Steps for milestone 1:

Remember – in order to manipulate anything on disk, you must first bring it into memory

1) Determine if you need to format the volume or not.
   a. In fsInit is the initFileSystem function where you want to start this code
   b. Malloc a Block of memory as your VCB pointer and LBAread block 0
   c. You now have a pointer to a structure, so look at the signature (magic number) in your structure and see if it matches.
   d. If it matches you have initialized your volume already – If not you need to initialize it

2) If you need to initialize it
   a. Initialize the values in your volume control block
   b. Initialize free space (see that below)
   c. Initialize the root directory (see that below)
   d. Set the values returned from above in the VCB
   e. LBAwrite the VCB to block 0

3) Initialize the Free Space
   a. Let's assume you are using a bitmap for your free space management. Given the default size, you will have 19,531 blocks – so you need 19,531 bits or 2,442 bytes or 5 blocks (so from this alone you can see that the free space map is NOT part of the VCB, but rather the VCB would have the block number where your free space starts).
   b. Before you can initialize the bits, you must have the memory to do so. So malloc the space needed. In this example 5 * blockSize
   c. Now you have a pointer to at least 2442 bytes (actually 2560 bytes). Now you can first decide where you want to store this free space map (like from block 1 – 5 inclusive), and you know block 0 is the VCB so the first 6 bits should be marked used and the rest marked free.
   d. Now write that to disk with LBAwrite(theFreeSpaceMap, 5, 1) – that is write 5 blocks starting from block 1
   e. Return the starting block number of the free space to the VCB init that called you so it knows how to set the VCB structure variable that indicates where free space starts. Or mark it yourself if the VCB is a global structure.

4) Initialize the Root Directory
   a. First again, you need memory – how much?
   b. Decide how many Directory Entries (DE) you want for a directory (at least an initial amount). I will use 50 for this example.
   c. Now multiply the size of your directory entry by the number of entries. For example, if my directory entry is 60 bytes, I would multiply 60 * 50 = 3000 bytes needed.

d. Now you need to determine how many blocks you need. In the example that would be 6 blocks. But wait…. 6 blocks is 3072 bytes and I am only using 3000 bytes, and my directory entry is only 60 bytes, so I can actually fit 51 directory entries in the space of 3072 bytes (now only wasting 12 bytes instead of 72).

e. Now you have a pointer to an array of directory entries. Loop through them (in our case for (i = 0; i < 51; i++) and initialize each directory entry structure to be in a known free state.

f. Now ask the free space system for 6 blocks. It should return to you the starting block number for those 6 blocks.

g. Now set the first directory entry to the "." Directory. So, the name is ".", the size is 3060 bytes, it starts at block number (whatever the free space system returned above), the time stamps are the current time, it is a directory, etc.

h. Now set the second directory entry to "..", but since this is the root, it is the same as the "." except the name is ".." rather than "."

i. Now write the root directory – 6 blocks starting from block (what was returned from the free space allocation).

j. Return the starting block number of the root directory (or set it in the VCB yourself)

Note – while your system is running you only ever init or read the freespace map once. Keep it in memory so you can manipulate it as needed and when you change it just write it out to disk again so that the disk reflects the current state.