Within our implementation, out file-system application will have a local copy of the super block for the filesystem called sblock. At all times, this sblock will be either current with the sblock in the file, or will be updated awaiting to be committed to the file.

The local sblock is a C struct called super. This contains an array of char called freeblocks, which signals which blocks are free and which aren't. Then we have a pointer to an array of inodes within it as well. These inodes are the inodes that reference each file in out file system.
The inode is C struct which has a name, a size, an array of blockPointers that this file uses and a number signifying how many blocks this inode uses.

The main function of the file-system program will read in a file specified and do the specific operations depending on whats written on the file. Thes e are the create, delete, write, read, and ls functions.

The create function will first allocate new space for an inode. Then it will copy over the name and the size into the new inode struct. It will then check over the inodes to see how many are being used. If not all are being used, then there should still be enough space, otherwise theres not enough space. We also check for the name to see if that file name is available or not.

The Delete function looks for the inode with the desired name. If it is found, then an empty buffer will be written over the blocks used by the inode. Those block indexes are saved so that later it can be signalled free in the super block.

The read function will look for the inode with the desired name, then, if found, will move the pointer to the block number of the block pointer of that inode (multiplied by 1024) and read in the data.

The write function essentially does the same as re ad, except instead of the instance of reading, it will write to that location.

The ls() function simply loops through the inodes and prints their names.