```
1
2   class File {
3   public:
4     //inode metadata;                    // for use by all files
5     struct stat metadata;                // for use by all files
6     // one or the other of the following (data and frame) should be empty.
7     string data;                         // for use by regular files
8     vector<dirent_frame> dentries;       // for use by directories
9     //list<dirent_frame> dentries;       // it'd be cleaner to make this a list
10  };
11
12
13  class Ilist {
14  public:
15    int count;
16    map<ino_t,File> entry;
17    int next() {
18      static int count = 2;  // ino counter stats at 2.
19      return count++;
20    }
21  } ilist;
22
23
24  Implemented and seem okay
25  void initialize()
26  int my_lstat( const char* path, struct stat *statbuf )
27  int my_fstat( ino_t fh, struct stat* statbuf )
28  int my_mkdir( const char *path, mode_t mode )
29  MY_DIR* fopendir( ino_t fh )
30  dirent* my_readdir( MY_DIR* dirp )
31  int my_closedir( MY_DIR* dirp )
32  MY_DIR * my_opendir( char path[PATH_MAX] )
33  int my_mknod( const char *path, mode_t mode, dev_t dev ) // being debugged
34  int my_rmdir( const char *path ) // erase a dirent_frame from dentries
35
36  Helpers (implemented and seem okay)
37  int ls(string s)
38  ino_t lookup( string name, ino_t fh )
39  ino_t find_ino( string path )
40  File* find_file( ino_t ino )
41  File* find_file( string s )
42  void show_stat( struct stat& root ) // displays a stat as Pfeiffer does.
43  split(const string s, const string pat )
44  join( const vector<string> v, const string pat, int start=0, int end=-1 )
45
46  Under devellopment
47  int lslr(string path) // It will work like "ls -lR"
48
49  ============================================================
50
51  High priority (Get these working first.)
52  int my_rename( const char *path, const char *newpath )
53  // changes name on a dentry and/or moves it to another directory
54  int my_link(const char *path, const char *newpath)  // needs testing/debugging
55  // my_link creates hard links, i.e., puts dentries into directories.  It has
56  // a preliminary implementation in the current code.
57  int my_unlink( const char *path ) // must destruct file when nlink == 0.
58  int my_creat( const char *fpath, mode_t mode )  // can probably use my_mknod
59  int my_open( const char *path, int flags ) // with certain flags,
60  // open() must create a file, say via my_creat()
61  int my_close( int fh )
62  int my_pread( int fh, char *buf, size_t size, off_t offset )
63  int my_pwrite( int fh, const char *buf, size_t size, off_t offset )
64
65  Medium priority (These have to do with protections and ownership.)
66  int my_access( const char *fpath, int mask ) // check bits of mode field
67  int my_chown(const char *path, uid_t uid, gid_t gid)
68  int my_chmod(const char *path, mode_t mode)
69  // changes permission bits in the metadata's mode field
70
71  Low priority (note that we won't implement symbolic links for now.)
72  int my_readlink( const char *path, char *link, size_t size )
73  int my_symlink(const char *path, const char *link)
```



```
74  int my_utime(const char *path, struct utimbuf *ubuf)
75  int my_fdatasync( ino_t fh )
76  int my_fsync( ino_t fh )
77  int my_ftruncate( ino_t fh, off_t offset )
78  int my_truncate(const char *path, off_t newsize)
79  int my_statvfs(const char *fpath, struct statvfs *statv)
80
81  Very low priority  (stuff having to do with ...xattr)
82  int my_lsetxattr( const char *fpath, const char *name, const char *value, size_t
     size, int flags )
83  int my_lgetxattr( const char *fpath, const char *name, char *value, size_t size,
     int flags )
84  int my_llistxattr( const char *path, char *list, size_t size )
85  int my_lremovexattr( const char *path, const char *name )
86
87
88
89
```