يعتمد FAT هو نظام ملفات يُستخدم لتنظيم وتتبع الملفات على وسائط التخزين مثل الأقراص الصلبة أو محركات USB. التي تُخزن فيها بيانات الملفات. النقاط الأساسية (Clusters) يحتوي على إدخالات تُشير إلى الكتل (FAT) على جدول لفهم عمله:

1. **الكتل (Clusters):** القرص مقسم إلى وحدات صغيرة تُسمى الكتل، وكل كتلة لها حجم محدد (مثل 4 KB أو 8 KB حسب نظام الملفات).
2. **جدول FAT:** يحتوي على إدخالات تُشير إلى حالة كل كتلة:
    - إذا كانت الكتلة تحتوي على جزء من ملف، فإن الإدخال يُشير إلى الكتلة التالية في تسلسل الملف.
    - إذا كانت الكتلة الأخيرة في الملف، يتم وضع علامة خاصة (مثل 0xFFFF في FAT16 أو 0xFFFFFFFF في FAT32).
    - إذا كانت الكتلة فارغة، تُعطى قيمة مثل 0x0000.
3. **الدليل الجذر (Root Directory):** يحتوي على معلومات الملفات مثل الاسم، الحجم، وموقع الكتلة الأولى في جدول FAT.
4. **أنواع FAT:** تختلف حسب عدد البتات لكل إدخال (FAT12 ،FAT16 ،FAT32)، مما يحدد الحد الأقصى لسعة التخزين.

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <vector>
#include <string>

using namespace std;

// Structure to store Boot Sector information
struct BootSector {
    uint16_t bytesPerSector;    // Bytes per sector
    uint8_t sectorsPerCluster;  // Sectors per cluster
    uint16_t reservedSectors;   // Reserved sectors
    uint8_t fatCopies;          // Number of FAT copies
    uint16_t rootEntries;       // Number of root directory entries
```

```
    uint32_t totalSectors;      // Total sectors
    uint16_t sectorsPerFat;     // Sectors per FAT
};

// Structure to store Root Directory entry
struct DirectoryEntry {
    char filename[8];           // File name (8 characters)
    char extension[3];          // File extension (3 characters)
    uint8_t attributes;         // Attributes (file, directory, etc.)
    uint16_t firstCluster;      // First cluster
    uint32_t fileSize;          // File size
};

// Function to read Boot Sector
void readBootSector(ifstream& disk, BootSector& bs) {
    disk.seekg(0x0B); // Move to bytesPerSector location
    disk.read((char*)&bs.bytesPerSector, 2);
    disk.read((char*)&bs.sectorsPerCluster, 1);
    disk.read((char*)&bs.reservedSectors, 2);
    disk.read((char*)&bs.fatCopies, 1);
    disk.read((char*)&bs.rootEntries, 2);
    disk.seekg(0x16); // Move to sectorsPerFat location
    disk.read((char*)&bs.sectorsPerFat, 2);
    disk.seekg(0x11); // Move to totalSectors location
    uint16_t sectors16;
    disk.read((char*)&sectors16, 2);
    if (sectors16 == 0) {
```

```
        disk.seekg(0x20);
        disk.read((char*)&bs.totalSectors, 4);
    } else {
        bs.totalSectors = sectors16;
    }
}

// Function to read a Root Directory entry
void readDirectoryEntry(ifstream& disk, DirectoryEntry& entry) {
    disk.read(entry.filename, 8);
    disk.read(entry.extension, 3);
    disk.read((char*)&entry.attributes, 1);
    disk.seekg(disk.tellg() + 14); // Skip unused fields
    disk.read((char*)&entry.firstCluster, 2);
    disk.read((char*)&entry.fileSize, 4);
}

// Function to read the cluster chain from FAT
vector<uint16_t> getClusterChain(ifstream& disk, uint16_t firstCluster, const
BootSector& bs) {
    vector<uint16_t> chain;
    uint16_t currentCluster = firstCluster;
    const int fatStart = bs.reservedSectors * bs.bytesPerSector; // FAT starting position

    while (currentCluster < 0xFFF8) { // 0xFFF8 or higher marks the end in FAT16
        chain.push_back(currentCluster);
        disk.seekg(fatStart + currentCluster * 2); // Each FAT16 entry is 2 bytes
```

```
        disk.read((char*)&currentCluster, 2);
    }
    return chain;
}

int main() {
    ifstream disk("disk.img", ios::binary);
    if (!disk.is_open()) {
        cout << "Failed to open disk image!" << endl;
        return 1;
    }

    // Read Boot Sector
    BootSector bs;
    readBootSector(disk, bs);

    // Calculate Root Directory location
    int fatSize = bs.sectorsPerFat * bs.bytesPerSector;
    int rootStart = bs.reservedSectors * bs.bytesPerSector + bs.fatCopies * fatSize;
    int rootSize = bs.rootEntries * 32; // Each entry is 32 bytes

    // Move to Root Directory
    disk.seekg(rootStart);

    // Read Root Directory entries
    cout << "Files in Root Directory:" << endl;
    for (int i = 0; i < bs.rootEntries; i++) {
```

```
        DirectoryEntry entry;
        readDirectoryEntry(disk, entry);

        // Skip empty or deleted entries
        if (entry.filename[0] == 0x00 || entry.filename[0] == (char)0xE5) continue;

        // Display file information
        string name(entry.filename, 8);
        string ext(entry.extension, 3);
        cout << "File name: " << name + "." + ext << endl;
        cout << "Size: " << entry.fileSize << " bytes" << endl;
        cout << "First cluster: " << entry.firstCluster << endl;

        // Read cluster chain
        vector<uint16_t> chain = getClusterChain(disk, entry.firstCluster, bs);
        cout << "Cluster chain: ";
        for (uint16_t cluster : chain) {
            cout << cluster << " ";
        }
        cout << endl << "----------------" << endl;
    }

    disk.close();
    return 0;
}
```