include_crawler – informal design document

```
/*
 * usage: ./include_crawler [-Idir] ... file.c|file.l|file.y ...
 *
 * processes the c/yacc/lex source file arguments, outputting the dependencies
 * between the corresponding .o file, the .c/l/y source file, and any included
 * .h files
 *
 * each .h file is also processed to yield a dependency between it and any
 * included .h files
 *
 * these dependencies are written to standard output in a form compatible with
 * make; for example, assume that foo.c includes inc1.h, and inc1.h includes
 * inc2.h and inc3.h; this results in
 *
 *              foo.o: foo.c inc1.h inc2.h inc3.h
 *
 * note that system includes (i.e. those in angle brackets) are NOT processed
 *
 * include_crawler uses the CPATH environment variable, which can contain a
 * set of directories separated by ':' to find included files;
 * if any additional directories are specified in the command line,
 * these are prepended to those in CPATH, left to right
 *
 * for example, if CPATH is "/home/user/include:/usr/local/group/include",
 * and if "-Ifoo/bar/include" is specified on the command line, then when
 * processing
 *         #include "x.h"
 * x.h will be located by searching for the following files in this order
 *
 *     ./x.h
 *     foo/bar/include/x.h
 *     /home/user/include/x.h
 *     /usr/local/group/include/x.h
 */

/*
 * general design of main()
 * ========================
 *
 * 1. look up CPATH in environment
 * 2. assemble directories list from ".", any -Idir flags, and fields in CPATH
 *    (if it is defined)
 * 3. create a master dictionary to map from file name to files upon which
 *    it depends
 * 4. create a workQ of files that need to be processed for #include lines
 * 5. for each file argument (after -Idir flags)
 *    a. insert mapping from file.o to file.ext (where ext is c, y, or l) into
 *       dictionary
 *    b. insert mapping from file.ext to empty list into dictionary
 *    c. append file.ext on workQ
 * 6. for each file on the workQ
 *    a. create a linked list for names of files #include'd
 *    b. invoke process(name, linkedlist, dictionary, workQ)
 * 7. for each file argument (after -Idir flags)
 *    a. create a set in which to track file names already printed
 *    b. create a linked list to track dependencies yet to print
 *    c. print "foo.o:", insert "foo.o" into set
 *       and append "foo.o" to linked list
 *    d. invoke print_dependencies(dictionary, printed, to_process)
 *
 * general design for process()
 * ============================
 *
 * 1. open the file
 * 2. for each line of the file
 *    a. skip leading whitespace
 *    b. if match "^#include"
 *       i. skip leading whitespace after "include"
 *       ii. if next character is '"'
 *           * collect remaining characters of file name (up to '"')
 *           * append file name to dependency list for this open file
 *           * if file name not already in the dictionary
 *             - create empty linked list of dependencies
 *             - insert mapping from file name to empty list in dictionary
 *             - append file name to workQ
 * 3. close file
 *
 * general design for print_dependencies()
 * =======================================
 *
 * 1. while there is still a file in the to_process linked list
 * 2. fetch next file from to_process
 * 3. lookup up the file in the dictionary, yielding the linked list of deps
 * 4. while there is a next element
 *    a. if the filename is already in the printed hash table, continue
 *    b. print the filename
 *    c. insert into printed
 *    d. append to to_process
 *
 * Additional helper functions
 * ===========================
 *
 * parse_file() - breaks up filename into root and extension
 * open_file()  - attempts to open a filename using the search path defined
 *              by the directories list.
 */
```