Question 7.3–1: (Solution, p 2) Describe the contents of the "object file" that is created by the compiler and subsequently given to the linker.

Question 7.3–2: (Solution, p 2) Describe two advantages of dynamic linking (load-time or run-time) over static linking (compile-time).

Question 7.3–3: (Solution, p 2) Describe two advantages of *static linking* (compile-time) over *dynamic linking* (load-time or run-time.

Question Disk–1: (Solution, p 2) Describe the notion of an *i-node* as it occurs in Unix filesystems.

Question Disk–2: (Solution, p 2) Name an advantage of an i-node for representing the data blocks occupied by a file instead of a FAT.

Question Disk–3: (Solution, p 3) Suppose we are have an antiquated 5.25-inch 360KB floppy disk, formatted using MS-DOS's FAT-16 filesystem with each block occupying two kilobytes. What is the maximum number of files we could fit onto the floppy? (Recall that each record in the directory file takes 32 bytes.)

Question Disk–4: (Solution, p 3) Storing file entries one after another in a directory means that a directory containing thousands of files could lead to very slow times for finding files, as each time a file is opened, the system would have to check every file entry to see which entry's name matches the requested filename. Describe a way to design the system to reduce this cost.

Solution 7.3–1: (Question, p 1) It has three major parts.

- the memory image of the compiled code, with 0's in place of any memory references whose locations are unknown at the time of compilation.
- a list of symbols (subroutine names, global variables) defined by the compiled program, accompanied with the offset where they are located within the memory image.
- a list of symbols used by the compiled program, accompanied with the offset of their use within the memory image.

In linking several object files together, the compiler will paste the memory images together, and then repair each address referring to a symbol by determining its location, which it can do by searching for the image defining that symbol.

Solution 7.3–2: (Question, p 1) I'll give you three.

- It allows multiple processes to share the dynamically linked subroutines in the same place in memory, saving duplication of memory.
- It reduces the size of executable files, since the dynamically linked subroutines need not be included.
- It allows the developer of the library to release an updated version, and all programs using the library would automatically begin using this newer version without the need to recompile the programs.

Solution 7.3–3: (Question, p 1) Again, I'll give you three.

- With statically linked libraries, all library routines are incorporated into the executable file. If the library is deleted, the routine's code is still in the executable file. Any dynamically linked libraries must exist in order for the program to work on a computer. There is the danger that somebody may delete a needed dynamically linked library, in which case the program will no longer be executable.
- Similarly, if somebody transfers the file to another computer with a different version of the dynamically linked library files, then the program may no longer behave the same. Thus, dynamically linked programs are less portable.
- Static linking is a more straightforward process for the operating system, resulting in a smaller and more reliable system.

Solution Disk–1: (Question, p 1) An i-node contains information about a file, including most importantly an array listing the blocks that the file occupies on the disk. The i-node information is stored in a node outside the directory file.

Solution Disk–2: (Question, p 1) I'll give you two.

- The i-node system use memory more efficiently, since we would only need the i-nodes of currently open files in memory, instead of the entire FAT.
- Also, i-nodes allow for the OS to quickly determine a block far into the file, whereas the FAT requires stepping through a linked list in memory.

Solution Disk–3: (Question, p 1) 176. [There are 180 total blocks. There would be at most 180 FAT entries, which would occupy $180 \cdot 2 = 360$ bytes (we multiply by two because each FAT entry is 2 bytes, or 16 bits); thus, the FAT fits into a single block. This leaves 179 data blocks, each of which could potentially hold a different file. The directory file representing these would require $179 \cdot 32 = 5728$ bytes, which will consume three of the data blocks. This leaves 176 data blocks unused by the FAT or the root directory.]

Solution Disk–4: (Question, p 1) We discussed two in class.

- We could design the directory file format to be structured around a hash table, so that when a file is requested, we could hash it to a particular bucket, and only the entries within that bucket need be searched.
- The OS could cache the most recently used file entries in memory. Then if a user reopened a file from disk, the OS could first check the cache, and if the entry is found, it would not have to look at the directory file on the disk at all.