**CS 218 DATA STRUCTURES**
**ASSIGNMENT 1**
**SECTION A, B, C and D**
**Fall 2019**

**DUE**: Monday Sept 16, 2019

**NOTE:** Late submissions will not be accepted

**TO SUBMIT:** Documented and well written structured code in C++ on classroom. Undocumented code will be assigned a zero.

**PROBLEM**

The data/files on a computer's hard disk is stored in form of bytes. The bytes in a hard disk are grouped into **sectors** where each sector has fixed number of bytes (say 512 bytes in one sector). Each sector on hard disk has a unique ID. The job of a file system is to allocate available sector(s) when a new file is created and deallocate reserved sectors when an existing file is deleted. A file on hard disk can be stored in multiple sectors (when size of the file is larger than the size of the sector). The **FAT**(File Allocation Table) file system maintains a **pool** of available sectors where new data/file can be stored. Whenever a new file is created on disk, the file system uses available sectors to store the data of that file on the disk. Similarly when a file is deleted, the sectors allocated to that file are moved back to the pool of available sectors.

Whenever a new file is created, FAT computes the total number of sectors required to store the file depending on the size of file. If a contiguous block of sectors is not available in the pool then FAT stores the file as chain of contiguous blocks of sectors. A **block** is a contiguous collection of available sectors where number of sectors in each block may vary. A block maintains the Id of starting sector and total number of sectors. So a file in a FAT file system is a collection of blocks where each block maintains the address of next block of the file and file stores the address of first block. In order to make the whole process efficient, the file system maintains the *pool of available blocks instead of available sectors*.

In this assignment, we will simulate the FAT file system. The FAT file system includes: ***disk, list of available blocks (pool) and list of files already stored on the disk***. We define the disk as a one dimensional array of ***bytes/characters*** of size ***numOfSectors*sizeOfSector***, where ***sizeOfSector*** indicates the number of characters (bytes) stored in one sector. A ***pool*** of available blocks is kept in a linked list of ***block.*** A block has two fields: start_sector_ID, and total_sectors. **Files** are kept in a linked list ***Files*** where each node has a file as data**.** Each **file** has a name, size (number of characters) and a linked list of ***blocks.*** Saving files requires claiming a sufficient number of blocks from pool, if available and update the pool accordingly. Then the contents of the file have to be written to the sectors assigned to the file in the array ***disk***. Deletion of a file requires removing the list of blocks corresponding with the file and transferring the sectors assigned to this file back to pool. No changes are made in disk.

Suppose we have a disk of 100MB where each sector is 512 bytes. This means we will have sector ID's from 0 to 199.

Figure 1 shows a link list of blocks called pool. The first available block if of 2 sectors starting from sector ID 10 and last block is of 156 sectors starting at sector ID 44. Figure 2 shows a sample link list of Files. There are 3 files on the disk. The size of file F1 is 3616 bytes that require $\left\lceil \frac{3616}{512} \right\rceil = 8$ sectors. It also tells that the data of the file is stored in 3 blocks. First block starts at sector ID 0 and has 4 sectors, second block has 2 sectors that starts from sector ID 7 and last block also has 2 sectors starting from sector ID 12.

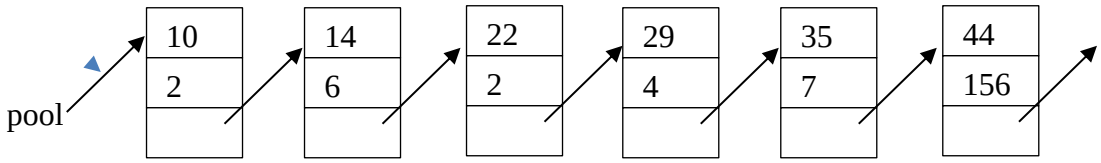


**Figure1: pool of blocks**

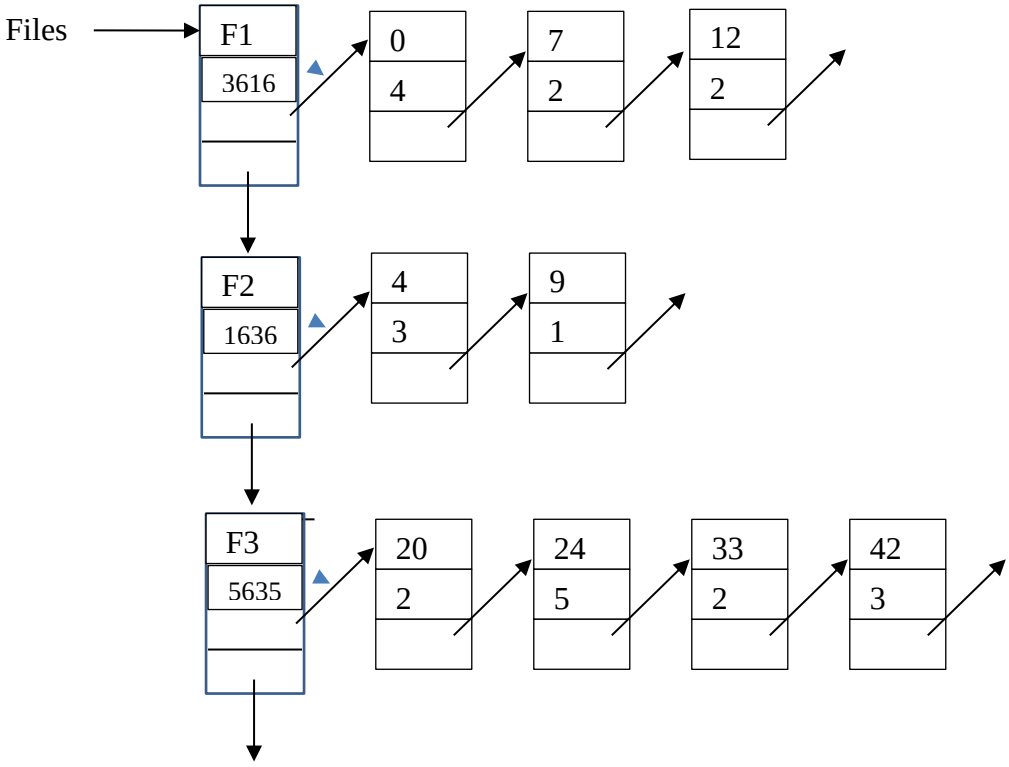


**Figure2: Files list containing three files F1, F2 and F3**

In order to implement the FAT file system you need to implement the following classes

**Class template of node**

*Data members:*

- data
- next

**Class template of singly linked list**

*Data member:*

- head
- tail
- size

**Class of block**
*Data member:*
- start_sector_ID
- total_sectors

**Class of file**
*Data members:*
- name
- size
- Link list of blocks

**Class FileSystem**
*Data members*:
- pool (Sorted Linked list of block)
- Files (Linked list of file)
- disk (dynamic array of characters)
- numOfSectors
- sizeOfSectors
- numOfSectorsInPool

*Member functions:*
**SaveFile:** The SaveFile function must take file name ***fname***, file content ***fcontent*** (character array) and file size ***fsize*** as parameter and should work as follows:

if ***fname*** matches with name of any saved file, then return false. If collective size of all blocks in the pool is less than ***fsize*** then return false. Otherwise Insert a new file ***f*** at the end of ***files***, make assignments: ***f.size = fsize, f.name = fname***. Then traverse the pool starting with the first block of pool and check if the number of sectors in current block is less than the ***fsize/sizeOfSector*** then allocate that block to ***f***, move it from the pool to the link list of block of the file (remember deleting this block from the pool and then inserting in the links list of f is inefficient), update ***fsize*** and move to the next block. If number of sectors in current block is equal to ***fsize/sizeOfSector*** then move the current block to the f and stop. Otherwise insert a new block in the linked list of f having initial sectors of the current block that can accommodate ***fsize*** and update current block of pool accordingly. Also updated the numberOfSectorsInPool. Finally write the contents of the file in the array ***disk*** at the allocated blocks of the ***disk*** and return true.

For example sizeOfSector = 512 bytes and we want to store a file F4 having size = 1936 bytes. This means we need $\left\lceil \frac{1936}{512} \right\rceil = 4$ sectors. Also assume that pool in Figure1 is available. So the file will pick entire block (10, 2) as number of sectors in the first block are 2. Next (14,6) has 6 sectors but we need 2 more sectors so we will include a block (14,2) in the file and second block in the pool will be updated to (16,4). Updated pool of available blocks after this saveFile operation and the disk is shown below.

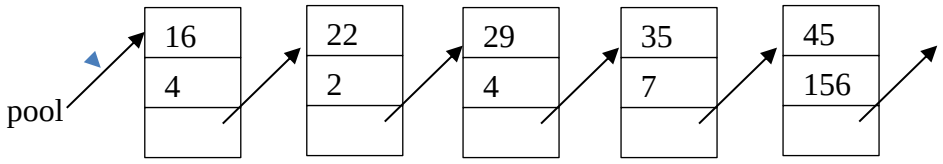


**Figure3: Updated pool of blocks**

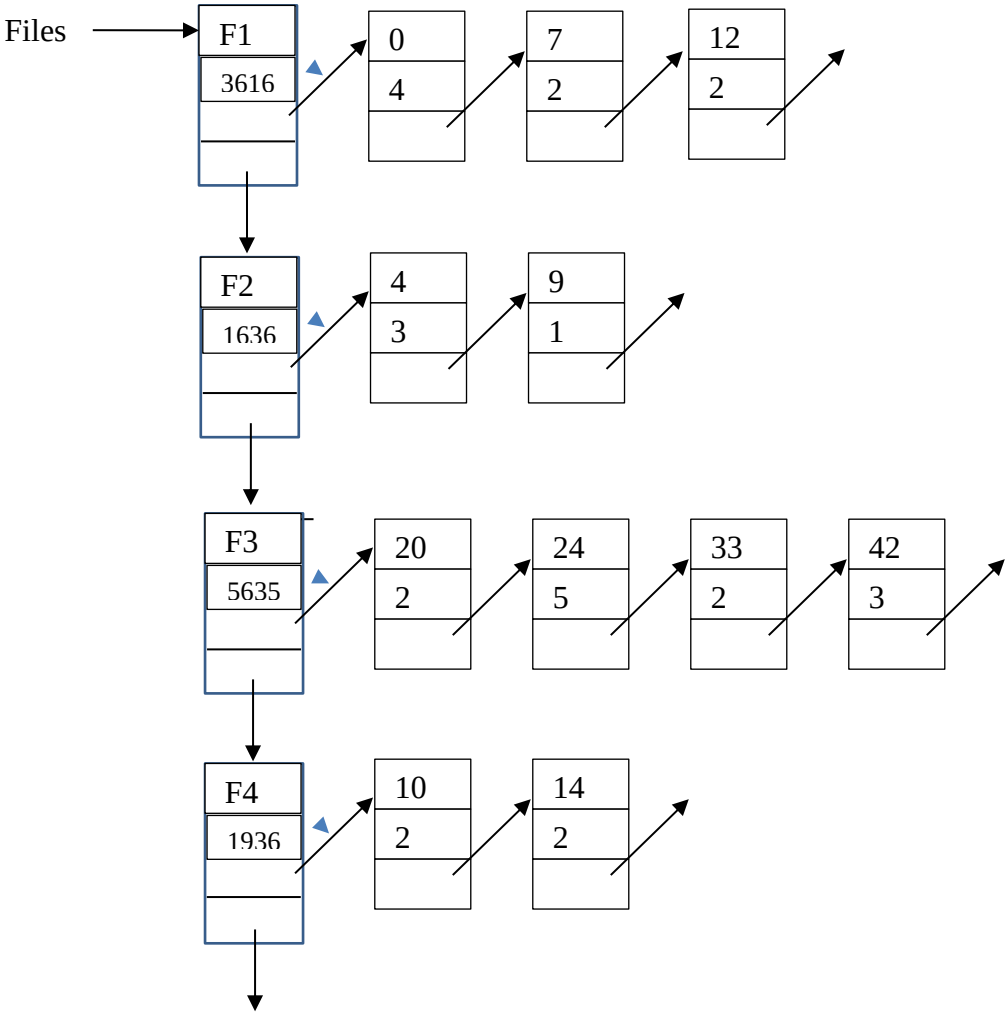


**Figure4: Updated Files list**

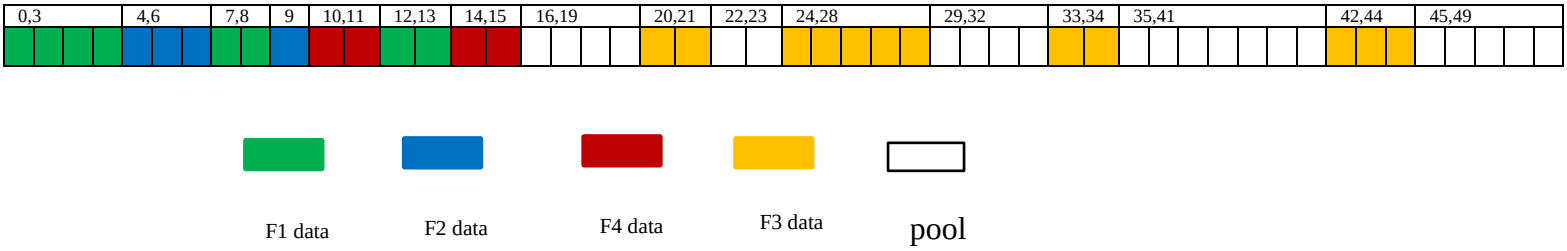


**Figure 5: First 50 sectors of disk**

**DeleteFile:**
This function must take file name ***fname*** as parameter and delete the file with name = ***fname*** from Files. Also move all the blocks of the file to pool in sorted order. If two consecutive block become one single contiguous block then you must also merge them into one single block. For example if file F4 is deleted then first block (10, 2) should be moved to first position in the pool and block (14, 2) should be merged with the block (16, 4) into a single block (14, 6). Finally update the numberOfSectorsInPool accordingly.

**ReadFile**:
Takes ***fname*** as parameter, find the file with name = ***fname***, if such file exists then output its content otherwise print an error message.

**Constructor**:
Initially Files is empty and pool has only one block having sectors 1 to numOfSectors (this means there is no file, entire disk is empty). numOfSectors and sizeOfSectors will be initialized to user provided values

**Destructor**: Delete array disk

Provide a driver function that creates a fileSystem object and a menu that prompts the user to perform different operations of the fileSystem

**VERY IMPORTANT**

- Academic integrity is expected of all the students. Plagiarism or cheating in any assessment will result in negative marking or an **F** grade in the course, and possibly more severe penalties.
- Please make sure that your implementation of this task is efficient and follow all the principals of object oriented paradigm.