CS 303
Summer 2013
Project 2

This project involves simulating a full file system on disk. Your program will model a hierarchical file system including the ability to store and erase files, add and remove subdirectories, and manage disk space. We will use a relatively small 'disk' of less than a hundred megabytes of simulated space—if you can manage that, you can manage a terabyte-sized disk as well.

Your program will need several data structures to carry out its work.

- For managing the sectors on the disk, a struct with two ints, for where a block of sectors begins and ends. (Rather than a 5,000 sector block of free spaces requiring a list of each specific block, all 5,000 of them, we'll use a single record for the boundaries.)
- To represent a file, a struct with 3 elements: a std::string for the name, an integer for the size in bytes, and a std::list of memory blocks to track the sectors that the file is stored in.
- A memory manager object. This will have two public functions (besides the constructor):
    - Malloc(int BytesNeeded), which takes the size in byes of a file. It returns a std::list of memory blocks, containing one or more blocks of space that have been allocated for the file. The total of all the memory blocks should be the minimum number of blocks needed to hold the file, bearing in mind that any the last block may not be fully allocated.
    - Free(). This takes a std::list of memory blocks (passed by reference) and adds the blocks in the list to the list of free space. Free() does not return a value.
    - The memory manager should internally (privately) maintain a list of blocks representing unused space on the disk.
- A Directory object. This is the largest component of the data structures needed. It has several parts:
    - Internally:
        - A std::string for the name of the directory
        - An optimized search tree (see below) to hold Files.
        - An optimized search tree (see below) to hold Directories (subfolders).
        - A list of memory blocks to hold the directory information itself. We assume that a directory can fit 128 entries in a single disk sector.
    - Public methods:
        - Add a File to a directory. This requires a const reference to a file (which we assume has an empty sector list) and a reference to the memory manager as parameters. DO NOT use a global memory manager. This method does not return anything. The memory manager is called to get a list of sectors to store the file, which are added to the file's list.
        - Remove a file from a directory. This takes 2 parameters, a string for the file name (passed by const reference) and the memory manager, passed by reference. It does not return anything. It removes the specified file from the directory. The memory blocks on the file's sector list are returned to the memory manager for re-use. If the file is not in the directory, the error is silently ignored.

        - MkDir. This makes a new, empty subdirectory. Its parameters are a string for the name and a reference to the memory manager. It returns a Boolean value indicating success or failure. For this program, it can simply return true at the end.
        - RmDir. This method removes a subdirectory. Parameters are the same as for MkDir. This method removes a subdirectory *if and only if it is already empty*. If the directory has any contents, the function returns false and no other action is taken.
        - There should be const methods to report how many files and how many subfolders the directory has.
        - You may find it helpful to add other methods: For example, to return the directory's name, or to remove all files at once, or to recursively empty an entire directory tree.
- Your main program will be given a text file of input. It will contain various commands to save or erase files, make directories, remove directories, etc. Your program will need to maintain appropriate data structures and print appropriate error messages. For example, attempting to remove a directory that is not empty should generate an error message. Saving the file c:/files/data/stuff/stuff.txt is an error if c:/files has no subdirectories. Your program will create 3 files: a log file showing what is being done and any error messages that result; a complete listing of the final directory tree; and a listing of free sectors in the disk.
- We will work with a small virtual disk: Only 50 megabytes (52,428,800 bytes) with 3200 clusters of 16K (16,384 bytes) each.
- To avoid the various hassles of backslashes being read as escape characters in C++, this program will use forward slashes to separate levels of the directory tree. Directory and file names will consist of letters, digits, and underscores, with no embedded white space or punctuation.
- Directories should use an advanced search tree for holding file and subfolder information. You may use AVL, Red-Black, or Splay trees. All are discussed in your text. For AVL or Red-Black trees, you can use physical deletion or lazy deletion, your choice. Splay trees must use physical deletion.
    - Because you'll be storing both Files and Directories in ordered search structures, you'll need to define comparison operators so you can order them. The comparison operators should compare the names of the files or directories. (Compare files to files and directories to directories.)
- As blocks of sectors are returned to the memory manager, they should be added to the back of the free list. This helps ensure that use is spread across the disk.