60: }
61:
62: int main (int argc, char** argv) {
63:     set_execname (argv[0]);
64:     for (int argi = 1; argi < argc; ++argi) {
65:         char* filename = argv[argi];
66:         string command = CPP + " " + filename;
67:         printf ("command=\"%s\"\n", command.c_str());
68:         FILE* pipe = popen (command.c_str(), "r");
69:         if (pipe == NULL) {
70:             syserrprintf (command.c_str());
71:         } else {
72:             cpplines (pipe, filename);
73:             int pclose_rc = pclose (pipe);
74:             eprint_status (command.c_str(), pclose_rc);
75:         }
76:     }
77:     return get_exitstatus();
78: }
79:
```



```
1: # $Id: Makefile,v 1.9 2015-03-25 19:01:26-07 - - $
2:
3: GCC          = g++ -g -O0 -Wall -Wextra -std=gnu++11
4: MKDEP        = g++ -MM -std=gnu++11
5: VALGRIND     = valgrind --leak-check=full --show-reachable=yes
6:
7: MKFILE       = Makefile
8: DEPFILE      = Makefile.dep
9: SOURCES      = auxlib.cpp cppstrtok.cpp
10: HEADERS      = auxlib.h
11: OBJECTS      = ${SOURCES:.cpp=.o}
12: EXECBIN      = cppstrtok
13: SRCFILES     = ${HEADERS} ${SOURCES} ${MKFILE}
14: SMALLFILES   = ${DEPFILE} foo.oc foo1.oh foo2.oh
15: CHECKINS     = ${SRCFILES} ${SMALLFILES}
16: LISTING      = Listing.ps
17:
18: all : ${EXECBIN}
19:
20: ${EXECBIN} : ${OBJECTS}
21:             ${GCC} -o${EXECBIN} ${OBJECTS}
22:
23: %.o : %.cpp
24:             ${GCC} -c $<
25:
26: ci :
27:         cid + ${CHECKINS}
28:         checksource ${CHECKINS}
29:
30: clean :
31:         - rm ${OBJECTS}
32:
33: spotless : clean
34:         - rm ${EXECBIN} ${LISTING} ${LISTING:.ps=.pdf} ${DEPFILE} \
35:           test.out test.err misc.lis
36:
37: ${DEPFILE} :
38:             ${MKDEP} ${SOURCES} >${DEPFILE}
39:
40: dep :
41:         - rm ${DEPFILE}
42:         ${MAKE} --no-print-directory ${DEPFILE}
43:
44: include Makefile.dep
45:
46: test : ${EXECBIN}
47:         ${VALGRIND} ${EXECBIN} foo.oc 1>test.out 2>test.err
48:
49: misc.lis : ${DEPFILE} foo.oc foo1.oh foo2.oh
50:         morecat ${DEPFILE} foo.oc foo1.oh foo2.oh >misc.lis
51:
52: lis : misc.lis test
53:         mkpspdf ${LISTING} ${SRCFILES} misc.lis test.out test.err
54:
```

```
1: ::::::::::::::::::::::::::::::
2: Makefile.dep
3: ::::::::::::::::::::::::::::::
4:     1  auxlib.o: auxlib.cpp auxlib.h
5:     2  cppstrtok.o: cppstrtok.cpp auxlib.h
6: ::::::::::::::::::::::::::::::
7: foo.oc
8: ::::::::::::::::::::::::::::::
9:     1  line 1// $Id: foo.oc,v 1.3 2013-09-19 18:03:21-07 - - $
10:    2  __FILE__ __LINE__ __DATE__ __TIME__
11:    3  foo.oc, line 3.
12:    4  #include "foo1.oh"
13:    5  foo.oc, line 5.
14:    6  #include "foo2.oh"
15:    7  /* Comment */ on line 7
16:    8  FOO1 + FOO2;
17:    9  foo.oc, line 9, last line.
18: ::::::::::::::::::::::::::::::
19: foo1.oh
20: ::::::::::::::::::::::::::::::
21:    1  // $Id: foo1.oh,v 1.2 2011-09-29 19:06:34-07 - - $
22:    2  __FILE__ __LINE__ __DATE__ __TIME__
23:    3  foo1.h, line 3.
24:    4  foo1.h, line 4.
25:    5  // Comment.
26:    6  foo1.h, line 6. /* Comment */ last line
27:    7  #define FOO1 "foo1"
28: ::::::::::::::::::::::::::::::
29: foo2.oh
30: ::::::::::::::::::::::::::::::
31:    1  // $Id: foo2.oh,v 1.2 2011-09-29 19:06:34-07 - - $
32:    2  __FILE__ __LINE__ __DATE__ __TIME__
33:    3  foo2.h, line 3.
34:    4  foo2.h, line 4.
35:    5  // Comment.
36:    6  foo2.h, line 6. /* Comment */ last line
37:    7  #define FOO2 "foo2"
```

```
1: command="/usr/bin/cpp foo.oc"
2: foo.oc:line 1: [# 1 "foo.oc"]
3: DIRECTIVE: line 1 file "foo.oc"
4: foo.oc:line 1: [# 1 "<built-in>"]
5: DIRECTIVE: line 1 file "<built-in>"
6: <built-in>:line 1: [# 1 "<command-line>"]
7: DIRECTIVE: line 1 file "<command-line>"
8: <command-line>:line 1: [# 1 "foo.oc"]
9: DIRECTIVE: line 1 file "foo.oc"
10: foo.oc:line 1: [line 1]
11: token 1.1: [line]
12: token 1.2: [1]
13: foo.oc:line 2: ["foo.oc" 2 "Mar 25 2015" "19:02:41"]
14: token 2.1: ["foo.oc"]
15: token 2.2: [2]
16: token 2.3: ["Mar"]
17: token 2.4: [25]
18: token 2.5: [2015]
19: token 2.6: ["19:02:41"]
20: foo.oc:line 3: [foo.oc, line 3.]
21: token 3.1: [foo.oc,]
22: token 3.2: [line]
23: token 3.3: [3.]
24: foo.oc:line 4: [# 1 "foo1.oh" 1]
25: DIRECTIVE: line 1 file "foo1.oh"
26: foo1.oh:line 1: []
27: foo1.oh:line 2: ["foo1.oh" 2 "Mar 25 2015" "19:02:41"]
28: token 2.1: ["foo1.oh"]
29: token 2.2: [2]
30: token 2.3: ["Mar"]
31: token 2.4: [25]
32: token 2.5: [2015]
33: token 2.6: ["19:02:41"]
34: foo1.oh:line 3: [foo1.h, line 3.]
35: token 3.1: [foo1.h,]
36: token 3.2: [line]
37: token 3.3: [3.]
38: foo1.oh:line 4: [foo1.h, line 4.]
39: token 4.1: [foo1.h,]
40: token 4.2: [line]
41: token 4.3: [4.]
42: foo1.oh:line 5: []
43: foo1.oh:line 6: [foo1.h, line 6. last line]
44: token 6.1: [foo1.h,]
45: token 6.2: [line]
46: token 6.3: [6.]
47: token 6.4: [last]
48: token 6.5: [line]
49: foo1.oh:line 7: [# 5 "foo.oc" 2]
50: DIRECTIVE: line 5 file "foo.oc"
51: foo.oc:line 5: [foo.oc, line 5.]
52: token 5.1: [foo.oc,]
53: token 5.2: [line]
54: token 5.3: [5.]
55: foo.oc:line 6: [# 1 "foo2.oh" 1]
56: DIRECTIVE: line 1 file "foo2.oh"
57: foo2.oh:line 1: []
58: foo2.oh:line 2: ["foo2.oh" 2 "Mar 25 2015" "19:02:41"]
```

```
59: token 2.1: ["foo2.oh"]
60: token 2.2: [2]
61: token 2.3: ["Mar]
62: token 2.4: [25]
63: token 2.5: [2015"]
64: token 2.6: ["19:02:41"]
65: foo2.oh:line 3: [foo2.h, line 3.]
66: token 3.1: [foo2.h,]
67: token 3.2: [line]
68: token 3.3: [3.]
69: foo2.oh:line 4: [foo2.h, line 4.]
70: token 4.1: [foo2.h,]
71: token 4.2: [line]
72: token 4.3: [4.]
73: foo2.oh:line 5: []
74: foo2.oh:line 6: [foo2.h, line 6. last line]
75: token 6.1: [foo2.h,]
76: token 6.2: [line]
77: token 6.3: [6.]
78: token 6.4: [last]
79: token 6.5: [line]
80: foo2.oh:line 7: [# 7 "foo.oc" 2]
81: DIRECTIVE: line 7 file "foo.oc"
82: foo.oc:line 7: [                on line 7]
83: token 7.1: [on]
84: token 7.2: [line]
85: token 7.3: [7]
86: foo.oc:line 8: ["foo1" + "foo2";]
87: token 8.1: ["foo1"]
88: token 8.2: [+]
89: token 8.3: ["foo2";]
90: foo.oc:line 9: [foo.oc, line 9, last line.]
91: token 9.1: [foo.oc,]
92: token 9.2: [line]
93: token 9.3: [9,]
94: token 9.4: [last]
95: token 9.5: [line.]
```

```
1: ==8732== Memcheck, a memory error detector
2: ==8732== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
3: ==8732== Using Valgrind-3.9.0 and LibVEX; rerun with -h for copyright in
fo
4: ==8732== Command: cppstrtok foo.oc
5: ==8732==
6: ==8732==
7: ==8732== HEAP SUMMARY:
8: ==8732==       in use at exit: 0 bytes in 0 blocks
9: ==8732==   total heap usage: 3 allocs, 3 frees, 342 bytes allocated
10: ==8732==
11: ==8732== All heap blocks were freed -- no leaks are possible
12: ==8732==
13: ==8732== For counts of detected and suppressed errors, rerun with: -v
14: ==8732== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 6 from 6)
```