```
#ifndef __AUXLIB_H__
#define __AUXLIB_H__

#include <stdarg.h>

//
// DESCRIPTION
//    Auxiliary library containing miscellaneous useful things.
//

//
// Error message and exit status utility.
//

void set_execname (char* argv0);
   //
   // Sets the program name for use by auxlib messages.
   // Must called from main before anything else is done,
   // passing in argv[0].
   //

const char* get_execname (void);
   //
   // Returns a read-only value previously stored by set_progname.
   //

void eprint_status (const char* command, int status);
   //
   // Print the status returned by wait(2) from a subprocess.
   //

int get_exitstatus (void);
   //
   // Returns the exit status.  Default is EXIT_SUCCESS unless
   // set_exitstatus (int) is called.  The last statement in main
   // should be:  ``return get_exitstatus();''.
   //

void set_exitstatus (int);
   //
   // Sets the exit status.  Remebers only the largest value passed in.
   //

```

```

void veprintf (const char* format, va_list args);
   //
   // Prints a message to stderr using the vector form of
   // argument list.
   //

void eprintf (const char* format, ...);
   //
   // Print a message to stderr according to the printf format
   // specified.  Usually called for debug output.
   // Precedes the message by the program name if the format
   // begins with the characters `%:'.
   //

void errprintf (const char* format, ...);
   //
   // Print an error message according to the printf format
   // specified, using eprintf.  Sets the exitstatus to EXIT_FAILURE.
   //

void syserrprintf (const char* object);
   //
   // Print a message resulting from a bad system call.  The
   // object is the name of the object causing the problem and
   // the reason is taken from the external variable errno.
   // Sets the exit status to EXIT_FAILURE.
   //

```

```

//
// Support for stub messages.
//
#define STUBPRINTF(...) \
        __stubprintf (__FILE__, __LINE__, __func__, __VA_ARGS__)
void __stubprintf (const char* file, int line, const char* func,
                   const char* format, ...);

//
// Debugging utility.
//

void set_debugflags (const char* flags);
   //
   // Sets a string of debug flags to be used by DEBUGF statements.
   // Uses the address of the string, and does not copy it, so it
   // must not be dangling.  If a particular debug flag has been set,
   // messages are printed.  The format is identical to printf format.
   // The flag "@" turns on all flags.
   //

bool is_debugflag (char flag);
   //
   // Checks to see if a debugflag is set.
   //

#ifdef NDEBUG
// Do not generate any code.
#define DEBUGF(FLAG,...)   /**/
#define DEBUGSTMT(FLAG,STMTS) /**/
#else
// Generate debugging code.
void __debugprintf (char flag, const char* file, int line,
                    const char* func, const char* format, ...);
#define DEBUGF(FLAG,...) \
        __debugprintf (FLAG, __FILE__, __LINE__, __func__, __VA_ARGS__)
#define DEBUGSTMT(FLAG,STMTS) \
        if (is_debugflag (FLAG)) { DEBUGF (FLAG, "\n"); STMTS }
#endif

//
// Definition of RCSID macro to include RCS info in objs and execbin.
//

#define RCS3(ID,N,X) static const char ID##N[] = X;
#define RCS2(N,X) RCS3(RCS_Id,N,X)
#define RCSH(X) RCS2(__COUNTER__,X)
#define RCSC(X) RCSH(X \
"\0$Compiled: " __FILE__ " " __DATE__ " " __TIME__ " $")
RCSH("$Id: auxlib.h,v 1.1 2013-09-20 19:38:26-07 - - $")
#endif
```

```

#include <assert.h>
#include <errno.h>
#include <libgen.h>
#include <limits.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <wait.h>

#include "auxlib.h"

static int exitstatus = EXIT_SUCCESS;
static const char* execname = NULL;
static const char* debugflags = "";
static bool alldebugflags = false;

void set_execname (char* argv0) {
   execname = basename (argv0);
}

const char* get_execname (void) {
   assert (execname != NULL);
   return execname;
}

static void eprint_signal (const char* kind, int signal) {
   eprintf (", %s %d", kind, signal);
   const char* sigstr = strsignal (signal);
   if (sigstr != NULL) fprintf (stderr, " %s", sigstr);
}

void eprint_status (const char* command, int status) {
   if (status == 0) return;
   eprintf ("%s: status 0x%04X", command, status);
   if (WIFEXITED (status)) {
      eprintf (", exit %d", WEXITSTATUS (status));
   }
   if (WIFSIGNALED (status)) {
      eprint_signal ("Terminated", WTERMSIG (status));
      #ifdef WCOREDUMP
      if (WCOREDUMP (status)) eprintf (", core dumped");
      #endif
   }
   if (WIFSTOPPED (status)) {
      eprint_signal ("Stopped", WSTOPSIG (status));
   }
   if (WIFCONTINUED (status)) {
      eprintf (", Continued");
   }
   eprintf ("\n");
}

int get_exitstatus (void) {
   return exitstatus;
}

```

```

void veprintf (const char* format, va_list args) {
   assert (execname != NULL);
   assert (format != NULL);
   fflush (NULL);
   if (strstr (format, "%:") == format) {
      fprintf (stderr, "%s: ", get_execname ());
      format += 2;
   }
   vfprintf (stderr, format, args);
   fflush (NULL);
}

void eprintf (const char* format, ...) {
   va_list args;
   va_start (args, format);
   veprintf (format, args);
   va_end (args);
}

void errprintf (const char* format, ...) {
   va_list args;
   va_start (args, format);
   veprintf (format, args);
   va_end (args);
   exitstatus = EXIT_FAILURE;
}

void syserrprintf (const char* object) {
   errprintf ("%:%s: %s\n", object, strerror (errno));
}

void set_exitstatus (int newexitstatus) {
   if (exitstatus < newexitstatus) exitstatus = newexitstatus;
   DEBUGF ('x', "exitstatus = %d\n", exitstatus);
}

void __stubprintf (const char* file, int line, const char* func,
                   const char* format, ...) {
   va_list args;
   fflush (NULL);
   printf ("%s: %s[%d] %s: ", execname, file, line, func);
   va_start (args, format);
   vprintf (format, args);
   va_end (args);
   fflush (NULL);
}

```

```

void set_debugflags (const char* flags) {
   debugflags = flags;
   if (strchr (debugflags, '@') != NULL) alldebugflags = true;
   DEBUGF ('x', "Debugflags = \"%s\", all = %d\n",
            debugflags, alldebugflags);
}

bool is_debugflag (char flag) {
   return alldebugflags or strchr (debugflags, flag) != NULL;
}

void __debugprintf (char flag, const char* file, int line,
                    const char* func, const char* format, ...) {
   va_list args;
   if (not is_debugflag (flag)) return;
   fflush (NULL);
   va_start (args, format);
   fprintf (stderr, "DEBUGF(%c): %s[%d] %s():\n",
             flag, file, line, func);
   vfprintf (stderr, format, args);
   va_end (args);
   fflush (NULL);
}

RCSC("$Id: auxlib.cc,v 1.1 2013-09-20 19:38:26-07 - - $")

```

```
// $Id: cppstrtok.cc,v 1.2 2013-09-20 19:38:26-07 - - $

// Use cpp to scan a file and print line numbers.
// Print out each input line read in, then strtok it for
// tokens.

#include <string>
using namespace std;

#include <errno.h>
#include <libgen.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <wait.h>

#include "auxlib.h"

const string CPP = "/usr/bin/cpp";
const size_t LINESIZE = 1024;

// Chomp the last character from a buffer if it is delim.
void chomp (char *string, char delim) {
   size_t len = strlen (string);
   if (len == 0) return;
   char *nlpos = string + len - 1;
   if (*nlpos == delim) *nlpos = '\0';
}

// Run cpp against the lines of the file.
void cpplines (FILE *pipe, char *filename) {
   int linenr = 1;
   char inputname[LINESIZE];
   strcpy (inputname, filename);
   for (;;) {
      char buffer[LINESIZE];
      char *fgets_rc = fgets (buffer, LINESIZE, pipe);
      if (fgets_rc == NULL) break;
      chomp (buffer, '\n');
      printf ("%s:line %d: [%s]\n", filename, linenr, buffer);
      // http://gcc.gnu.org/onlinedocs/cpp/Preprocessor-Output.html
      int sscanf_rc = sscanf (buffer, "# %d \"%[^\"]\"",
                              &linenr, filename);
      if (sscanf_rc == 2) {
         printf ("DIRECTIVE: line %d file \"%s\"\n", linenr, filename);
         continue;
      }
      char *savepos = NULL;
      char *bufptr = buffer;
      for (int tokenct = 1;; ++tokenct) {
         char *token = strtok_r (bufptr, " \t\n", &savepos);
         bufptr = NULL;
         if (token == NULL) break;
         printf ("token %d.%d: [%s]\n",
                 linenr, tokenct, token);
      }
      ++linenr;
   }
}

int main (int argc, char **argv) {
   set_execname (argv[0]);
   for (int argi = 1; argi < argc; ++argi) {
      char *filename = argv[argi];
      string command = CPP + " " + filename;
      printf ("command=\"%s\"\n", command.c_str());
      FILE *pipe = popen (command.c_str(), "r");
      if (pipe == NULL) {
         syserrprintf (command.c_str());
      }else {
         cpplines (pipe, filename);
         int pclose_rc = pclose (pipe);
         eprint_status (command.c_str(), pclose_rc);
      }
   }
   return get_exitstatus();
}

```

```
# $Id: Makefile,v 1.5 2013-09-25 13:51:12-07 - - $

GCC        = g++ -g -O0 -Wall -Wextra -std=gnu++0x
MKDEPS     = g++ -MM -std=gnu++0x
VALGRIND   = valgrind --leak-check=full --show-reachable=yes

MKFILE     = Makefile
DEPSFILE   = Makefile.deps
SOURCES    = auxlib.cc cppstrtok.cc
HEADERS    = auxlib.h
OBJECTS    = ${SOURCES:.cc=.o}
EXECBIN    = cppstrtok
SRCFILES   = ${HEADERS} ${SOURCES} ${MKFILE}
SMALLFILES = ${DEPSFILE} foo.oc foo1.oh foo2.oh
CHECKINS   = ${SRCFILES} ${SMALLFILES}
LISTING    = Listing.ps

all : ${EXECBIN}

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

%.o : %.cc
        ${GCC} -c $<

ci :
        cid + ${CHECKINS}
        checksource ${CHECKINS}

clean :
        - rm ${OBJECTS}

spotless : clean
        - rm ${EXECBIN} ${LISTING} ${LISTING:.ps=.pdf} test.lis

${DEPSFILE} :
        ${MKDEPS} ${SOURCES} >${DEPSFILE}

deps :
        - rm ${DEPSFILE}
        ${MAKE} --no-print-directory ${DEPSFILE}

include Makefile.deps

test : ${EXECBIN}
        ${VALGRIND} ${EXECBIN} foo.oc 1>test.out 2>test.err
        morecat ${SMALLFILES} test.out test.err >test.lis 2>&1
        rm test.out test.err

lis : test
        mkpspdf ${LISTING} ${SRCFILES} test.lis

```

```
::::::::::::::::
Makefile.deps
::::::::::::::::
     1  auxlib.o: auxlib.cc auxlib.h
     2  cppstrtok.o: cppstrtok.cc auxlib.h
::::::::::::::::
foo.oc
::::::::::::::::
     1  line 1// $Id: foo.oc,v 1.3 2013-09-19 18:03:21-07 - - $
     2  __FILE__ __LINE__ __DATE__ __TIME__
     3  foo.oc, line 3.
     4  #include "foo1.oh"
     5  foo.oc, line 5.
     6  #include "foo2.oh"
     7  /* Comment */ on line 7
     8  FOO1 + FOO2;
     9  foo.oc, line 9, last line.
::::::::::::::::
foo1.oh
::::::::::::::::
     1  // $Id: foo1.oh,v 1.2 2011-09-29 19:06:34-07 - - $
     2  __FILE__ __LINE__ __DATE__ __TIME__
     3  foo1.h, line 3.
     4  foo1.h, line 4.
     5  // Comment.
     6  foo1.h, line 6. /* Comment */ last line
     7  #define FOO1 "foo1"
::::::::::::::::
foo2.oh
::::::::::::::::
     1  // $Id: foo2.oh,v 1.2 2011-09-29 19:06:34-07 - - $
     2  __FILE__ __LINE__ __DATE__ __TIME__
     3  foo2.h, line 3.
     4  foo2.h, line 4.
     5  // Comment.
     6  foo2.h, line 6. /* Comment */ last line
     7  #define FOO2 "foo2"
::::::::::::::::
test.out
::::::::::::::::
     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 "Sep 25 2013" "13:52:51"]
    14  token 2.1: ["foo.oc"]
    15  token 2.2: [2]
    16  token 2.3: ["Sep]
    17  token 2.4: [25]
    18  token 2.5: [2013"]
    19  token 2.6: ["13:52:51"]
    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 "Sep 25 2013" "13:52:51"]
    28  token 2.1: ["foo1.oh"]
    29  token 2.2: [2]
    30  token 2.3: ["Sep]
    31  token 2.4: [25]
    32  token 2.5: [2013"]
    33  token 2.6: ["13:52:51"]
    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 "Sep 25 2013" "13:52:51"]
    59  token 2.1: ["foo2.oh"]
    60  token 2.2: [2]
    61  token 2.3: ["Sep]
    62  token 2.4: [25]
    63  token 2.5: [2013"]
    64  token 2.6: ["13:52:51"]
    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.]
::::::::::::::::
test.err
::::::::::::::::
     1  ==26318== Memcheck, a memory error detector
     2  ==26318== Copyright (C) 2002-2012, and GNU GPL'd, by Julian Seward et al.
     3  ==26318== Using Valgrind-3.8.1 and LibVEX; rerun with -h for copyright info
     4  ==26318== Command: cppstrtok foo.oc
     5  ==26318==
     6  ==26318==
     7  ==26318== HEAP SUMMARY:
     8  ==26318==     in use at exit: 0 bytes in 0 blocks
     9  ==26318==   total heap usage: 4 allocs, 4 frees, 386 bytes allocated
    10  ==26318==
    11  ==26318== All heap blocks were freed -- no leaks are possible
    12  ==26318==
    13  ==26318== For counts of detected and suppressed errors, rerun with: -v
    14  ==26318== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 6 from 6)
```