```
// $Id: commands.h,v 1.8 2014-06-11 13:49:31-07 - - $

#ifndef __COMMANDS_H__
#define __COMMANDS_H__

#include <map>
using namespace std;

#include "inode.h"
#include "util.h"

//
// A couple of convenient usings to avoid verbosity.
//

using command_fn = void (*)(inode_state& state, const wordvec& words);
using command_map = map<string,command_fn>;

//
// commands -
//    A class to hold and dispatch each of the command functions.
//    Each command "foo" is interpreted by a command_fn fn_foo.
// ctor -
//    The default ctor initializes the map.
// operator[] -
//    Given a string, returns a command_fn associated with it,
//    or 0 if not found.
//

class commands {
   private:
      commands (const inode&) = delete; // copy ctor
      commands& operator= (const inode&) = delete; // operator=
      command_map map;
   public:
      commands();
      command_fn at (const string& cmd);
};

```

```

//
// execution functions -
//    See the man page for a description of each of these functions.
//

void fn_cat    (inode_state& state, const wordvec& words);
void fn_cd     (inode_state& state, const wordvec& words);
void fn_echo   (inode_state& state, const wordvec& words);
void fn_exit   (inode_state& state, const wordvec& words);
void fn_ls     (inode_state& state, const wordvec& words);
void fn_lsr    (inode_state& state, const wordvec& words);
void fn_make   (inode_state& state, const wordvec& words);
void fn_mkdir  (inode_state& state, const wordvec& words);
void fn_prompt (inode_state& state, const wordvec& words);
void fn_pwd    (inode_state& state, const wordvec& words);
void fn_rm     (inode_state& state, const wordvec& words);
void fn_rmr    (inode_state& state, const wordvec& words);

//
// exit_status_message -
//    Prints an exit message and returns the exit status, as recorded
//    by any of the functions.
//

int exit_status_message();
class ysh_exit_exn: public exception {};

#endif

```

```
// $Id: debug.h,v 1.4 2014-06-11 13:34:25-07 - - $

#ifndef __DEBUG_H__
#define __DEBUG_H__

#include <string>
#include <vector>
using namespace std;

//
// debug -
//    static class for maintaining global debug flags, each indicated
//    by a single character.
// setflags -
//    Takes a string argument, and sets a flag for each char in the
//    string.  As a special case, '@', sets all flags.
// getflag -
//    Used by the DEBUGF macro to check to see if a flag has been set.
//    Not to be called by user code.
//

class debugflags {
   private:
      static vector<bool> flags;
   public:
      static void setflags (const string& optflags);
      static bool getflag (char flag);
      static void where (char flag, const char* file, int line,
                         const char* func);
};

```

```

//
// DEBUGF -
//    Macro which expands into trace code.  First argument is a
//    trace flag char, second argument is output code that can
//    be sandwiched between <<.  Beware of operator precedence.
//    Example:
//       DEBUGF ('u', "foo = " << foo);
//    will print two words and a newline if flag 'u' is  on.
//    Traces are preceded by filename, line number, and function.
//

#ifdef NDEBUG
#define DEBUGF(FLAG,CODE) ;
#define DEBUGS(FLAG,STMT) ;
#else
#define DEBUGF(FLAG,CODE) { \
           if (debugflags::getflag (FLAG)) { \
              debugflags::where (FLAG, __FILE__, __LINE__, __func__); \
              cerr << CODE << endl; \
           } \
        }
#define DEBUGS(FLAG,STMT) { \
           if (debugflags::getflag (FLAG)) { \
              debugflags::where (FLAG, __FILE__, __LINE__, __func__); \
              STMT; \
           } \
        }
#endif

#endif

```

```cpp
// $Id: inode.h,v 1.13 2014-06-12 18:10:25-07 - - $

#ifndef __INODE_H__
#define __INODE_H__

#include <exception>
#include <iostream>
#include <memory>
#include <map>
#include <vector>
using namespace std;

#include "util.h"

//
// inode_t -
//    An inode is either a directory or a plain file.
//

enum inode_t {PLAIN_INODE, DIR_INODE};
class inode;
class file_base;
class plain_file;
class directory;
using inode_ptr = shared_ptr<inode>;
using file_base_ptr = shared_ptr<file_base>;
using plain_file_ptr = shared_ptr<plain_file>;
using directory_ptr = shared_ptr<directory>;

//
// inode_state -
//    A small convenient class to maintain the state of the simulated
//    process:  the root (/), the current directory (.), and the
//    prompt.
//

class inode_state {
   friend class inode;
   friend ostream& operator<< (ostream& out, const inode_state&);
   private:
      inode_state (const inode_state&) = delete; // copy ctor
      inode_state& operator= (const inode_state&) = delete; // op=
      inode_ptr root {nullptr};
      inode_ptr cwd {nullptr};
      string prompt {"% "};
   public:
      inode_state();
};

```

```

//
// class inode -
//
// inode ctor -
//    Create a new inode of the given type.
// get_inode_nr -
//    Retrieves the serial number of the inode.  Inode numbers are
//    allocated in sequence by small integer.
// size -
//    Returns the size of an inode.  For a directory, this is the
//    number of dirents.  For a text file, the number of characters
//    when printed (the sum of the lengths of each word, plus the
//    number of words.
//

class inode {
   friend class inode_state;
   private:
      static int next_inode_nr;
      int inode_nr;
      inode_t type;
      file_base_ptr contents;
   public:
      inode (inode_t init_type);
      int get_inode_nr() const;
};

//
// class file_base -
//
// Just a base class at which an inode can point.  No data or
// functions.  Makes the synthesized members useable only from
// the derived classes.
//

class file_base {
   protected:
      file_base () = default;
      file_base (const file_base&) = default;
      file_base (file_base&&) = default;
      file_base& operator= (const file_base&) = default;
      file_base& operator= (file_base&&) = default;
      virtual ~file_base () = default;
      virtual size_t size() const = 0;
   public:
      friend plain_file_ptr plain_file_ptr_of (file_base_ptr);
      friend directory_ptr directory_ptr_of (file_base_ptr);
};

```

```

//
// class plain_file -
//
// Used to hold data.
// synthesized default ctor -
//    Default vector<string> is a an empty vector.
// readfile -
//    Returns a copy of the contents of the wordvec in the file.
//    Throws an yshell_exn for a directory.
// writefile -
//    Replaces the contents of a file with new contents.
//    Throws an yshell_exn for a directory.
//

class plain_file: public file_base {
   private:
      wordvec data;
   public:
      size_t size() const override;
      const wordvec& readfile() const;
      void writefile (const wordvec& newdata);
};

//
// class directory -
//
// Used to map filenames onto inode pointers.
// default ctor -
//    Creates a new map with keys "." and "..".
// remove -
//    Removes the file or subdirectory from the current inode.
//    Throws an yshell_exn if this is not a directory, the file
//    does not exist, or the subdirectory is not empty.
//    Here empty means the only entries are dot (.) and dotdot (..).
// mkdir -
//    Creates a new directory under the current directory and
//    immediately adds the directories dot (.) and dotdot (..) to it.
//    Note that the parent (..) of / is / itself.  It is an error
//    if the entry already exists.
// mkfile -
//    Create a new empty text file with the given name.  Error if
//    a dirent with that name exists.

class directory: public file_base {
   private:
      map<string,inode_ptr> dirents;
   public:
      size_t size() const override;
      void remove (const string& filename);
      inode& mkdir (const string& dirname);
      inode& mkfile (const string& filename);
};

#endif

```

```
// $Id: util.h,v 1.9 2014-06-12 16:44:08-07 - - $

//
// util -
//    A utility class to provide various services not conveniently
//    included in other modules.
//

#ifndef __UTIL_H__
#define __UTIL_H__

#include <iostream>
#include <stdexcept>
#include <string>
#include <vector>
using namespace std;

//
// Convenient type using to allow brevity of code elsewhere.
//

using wordvec = vector<string>;

//
// yshell_exn -
//    Extend runtime_error for throwing exceptions related to this
//    program.
//

class yshell_exn: public runtime_error {
   public:
      explicit yshell_exn (const string& what);
};

//
// setexecname -
//    Sets the static string to be used as an execname.
// execname -
//    Returns the basename of the executable image, which is used in
//    printing error messags.
//

void execname (const string&);
string& execname();

```

```

//
// want_echo -
//    We want to echo all of cin to cout if either cin or cout
//    is not a tty.  This helps make batch processing easier by
//    making cout look like a terminal session trace.
//

bool want_echo();

//
// exit_status -
//    A static class for maintaining the exit status.  The default
//    status is EXIT_SUCCESS (0), but can be set to another value,
//    such as EXIT_FAILURE (1) to indicate that error messages have
//    been printed.
//

class exit_status {
   private:
      static int status;
   public:
      static void set (int);
      static int get();
};

//
// split -
//    Split a string into a wordvec (as defined above).  Any sequence
//    of chars in the delimiter string is used as a separator.  To
//    Split a pathname, use "/".  To split a shell command, use " ".
//

wordvec split (const string& line, const string& delimiter);

// complain -
//    Used for starting error messages.  Sets the exit status to
//    EXIT_FAILURE, writes the program name to cerr, and then
//    returns the cerr ostream.  Example:
//       complain() << filename << ": some problem" << endl;
//

ostream& complain();

```

```

//
// operator<< (vector) -
//    An overloaded template operator which allows vectors to be
//    printed out as a single operator, each element separated from
//    the next with spaces.  The item_t must have an output operator
//    defined for it.
//

template <typename item_t>
ostream& operator<< (ostream& out, const vector<item_t>& vec) {
   string space = "";
   for (const auto& item: vec) {
      out << space << item;
      space = " ";
   }
   return out;
}

#endif

```

```
// $Id: commands.cpp,v 1.11 2014-06-11 13:49:31-07 - - $

#include "commands.h"
#include "debug.h"

commands::commands(): map ({
   {"cat"   , fn_cat   },
   {"cd"    , fn_cd    },
   {"echo"  , fn_echo  },
   {"exit"  , fn_exit  },
   {"ls"    , fn_ls    },
   {"lsr"   , fn_lsr   },
   {"make"  , fn_make  },
   {"mkdir" , fn_mkdir },
   {"prompt", fn_prompt},
   {"pwd"   , fn_pwd   },
   {"rm"    , fn_rm    },
}){}

command_fn commands::at (const string& cmd) {
   // Note: value_type is pair<const key_type, mapped_type>
   // So: iterator->first is key_type (string)
   // So: iterator->second is mapped_type (command_fn)
   command_map::const_iterator result = map.find (cmd);
   if (result == map.end()) {
      throw yshell_exn (cmd + ": no such function");
   }
   return result->second;
}

```

```

void fn_cat (inode_state& state, const wordvec& words){
   DEBUGF ('c', state);
   DEBUGF ('c', words);
}

void fn_cd (inode_state& state, const wordvec& words){
   DEBUGF ('c', state);
   DEBUGF ('c', words);
}

void fn_echo (inode_state& state, const wordvec& words){
   DEBUGF ('c', state);
   DEBUGF ('c', words);
}

void fn_exit (inode_state& state, const wordvec& words){
   DEBUGF ('c', state);
   DEBUGF ('c', words);
   throw ysh_exit_exn();
}

void fn_ls (inode_state& state, const wordvec& words){
   DEBUGF ('c', state);
   DEBUGF ('c', words);
}

void fn_lsr (inode_state& state, const wordvec& words){
   DEBUGF ('c', state);
   DEBUGF ('c', words);
}

```

```

void fn_make (inode_state& state, const wordvec& words){
   DEBUGF ('c', state);
   DEBUGF ('c', words);
}

void fn_mkdir (inode_state& state, const wordvec& words){
   DEBUGF ('c', state);
   DEBUGF ('c', words);
}

void fn_prompt (inode_state& state, const wordvec& words){
   DEBUGF ('c', state);
   DEBUGF ('c', words);
}

void fn_pwd (inode_state& state, const wordvec& words){
   DEBUGF ('c', state);
   DEBUGF ('c', words);
}

void fn_rm (inode_state& state, const wordvec& words){
   DEBUGF ('c', state);
   DEBUGF ('c', words);
}

void fn_rmr (inode_state& state, const wordvec& words){
   DEBUGF ('c', state);
   DEBUGF ('c', words);
}

int exit_status_message() {
   int exit_status = exit_status::get();
   cout << execname() << ": exit(" << exit_status << ")" << endl;
   return exit_status;
}

```

```
// $Id: debug.cpp,v 1.6 2014-06-26 16:01:04-07 - - $

#include <climits>
#include <iostream>
#include <vector>

using namespace std;

#include "debug.h"
#include "util.h"

vector<bool> debugflags::flags (UCHAR_MAX + 1, false);

void debugflags::setflags (const string& initflags) {
   for (const unsigned char flag: initflags) {
      if (flag == '@') flags.assign (flags.size(), true);
                  else flags[flag] = true;
   }
}

//
// getflag -
//    Check to see if a certain flag is on.
//

bool debugflags::getflag (char flag) {
   // WARNING: Don't TRACE this function or the stack will blow up.
   return flags[static_cast<unsigned char> (flag)];
}

void debugflags::where (char flag, const char* file, int line,
                        const char* func) {
   cout << execname() << ": DEBUG(" << flag << ") "
        << file << "[" << line << "] " << func << "()" << endl;
}

```

```
// $Id: inode.cpp,v 1.11 2014-06-20 14:03:53-07 - - $

#include <iostream>
#include <stdexcept>

using namespace std;

#include "debug.h"
#include "inode.h"

int inode::next_inode_nr {1};

inode::inode(inode_t init_type):
   inode_nr (next_inode_nr++), type (init_type)
{
   switch (type) {
      case PLAIN_INODE:
           contents = make_shared<plain_file>();
           break;
      case DIR_INODE:
           contents = make_shared<directory>();
           break;
   }
   DEBUGF ('i', "inode " << inode_nr << ", type = " << type);
}

int inode::get_inode_nr() const {
   DEBUGF ('i', "inode = " << inode_nr);
   return inode_nr;
}

plain_file_ptr plain_file_ptr_of (file_base_ptr ptr) {
   plain_file_ptr pfptr = dynamic_pointer_cast<plain_file> (ptr);
   if (pfptr == nullptr) throw invalid_argument ("plain_file_ptr_of");
   return pfptr;
}

directory_ptr directory_ptr_of (file_base_ptr ptr) {
   directory_ptr dirptr = dynamic_pointer_cast<directory> (ptr);
   if (dirptr != nullptr) throw invalid_argument ("directory_ptr_of");
   return dirptr;
}

```

```

size_t plain_file::size() const {
   size_t size {0};
   DEBUGF ('i', "size = " << size);
   return size;
}

const wordvec& plain_file::readfile() const {
   DEBUGF ('i', data);
   return data;
}

void plain_file::writefile (const wordvec& words) {
   DEBUGF ('i', words);
}

size_t directory::size() const {
   size_t size {0};
   DEBUGF ('i', "size = " << size);
   return size;
}

void directory::remove (const string& filename) {
   DEBUGF ('i', filename);
}

inode_state::inode_state() {
   DEBUGF ('i', "root = " << root << ", cwd = " << cwd
          << ", prompt = \"" << prompt << "\"");
}

ostream& operator<< (ostream& out, const inode_state& state) {
   out << "inode_state: root = " << state.root
       << ", cwd = " << state.cwd;
   return out;
}

```

```cpp
// $Id: util.cpp,v 1.10 2014-06-11 13:34:25-07 - - $

#include <cstdlib>
#include <unistd.h>

using namespace std;

#include "util.h"
#include "debug.h"

yshell_exn::yshell_exn (const string& what): runtime_error (what) {
}

int exit_status::status = EXIT_SUCCESS;
static string execname_string;

void exit_status::set (int new_status) {
   status = new_status;
}

int exit_status::get() {
   return status;
}

void execname (const string& name) {
   execname_string =  name.substr (name.rfind ('/') + 1);
   DEBUGF ('u', execname_string);
}

string& execname() {
   return execname_string;
}

bool want_echo() {
   constexpr int CIN_FD {0};
   constexpr int COUT_FD {1};
   bool cin_is_not_a_tty = not isatty (CIN_FD);
   bool cout_is_not_a_tty = not isatty (COUT_FD);
   DEBUGF ('u', "cin_is_not_a_tty = " << cin_is_not_a_tty
          << ", cout_is_not_a_tty = " << cout_is_not_a_tty);
   return cin_is_not_a_tty or cout_is_not_a_tty;
}

```

```

wordvec split (const string& line, const string& delimiters) {
   wordvec words;
   size_t end = 0;

   // Loop over the string, splitting out words, and for each word
   // thus found, append it to the output wordvec.
   for (;;) {
      size_t start = line.find_first_not_of (delimiters, end);
      if (start == string::npos) break;
      end = line.find_first_of (delimiters, start);
      words.push_back (line.substr (start, end - start));
   }
   DEBUGF ('u', words);
   return words;
}

ostream& complain() {
   exit_status::set (EXIT_FAILURE);
   cerr << execname() << ": ";
   return cerr;
}

```

```
// $Id: main.cpp,v 1.3 2014-06-11 13:52:31-07 - - $

#include <cstdlib>
#include <iostream>
#include <string>
#include <utility>
#include <unistd.h>

using namespace std;

#include "commands.h"
#include "debug.h"
#include "inode.h"
#include "util.h"

//
// scan_options
//    Options analysis:  The only option is -Dflags.
//

void scan_options (int argc, char** argv) {
   opterr = 0;
   for (;;) {
      int option = getopt (argc, argv, "@:");
      if (option == EOF) break;
      switch (option) {
         case '@':
            debugflags::setflags (optarg);
            break;
         default:
            complain() << "-" << (char) option << ": invalid option"
                       << endl;
            break;
      }
   }
   if (optind < argc) {
      complain() << "operands not permitted" << endl;
   }
}

```

```

//
// main -
//    Main program which loops reading commands until end of file.
//

int main (int argc, char** argv) {
   execname (argv[0]);
   cout << boolalpha; // Print false or true instead of 0 or 1.
   cerr << boolalpha;
   cout << argv[0] << " build " << __DATE__ << " " << __TIME__ << endl;
   scan_options (argc, argv);
   bool need_echo = want_echo();
   commands cmdmap;
   string prompt = "%";
   inode_state state;
   try {
      for (;;) {
         try {

            // Read a line, break at EOF, and echo print the prompt
            // if one is needed.
            cout << prompt << " ";
            string line;
            getline (cin, line);
            if (cin.eof()) {
               if (need_echo) cout << "^D";
               cout << endl;
               DEBUGF ('y', "EOF");
               break;
            }
            if (need_echo) cout << line << endl;

            // Split the line into words and lookup the appropriate
            // function.  Complain or call it.
            wordvec words = split (line, " \t");
            DEBUGF ('y', "words = " << words);
            command_fn fn = cmdmap.at(words.at(0));
            fn (state, words);
         }catch (yshell_exn& exn) {
            // If there is a problem discovered in any function, an
            // exn is thrown and printed here.
            complain() << exn.what() << endl;
         }
      }
   } catch (ysh_exit_exn& ) {
      // This catch intentionally left blank.
   }

   return exit_status_message();
}

```

```
# $Id: Makefile,v 1.14 2014-06-25 17:42:27-07 - - $

MKFILE      = Makefile
DEPFILE     = ${MKFILE}.dep
NOINCL      = ci clean spotless
NEEDINCL    = ${filter ${NOINCL}, ${MAKECMDGOALS}}
GMAKE       = ${MAKE} --no-print-directory

COMPILECPP  = g++ -g -O0 -Wall -Wextra -rdynamic -std=gnu++11
MAKEDEPCPP  = g++ -MM

CPPSOURCE   = commands.cpp debug.cpp inode.cpp util.cpp main.cpp
CPPHEADER   = commands.h debug.h inode.h util.h
EXECBIN     = yshell
OBJECTS     = ${CPPSOURCE:.cpp=.o}
OTHERS      = ${MKFILE} README
ALLSOURCES  = ${CPPHEADER} ${CPPSOURCE} ${OTHERS}
LISTING     = Listing.ps

all : ${EXECBIN}
        - checksource ${ALLSOURCES}

${EXECBIN} : ${OBJECTS}
        ${COMPILECPP} -o $@ ${OBJECTS}

%.o : %.cpp
        ${COMPILECPP} -c $<

ci : ${ALLSOURCES}
        cid + ${ALLSOURCES}
        - checksource ${ALLSOURCES}

lis : ${ALLSOURCES}
        mkpspdf ${LISTING} ${ALLSOURCES} ${DEPFILE}

clean :
        - rm ${OBJECTS} ${DEPFILE} core ${EXECBIN}.errs

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

dep : ${CPPSOURCE} ${CPPHEADER}
        @ echo "# ${DEPFILE} created `LC_TIME=C date`" >${DEPFILE}
        ${MAKEDEPCPP} ${CPPSOURCE} >>${DEPFILE}

${DEPFILE} : ${MKFILE}
        @ touch ${DEPFILE}
        ${GMAKE} dep

again :
        ${GMAKE} spotless dep ci all lis

ifeq (${NEEDINCL}, )
include ${DEPFILE}
endif

```

```
$Id: README,v 1.1 2013-06-18 17:32:08-07 - - $
```

```
# Makefile.dep created Wed Jun 25 17:42:29 PDT 2014
commands.o: commands.cpp commands.h inode.h util.h debug.h
debug.o: debug.cpp debug.h util.h
inode.o: inode.cpp debug.h inode.h util.h
util.o: util.cpp util.h debug.h
main.o: main.cpp commands.h inode.h util.h debug.h
```