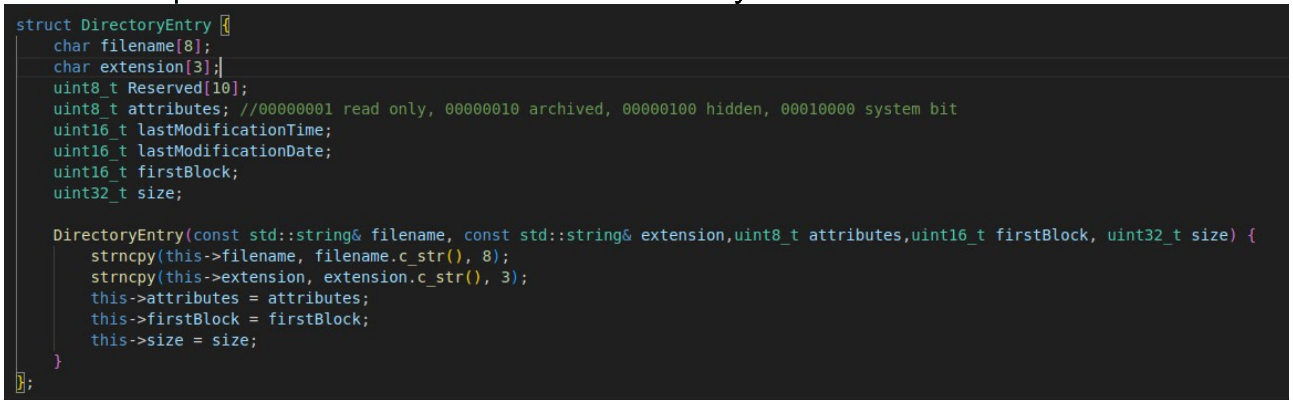
CSE 312/504 HW3 Report

Firstly define directory entries.

Filename represents name of our file. It’s size is 8 byte.



```
struct DirectoryEntry {
    char filename[8];
    char extension[3];
    uint8_t Reserved[10];
    uint8_t attributes; //00000001 read only, 00000010 archived, 00000100 hidden, 00010000 system bit
    uint16_t lastModificationTime;
    uint16_t lastModificationDate;
    uint16_t firstBlock;
    uint32_t size;

    DirectoryEntry(const std::string& filename, const std::string& extension,uint8_t attributes,uint16_t firstBlock, uint32_t size) {
        strncpy(this->filename, filename.c_str(), 8);
        strncpy(this->extension, extension.c_str(), 3);
        this->attributes = attributes;
        this->firstBlock = firstBlock;
        this->size = size;
    }
};
```

Extension is type of file. For example if our files name is alp.txt, alp is the filename and txt is type of our file. So txt becames extension. It’s size is 3 byte.

Attributes contains bits to indicate that a file is read-only, needs to be archived, is hidden, or is a system file.If it’s read only, it’s can not be writted. If it’s archieved, When a file is modified, the file system sets the "archived" attribute to indicate that the file should be included in the backup. This attribute serves as a flag for backup software or other backup mechanisms to identify which files need to be backed up.Once the file is successfully backed up or archived, the "archived" attribute is typically cleared, indicating that the file has been included in the backup and is up to date.This depends on operating system and file system.The hidden bit make file invisible to directory listings.And system bit also hides files.System files cannot be deleted by del command.It’s size is 1 byte. Like writed in comment, 00000001 read only, 00000010 archived, 00000100 hidden, 00010000 system bit.

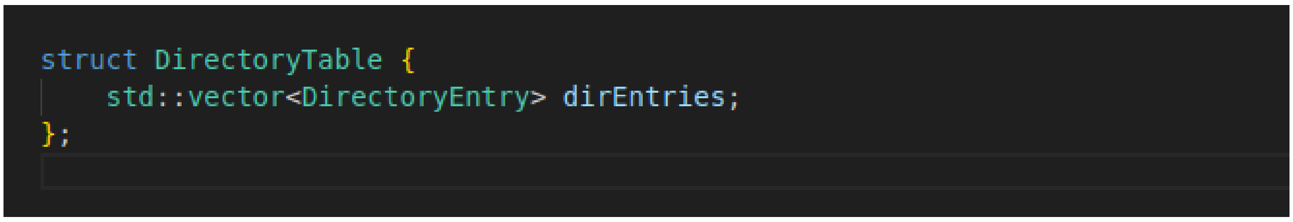
lastModificationTime keeps time of the last modify of file. It’s 5 bits represents seconds, 6 bit represents minutes and 5 bits represents hours.It’s size is 2 byte.

LastModificationDate keeps date of the last modify of the file. It’s 5 bits represents day, 4 bits represents month and 7 bits represents year.It’s size is 2 byte.

firstBlock keeps information about starting point of a file's data within the file system's block or allocation scheme.By following the chain, all the blocks can be found..It’s size is 2 byte.

Size keeps size of the file.It’s size is 4 byte.

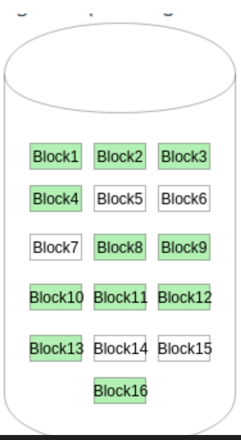
Secondly define directory entries table:

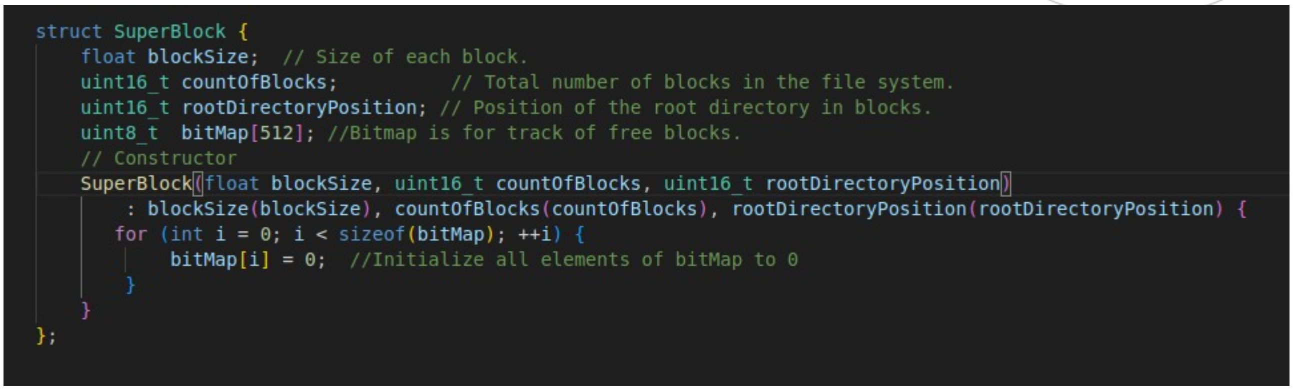


```
struct DirectoryTable {
    std::vector<DirectoryEntry> dirEntries;
};
```

It keeps every directory entry in directory table structure vector.

For keeping track of free blocks, I want to use bitmaps.A disk with n blocks requires a bitmap with n bits. In my bitmap, free blocks are represented by 1s in and allocated blocks by 0s. If we think that green one’s are allocated blocks, our bit map became 0000111000000110.





```
struct SuperBlock {
    float blockSize;  // Size of each block.
    uint16_t countOfBlocks;          // Total number of blocks in the file system.
    uint16_t rootDirectoryPosition; // Position of the root directory in blocks.
    uint8_t  bitMap[512]; //Bitmap is for track of free blocks.
    // Constructor
    SuperBlock(float blockSize, uint16_t countOfBlocks, uint16_t rootDirectoryPosition)
        : blockSize(blockSize), countOfBlocks(countOfBlocks), rootDirectoryPosition(rootDirectoryPosition) {
       for (int i = 0; i < sizeof(bitMap); ++i) {
            bitMap[i] = 0;  //Initialize all elements of bitMap to 0
        }
    }
};
```

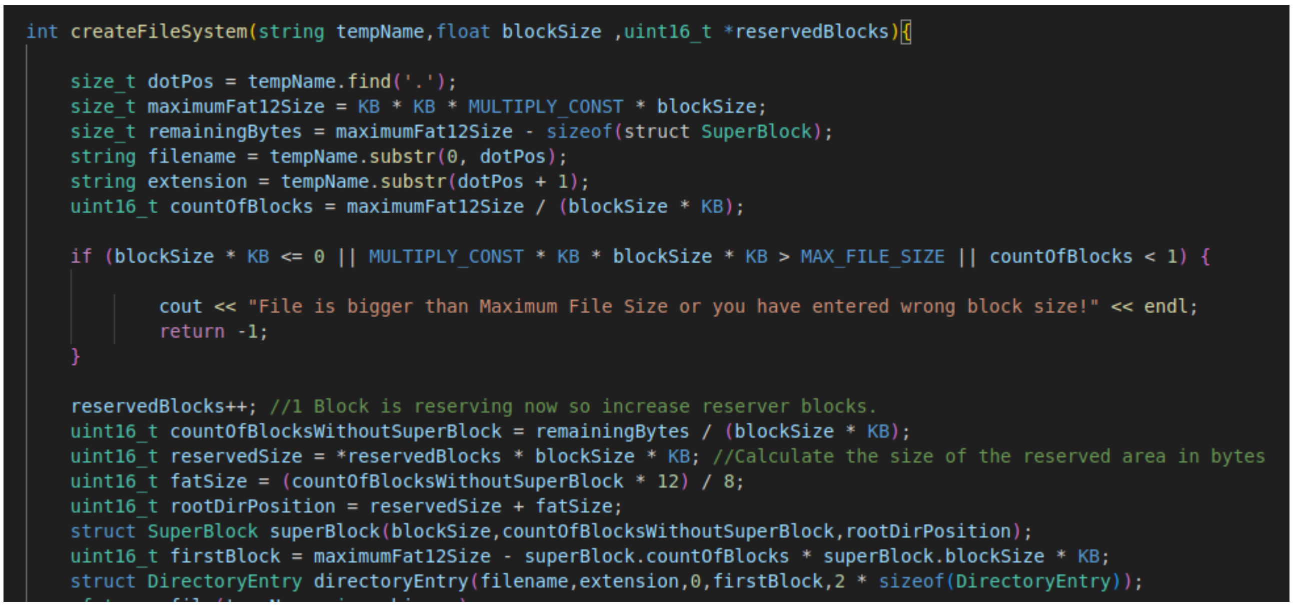
blockSize is the size of blocks

countOfBlocks is count of total blocks in file system.

rootDirectoryPosition is the position of root directory in blocks.

BitMap is for track of free blocks. In fig 4.31 like we can see maximum partition size is 16MB in FAT-12. So we should find bit represent of this to found bit count of bit map. So count of bitMap should be (16 * 1024 * 1024) / (4 * 1024) = 4096 bits. So bitMap should be created this way.

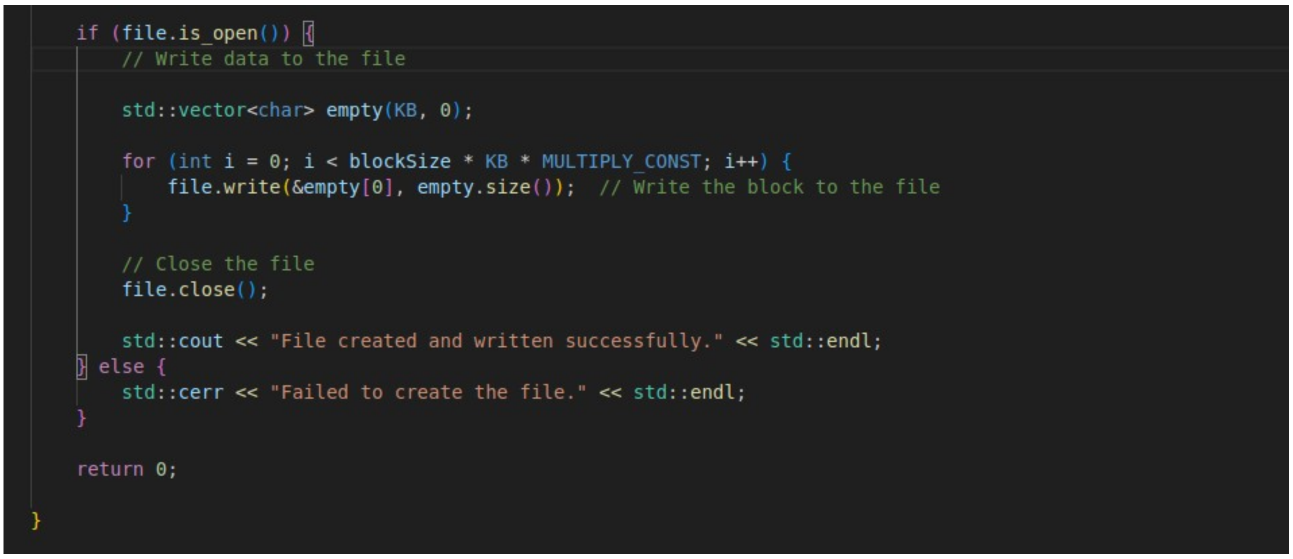
So total bytes of Superblock becames: 4 + 2 + 2 + 512 = 520 bytes.



```
int createFileSystem(string tempName,float blockSize ,uint16_t *reservedBlocks){

    size_t dotPos = tempName.find('.');
    size_t maximumFat12Size = KB * KB * MULTIPLY_CONST * blockSize;
    size_t remainingBytes = maximumFat12Size - sizeof(struct SuperBlock);
    string filename = tempName.substr(0, dotPos);
    string extension = tempName.substr(dotPos + 1);
    uint16_t countOfBlocks = maximumFat12Size / (blockSize * KB);

    if (blockSize * KB <= 0 || MULTIPLY_CONST * KB * blockSize * KB > MAX_FILE_SIZE || countOfBlocks < 1) {

            cout << "File is bigger than Maximum File Size or you have entered wrong block size!" << endl;
            return -1;
    }

    reservedBlocks++; //1 Block is reserving now so increase reserver blocks.
    uint16_t countOfBlocksWithoutSuperBlock = remainingBytes / (blockSize * KB);
    uint16_t reservedSize = *reservedBlocks * blockSize * KB; //Calculate the size of the reserved area in bytes
    uint16_t fatSize = (countOfBlocksWithoutSuperBlock * 12) / 8;
    uint16_t rootDirPosition = reservedSize + fatSize;
    struct SuperBlock superBlock(blockSize,countOfBlocksWithoutSuperBlock,rootDirPosition);
    uint16_t firstBlock = maximumFat12Size - superBlock.countOfBlocks * superBlock.blockSize * KB;
    struct DirectoryEntry directoryEntry(filename,extension,0,firstBlock,2 * sizeof(DirectoryEntry));
```

Firstly I am seperating name of file and extension of file for filling filename and extension fields in DirectoryEntry. Secondly, my blocks contain superblock. So I must decrement sizeof SuperBlock from maximumFat12Size to find remaining size for normal blocks. Then I am founding countOfBlocks with dividing maximumFat12Size (maximum file size for that given block size) to blocksize * KB.

I am controlling if informations creating higher size for maximum limit. If it’s not continuing. Reserved blocks represents using blocks in file system. Then I am founding blockCount without superblock.Reserved size represents total reserved area in file system.
In FAT12 systems, Clusters are represented with 16 bits which means 16/8 = 1.5 Bytes. So for finding fat size, I multiplied my countOfBlocksWithoutSuperBlock with 1.5. That gives me fatsize. Adding reserverSize to fatSize, I am founding rootDirPosition.Then I am filling my superBlock with this entries. Every blocks in bitmap 0 in the startup because there is no allocated blocks now. Then I am calculating first block.With this infos, I am filling my directoryEntry (file).



```cpp
    if (file.is_open()) {
        // Write data to the file

        std::vector<char> empty(KB, 0);

        for (int i = 0; i < blockSize * KB * MULTIPLY_CONST; i++) {
            file.write(&empty[0], empty.size());  // Write the block to the file
        }

        // Close the file
        file.close();

        std::cout << "File created and written successfully." << std::endl;
    } else {
        std::cerr << "Failed to create the file." << std::endl;
    }

    return 0;

}
```

Then I am writing a file with this informations. It gives me the size for blocksize and FAT12 table.(Figure 4.31);