Issues we Faced

Cameron Daly and Anthony Duan

Issue 1: How to Utilize the Inodes / Blocks / Entries

- As “data blocks” could be used to hold either directory entries or actual data, there was some initial confusion about whether we should have attempted to place both in the same block
  - This would have reduced the available amount of space for any individual files’ data
  - Inodes would reference the data blocks, but these could also technically be directory blocks
  - We ended up dealing with the formatting of blocks as their creation came along
  - I.e. creating “data” or “directory” blocks both in the blocks 67 and onwards range
- We used helper functions to convert data to the various variables of a block
  - Adding to directories, looking up info in them, reading and writing to inodes and blocks
  - This also made the specified functions a lot easier to write, as the inode allocations and block readings were taken care of
  - These helper functions didn’t care what specific kind of block something was; there was a certain degree of “abstraction” if it may be called that

Issue 2: Creating new structs, switching from block to block

- The creation of the PDOS_FILE typedef struct was among the later parts of our project to be completed
  - This required some thinking of the exact position of the pointer in the file, as well as inode location
  - It also required variables for modes, which became important in pdos_fgetc() and pdos_fputc() in particular.
  - We created a few “if” statements for this purpose
- Another typedef struct that created some issue was DIR_ENTRY
  - We knew multiple DIR_ENTRY structs went into a DIR_BLOCK, and weren’t sure how to best size it – we knew the different components, but the size of the buffer was one issue
  - We needed 4 bytes in the DIR_BLOCK to keep track of DIR_ENTRYs, so that left 1020 – we considered dividing it into 20 of 51 bytes, but that was so inefficient buffer-wise that we attempted 32 of 32 bytes, changed to 31 of 32 bytes, leaving some at the end unused.

Issue 3: Allocating data and inode blocks

- The data blocks had to exist, but could obviously not be allocated to a file or directory upon creation
  - In theory, there could be up to 15 data blocks, but then we'd run out of room if we created too many files
  - We couldn't allocate *distinct* sets of data blocks, but if we randomly assigned a group of data blocks to a file then what if there was a collision?
  - We only allocated single data blocks at a time, and only if we ran out of room in the file we were adding too – we kept track of file size for this purpose
  - When initializing a disk, we moreover only allocated the first inode and data block
  - This was intended to be more efficient
- This was different from the inode blocks
  - We realized that unlike data blocks, allocated to individual files as they went along, inode blocks had no such assignment; thus, we initialized them with the disk, albeit keeping them blank.
  - This avoided complicated code once the disk had already been made.