CS 201 DATA STRUCTURES
ASSIGNMENT 2
SECTION A&B
Fall 2017

DUE: Saturday Sept 16, 2017

NOTE: Late submissions will not be accepted

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

PROBLEM 1

The hard disk in a computer is divided into sectors where each sector has fixed number of bytes. Each sector has a unique ID. A file system manages the sectors on the hard disk to store files on the disk. A file on hard disk can be stored in multiple sectors if size of the file is larger than the size of the sector. The FAT(File Allocation Table) file system maintains a poo l of available sectors and 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. While storing a file, there may be a case that desired number of contiguous sectors are not available in the pool. In that case, file is stored in a chain of blocks. 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 ending sector. So a file 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.

Write a program to simulate FAT file system. Define the disk as a one dimensional array ***disk*** 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_num, and end_sector_num. Files are kept in a linked list ***files.*** 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 only requires removing the list of blocks corresponding with this file and transferring the sectors assigned to this file back to pool. No changes are made in disk.

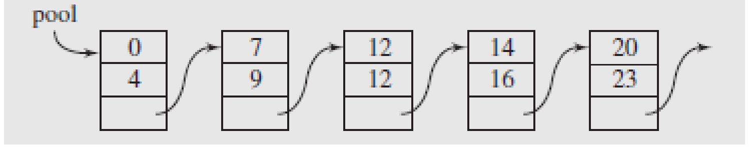

Figure1: pool of blocks

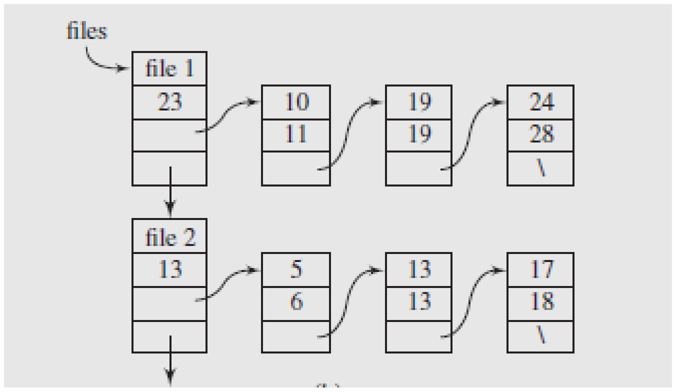

Figure2: files list containing two files file1 and file2

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

Class template of singly linked list
Data member:

- head

Member functions:

- constructor: initializes an empty list
- destructor: destroy all the nodes of the list
- begin: return a listIterator at the start of the list
- end: return a listIterator that points to nullptr
- insert: takes listIterator it and data d as parameter and insert new node with data d after node pointed by listIterator it
- remove: takes listIterator it as parameter, remove the node that is next to the node pointed by it and return the data of removed node
- insertAtStart: takes data d as input and insert a new node at the start of the list
- removeFromStart: remove the first node of the list and return its data
- isEmpty: returns true if list is empty and false otherwise

Class template of listIterator
Data member:

- current

Member functions:

- constructor: default initializes current to nullptr and overloded initialize current to the value of parameter
- operator++: assign current to the next node if one exists
- operator*: returns the data of current node
- operator!= return true if both iterators are different and false otherwise

Class of block
Data member:

- startSector
- endSector

Class of file

Data members:

- name
- size
- Link list of blocks

Class FileSystem
Data members:

- pool (Linked list of block)
- files (Linked list of files)
- disk (dynamic array of characters)
- numOfSectors
- sizeOfSectors

Member functions:
SaveFile: The SaveFile function must take file name ***fname***, file content ***fcontent*** 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 start of ***files***, make assignments: ***f.size = fsize, f.name = fname***. Then traverse the pool using an object of listIterator starting with the first block of pool and check if the number of sectors in current block is less than the ***fsize*** then allocate that block to ***f***, remove it from the pool, update ***fsize*** and move to the next block. If number of sectors in current block is equal to ***fsize*** then allocate current block to the file remove it from pool and stop. Otherwise insert a new block in the linked list of file having initial sectors of the current block that can accommodate ***fsize*** and update current block accordingly. Finally write the contents of the file in the array ***disk*** at the allocated blocks of the ***disk*** and return true.

For example sizeOfSectors = 512 bytes and we want to store a file having size = 3616 bytes. This means we need $\left\lceil \frac{3616}{512} \right\rceil = 8$ sectors. Also assume that pool of available blocks is (1,4), (12, 19), (23, 25), (27, 29), (32,35), (39, 41). So the file will pick entire block(1,4) as number of sectors in the first block are 4. Next (12,19) has 8 sectors but we need 4 more sectors so we will include a block (12,15) in the file and second block in the pool will be updated to (16,19). Updated pool of available blocks after this saveFile operation will be (16, 19), (23, 25), (27, 29), (32,35), (39, 41)

DeleteFile:
This function must take file name ***fname*** as parameter and delete that file from with name = ***fname*** from files. Also move all the blocks of the file to pool

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

Note: You can add other utility functions in the above mentioned classes but make sure that you implementation is efficient

PROBLEM 2

Suppose you want to implement the game pass the bucket. There are n people sitting in form of a circle numbered from 1 to n. There will be n-1 phases of this game. In the first phase, starting at person 1 a basket is passed to the next person. After m passes, the person holding the basket is removed from the circle. In each subsequent phase, the game continues with the person who is sitting next to the dropped person. The game ends when there is only one person left. The last person is the winner of this game. If n= 5 and m=1 then players will be removed in the order: 2, 4, 1,and 5. Player 3 will be the winner.

Implement circular singly list to design this game. Initially insert n(user defined) number of nodes in the list. For each phase of the game generate a random number between 1 and 10 and consider that random number as m. After n-1 phases output the number of the winner.

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.