`ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

Project 01:

Quản lý hệ thống tập tin trên Windows

Môn: Hệ điều hành

Giảng viên hướng dẫn: ThS. Lê Viết Long

Thành viên nhóm:

21120279 – Lê Trần Minh Khuê

21120289 – Diệp Quốc Hoàng Nam

21120290 – Hoàng Trung Nam

21120348 – Lương Thanh Tú

21120354 – Nguyễn Trần Trình

Thành phố Hồ Chí Minh, tháng 11 / 2023

LỜI CẢM ƠN

Chúng em xin gửi lời cảm ơn chân thành đến Khoa Công nghệ Thông tin, Trường Đại học Khoa học Tự nhiên Thành phố Hồ Chí Minh đã tạo điều kiện thuận lợi cho chúng em học tập và hoàn thành bài đồ án này. Đặc biệt, chúng em xin bày tỏ lòng biết ơn sâu sắc đến ThS. Lê Viết Long, người thầy đã dày công truyền đạt kiến thức và hướng dẫn chúng em trong quá trình làm bài.

Chúng em đã cố gắng vận dụng những kiến thức đã học được để hoàn thành bài đồ án. Nhưng do kiến thức hạn chế và không có nhiều kinh nghiệm thực tiễn nên khó tránh khỏi những thiếu sót trong quá trình nghiên cứu và trình bày. Rất kính mong sự góp ý của quý thầy để bài báo cáo của chúng em được hoàn thiện hơn.

Một lần nữa, chúng em xin trân trọng cảm ơn sự quan tâm giúp đỡ của thầy trong suốt quá trình thực hiện đồ án này.

Xin trân trọng cảm ơn!

MỤC LỤC

DANH MỤC HÌNH

DANH MỤC BẢNG

DANH MỤC MÃ NGUỒN

1. TỔNG QUAN

1.1. THÔNG TIN NHÓM

STT	MSSV	Họ và Tên	Email
1	21120279	Lê Trần Minh Khuê	21120279@student.hcmus.edu.vn
2	21120289	Diệp Quốc Hoàng Nam	21120289@student.hcmus.edu.vn
3	21120290	Hoàng Trung Nam	21120290@student.hcmus.edu.vn
4	21120348	Lương Thanh Tú	21120348@student.hcmus.edu.vn
5	21120354	Nguyễn Trần Trình	21120354@student.hcmus.edu.vn

Bảng 1. Danh sách thành viên

1.2. BẢNG PHÂN CÔNG CÔNG VIỆC

MSSV	Họ và Tên	Mô tả công việc
21120279	Lê Trần Minh Khuê	Kiểm thử, chính sửa code và viết báo cáo cho phần FAT32, định dạng và chỉnh sửa toàn báo cáo.
21120289	Diệp Quốc Hoàng Nam	Viết và kiểm thử chương trình phần NTFS. Viết báo cáo NTFS.
21120290	Hoàng Trung Nam	Viết và kiểm thử chương trình phần NTFS. Viết báo cáo NTFS.
21120348	Lương Thanh Tú	Viết chương trình và báo cáo phần hiển thị cây thư mục và đọc tập tin .txt của FAT32.
21120354	Nguyễn Trần Trình	Viết và kiểm thử chương trình phần FAT32. Viết báo cáo đọc Boot Sector FAT32.

Bảng 2. Phân công công việc

1.3. CÁC NỘI DUNG ĐÃ HOÀN THÀNH

Yêu cầu	Hệ thống	Công việc	Tỉ lệ hoàn thành
1	FAT32	Đọc các thông tin mô tả trong Boot Sector	100%
	NTFS	Đọc các thông tin mô tả trong Partition Boot Sector	100%
2	FAT32	Hiển thị thông tin cây thư mục của phân vùng và đọc tập tin .txt.	100%

	NTFS	Hiển thị thông tin cây thư mục của phân vùng và đọc tập tin .txt.	90%

Bảng 3. Mức độ hoàn thành

1.4. TỔ CHỨC BÀI LÀM VÀ CÁC GHI CHÚ LẬP TRÌNH

1.4.1. Tổ chức bài làm

Chúng em bố trí bài làm thành các file như sau:

- Source: chứa source code chương trình
    - FAT32: thư mục chứa source code phần FAT32
        - FAT32.h: Chứa phần định nghĩa các cấu trúc và các hàm cần thiết.
        - FAT32.cpp: Chứa mã nguồn của các hàm.
        - main.cpp: chứa hàm main, có các phần gọi thực thi các hàm.
    - NTFS:
        - main.cpp: chứa hàm main, có các phần gọi thực thi các hàm.
        - Master.cpp: chứa mã nguồn các hàm chung cho đọc đĩa
        - NTFSModule.cpp: chứa mã nguồn hàm của NTFS
        - NTFSModule.h: chứa phần định nghĩa các hàm và cấu trúc của NTFS
        - MFTEntry.cpp: chứa mã nguồn các hàm liên quan MFT
        - MFTEntry.h: chứa phần định nghĩa các hàm và cấu trúc liên quan MFT
        - Master.h: chứa phần định nghĩa các hàm cần thiết cho đọc đĩa
        - MFTutil.cpp: chứa mã nguồn hàm đọc filename từ MFT
        - MFTutil.h: chứa phần định nghĩa hàm đọc filename từ một MFT entry.
    - Release: chứa các file thực thi
- Report: Chứa file báo cáo (Report.pdf – file này).

1.4.2. Môi trường thực nghiệm:

Các thuật toán được chạy và đo các số liệu trên thiết bị có cấu hình như sau:

- CPU: 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz 1.69 GHz
- RAM: 16.0 GB
- Hệ điều hành: Windows 11 Home Single Language (64-bit, Version 22H2).

Chương trình được cài đặt và biên dịch chủ yếu trên IDE Visual Studio version 2022.

2. NỘI DUNG BÁO CÁO

2.1. HỆ THỐNG TẬP TIN FAT32

2.1.1. Đọc các thông tin mô tả trong Boot Sector

2.1.1.1. Các thông tin sẽ đọc bao gồm:

Name	Offset Hex	Size (bytes)	Description	Ký hiệu
BPB_BytsPerSec	B	2	Số bytes/sector	
BPB_SecPerClus	D	1	Số sectors/cluster	S_C
BPB_RsvdSecCnt	E	2	Số sector để dành (khác 0) (Số sector trước bảng FAT)	S_B
BPB_NumFATs	10	1	Số bảng FAT	N_F
BPB_RootEntCnt	11	2	FAT12, FAT16: số entry trong bảng RDET FAT32: có giá trị là 0	N_{RDET}
BPB_TotSec32	20	4	Số sector trong Volume. Nếu bằng 0, BPB_TotSec16 phải khác 0	N_V
BPB_FATSz32	24	4	số sector trong 1 bảng FAT (BPB_FATSz16 must be 0.)	S_f
BPB_RootClus	2C	4	Chỉ số cluster đầu tiên của RDET (thông thường: 2)	
BPB_FSInfo	30	2	Additional Information Sector	
BS_FilSysType	52	8	Chuỗi nhận diện loại FAT: ”FAT32 ”.	

Bảng 4. Cấu trúc các trường thông tin cần đọc trong BootSector

2.1.1.2. Định nghĩa cấu trúc FAT32_BOOTSECTOR:

```
typedef struct
{
    uint8_t BS_jmpBoot[3];              // 0x00: Jump instruction to start executing code
    uint8_t BS_OEMName[8];              // 0x03: OEM manufacturer name
    uint8_t BPB_BytsPerSec[2];          // 0x0B: Bytes per sector
    uint8_t BPB_SecPerClus;             // 0x0D: Sectors per cluster
    uint8_t BPB_RsvdSecCnt[2];          // 0x0E: Reserved sector count
    uint8_t BPB_NumFATs;                // 0x10: Number of File Allocation Tables (FATs)
    uint8_t BPB_RootEntCnt[2];          // 0x11: Number of root directory entries
    uint8_t BPB_TotSec16[2];           // 0x13: Total sectors (16-bit) - for FAT16
    uint8_t BPB_Media;                  // 0x15: Media type (e.g., hard disk)
    uint8_t BPB_FATSz16[2];             // 0x16: FAT size (16-bit) - for FAT16
    uint8_t BPB_SecPerTrk[2];           // 0x18: Sectors per track
    uint8_t BPB_NumHeads[2];            // 0x1A: Number of heads
    uint8_t BPB_HiddenSec[4];           // 0x1C: Hidden sectors before the boot sector
    uint8_t BPB_TotSec32[4];            // 0x20: Total sectors (32-bit) - for FAT32
    uint8_t BPB_FATSz32[4];             // 0x24: FAT size (32-bit) - for FAT32
    uint8_t BPB_ExtFlags[2];            // 0x28: Extended flags
    uint8_t BPB_FSVer[2];               // 0x2A: File system version
    uint8_t BPB_RootClus[4];            // 0x2C: First cluster of root directory
    uint8_t BPB_FSInfo[2];              // 0x30: FSInfo sector
    uint8_t BPB_BkBootSec[2];           // 0x32: Backup boot sector
    uint8_t BPB_Reserved[12];           // 0x34: Reserved bytes
    uint8_t BS_DrvNum;                  // 0x40: Drive number
    uint8_t BS_Reserved1;               // 0x41: Reserved byte
    uint8_t BS_BootSig;                 // 0x42: Boot signature
    uint8_t BS_VolID[4];                // 0x43: Volume ID
    uint8_t BS_VolLab[11];              // 0x47: Volume label
    uint8_t BS_FilSysType[8];           // 0x52: File system type (usually "FAT32")
    uint8_t reservedCode[420];          // 0x5A: Reserved bytes
    uint8_t endSignal[2];               // 0x1FE: End of the boot sector
} FAT32_BOOTSECTOR, * PFAT32_BOOTSECTOR;
```

Code 1: Struct FAT32_BOOTSECTOR

2.1.1.3. Các hàm hỗ trợ chuyển đổi giữa các hệ cơ số:

```
// Converse 8-bit Integers into one 32-bit Integer
uint32_t bytesToInt(const uint8_t bytes[], int n) {
    uint32_t res = 0;
    for (int i = n - 1; i >= 0; i--) {
        res = (res << 8) | bytes[i];
    }
    return res;
}
```

Code 2:Hàm chuyển từ kiểu 8-bit Integers sang 32-bit Integers

```
// Converse 8-bit Integers into one string
wstring bytesToWString(const uint8_t bytes[], int n) {
    wstring result = L"";

    for (int i = 0; i < n - 1; i += 2) {
        if ((bytes[i] == 0 && bytes[i + 1] == 0) || (bytes[i] == 0xff && bytes[i + 1] == 0xff)) {
            break;
        }

        result += wchar_t(bytes[i]);
    }

    return result;
}
```

Code 3: Hàm chuyển 8-bit Integers sang string

```
// Converse 8-bit Integer into Hex String
wstring bytesToHex(BYTE byte) {
    wstring s = L"";
    int sum = (int)byte;

    s = _HEX_CHAR[sum / 16];
    s = s + _HEX_CHAR[sum % 16];

    return s;
}
```

Code 4: Hàm chuyển 8-bit Integers sang Hex String

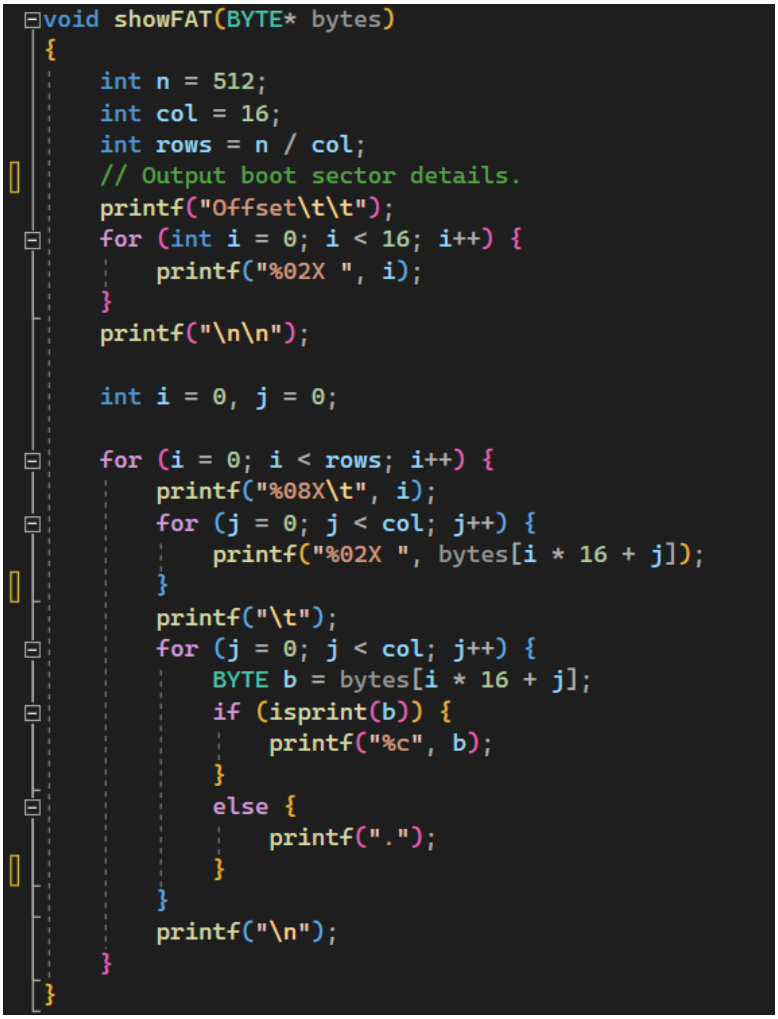
2.1.1.4. Hàm đọc Sector đã được cung cấp:

```
// Read sector and save into buffer
int readSector(LPCWSTR drive, int readPoint, BYTE sector[512])
{
    DWORD bytesRead;
    HANDLE device = NULL;

    device = CreateFile(drive,    // Drive to open
        GENERIC_READ,           // Access mode
        FILE_SHARE_READ | FILE_SHARE_WRITE,        // Share Mode
        NULL,                   // Security Descriptor
        OPEN_EXISTING,          // How to create
        0,                      // File attributes
        NULL);                  // Handle to template

    if (device == INVALID_HANDLE_VALUE) // Open Error
    {
        return 1;
    }

    SetFilePointer(device, readPoint, NULL, FILE_BEGIN); //Set a Point to Read

    // Read here
    if (!ReadFile(device, sector, 512, &bytesRead, NULL)) // Read Error
    {
        CloseHandle(device);
        return 2;
    }
    else
    {
        CloseHandle(device);
        return 0;
    }
}
```

Code 5: Hàm readSector

2.1.1.5. Hàm hiển thị BootSector:

- **Input:** BYTE* bytes – các bytes dữ liệu đã được đọc từ hàm readSector.
- **Output:** In ra BootSector vừa đọc được.



```
void showFAT(BYTE* bytes)
{
    int n = 512;
    int col = 16;
    int rows = n / col;
    // Output boot sector details.
    printf("Offset\t\t");
    for (int i = 0; i < 16; i++) {
        printf("%02X ", i);
    }
    printf("\n\n");

    int i = 0, j = 0;

    for (i = 0; i < rows; i++) {
        printf("%08X\t", i);
        for (j = 0; j < col; j++) {
            printf("%02X ", bytes[i * 16 + j]);
        }
        printf("\t");
        for (j = 0; j < col; j++) {
            BYTE b = bytes[i * 16 + j];
            if (isprint(b)) {
                printf("%c", b);
            }
            else {
                printf(".");
            }
        }
        printf("\n");
    }
}
```

Code 6: Hàm showFAT - in BootSector

2.1.1.6. Hàm đọc các thông tin mô tả BootSector:

- **Input:** dữ liệu lưu trong cấu trúc FAT32_BOOTSECTOR.
- **Output:** In ra các thông tin mô tả bao gồm:
    - File system type
    - Bytes per sectorA
    - Sector per cluster (Sc)
    - Sector in Boot Sector (Sb)
    - Number of FATs (nF)
    - Number of root directory entries (Sr)
    - Size of volume (Sv)
    - Size of FAT (Sf)
    - Start cluster RDET

- Additional Information Sector
- Boot Sector's Backup Sectors

- **Xử lý:**
  - File system type là kiểu ký tự, nên ta xử lý gán biến này về dạng char trước khi in ra.
  - Các thông tin có kích thước >1 byte được chuyển về int bằng hàm bytesToInt() trước khi in ra.
  - Các thông tin có kích thước = 1 byte được thực hiện ép kiểu (int).

```
// Print Boot Sector Info
void printBootSectorInfo(const FAT32_BOOTSECTOR& boot_sector) {

    char fileSystemType[9];
    memcpy(fileSystemType, boot_sector.BS_FilSysType, 8);
    fileSystemType[8] = '\0';

    wcout << L"FAT32 Boot Sector Basic Info:\n";
    wcout << L"File system type                    :";
    wcout << setw(13) << fileSystemType << endl;

    wcout << L"Bytes per sector                    :";
    wcout << setw(10) << bytesToInt(boot_sector.BPB_BytsPerSec,2) << endl;

    wcout << L"Sector per cluster (Sc)             :";
    wcout << setw(10) << (int)boot_sector.BPB_SecPerClus << " sector(s)" << endl;

    wcout << L"Sector in Boot Sector (Sb)          :";
    wcout << setw(10) << bytesToInt(boot_sector.BPB_RsvdSecCnt, 2) << " sector(s)" << endl;

    wcout << L"Number of FATs (nF)                 :";
    wcout << setw(10) << (int)boot_sector.BPB_NumFATs << " FAT(s)" << endl;

    wcout << L"Number of root directory entries (Sr) :";
    wcout << setw(10) << bytesToInt(boot_sector.BPB_RootEntCnt,2) << " sector(s)" << endl;

    wcout << L"Size of volume (Sv)                 :";
    wcout << setw(10) << bytesToInt(boot_sector.BPB_TotSec32, 4) << " sector(s)" << endl;

    wcout << L"Size of FAT (Sf)                    :";
    wcout << setw(10) << bytesToInt(boot_sector.BPB_FATSz32, 4) << " sector(s)" << endl;

    wcout << L"Start cluster RDET                  :";
    wcout << setw(10) << bytesToInt(boot_sector.BPB_RootClus, 4) << endl;

    wcout << L"Additional Information Sector       :";
    wcout << setw(10) << bytesToInt(boot_sector.BPB_FSInfo, 2) << " sector(s)" << endl;

    wcout << L"Boot Sector's Backup Sectors        :";
    wcout << setw(10) << bytesToInt(boot_sector.BPB_BkBootSec, 2) << " sector(s)" << endl;
    wcout << endl << endl;
}
```

Code 7: Hàm in thông tin mô tả BootSector

2.1.1.7. Quá trình gọi thực thi các hàm:

- Khai báo các biến cần thiết:
    - `BYTE sector[512];` là một mảng 512 byte, được sử dụng để lưu trữ dữ liệu từ phần khởi động của ổ đĩa.
    - `FAT32_BOOTSECTOR boot_sector;` là một biến cấu trúc kiểu `FAT32_BOOTSECTOR`, sẽ được sử dụng để lưu trữ thông tin từ phần khởi động.
- `wchar_t disk_path[] = L"\\\\.\\?:";` Đây là một mảng ký tự được sử dụng để định đường dẫn tới ổ đĩa. Sau khi người dùng nhập ký tự đại diện cho ổ đĩa thì disk_path sẽ là đường dẫn đến ổ đĩa đó.
- Tiến hành `readSector(disk_path, 0, sector) =>` Đọc thông tin BootSector và lưu vào biến `sector` đã khai báo trước đó. Nếu không đọc được, in ra thông báo và kết thúc.
- Ngược lại, gọi hàm `showFAT(sector);` để in BootSector vừa đọc được.
- Sao chép dữ liệu từ mảng sector vào biến boot_sector, sử dụng memcpy. Kích thước sao chép được xác định bằng `sizeof(FAT32_BOOTSECTOR)` do ta chỉ lấy một phần cần thiết để đọc thông tin.
- Gọi hàm `printBootSectorInfo(boot_sector);` để in các thông tin mô tả về BootSector.

```
int main() {

    BYTE sector[512];
    FAT32_BOOTSECTOR boot_sector;

    // Input disk
    wchar_t disk_path[] = L"\\\\.\\?:";
    wcout << L"Disk letter: ";
    wcin >> disk_path[4];

    // Start reading disk
    if (readSector(disk_path, 0, sector)) {
        wcout << L"Cant Read Disk" << endl;
        return 1;
    }

    // Show File Allocation Table
    showFAT(sector);

    //Put bytes in sector buffer into FAT32_BOOTSECTOR struct
    memcpy(&boot_sector, sector, sizeof(FAT32_BOOTSECTOR));
    printBootSectorInfo(boot_sector);
```

Code 8: Cấu trúc chương trình đọc BootSector - FAT32

2.1.2. Hiển thị thông tin cây thư mục của phân vùng và đọc tập tin .txt.

2.1.2.1. Source code hàm main:

```
    uint32_t sb = bytesToInt(boot_sector.BPB_RsvdSecCnt, 2),
        sc = uint32_t(boot_sector.BPB_SecPerClus),
        nf = uint32_t(boot_sector.BPB_NumFATs),
        sf = bytesToInt(boot_sector.BPB_FATSz32, 4);
    uint32_t first_sector_of_data = sb + nf * sf;

    BYTE FAT01[512];
    // read FAT1
    readSector(disk_path, sb * 512, FAT01);
    // Read RDET as well as item property at once
    wcout << "Disk items' simple properies:\n\n";
    read_RDET(disk_path, first_sector_of_data, 0, FAT01, sc, first_sector_of_data);

    // Print folder tree
    // *****UPDATED*******
    wcout << "Folder Tree:\n\t|" << disk_path[4] << ":\n";
    printTree();

    system("pause");

    return 0;
}
```

Code 9: Cấu trúc chương trình hiển thị cây thư mục và đọc tập tin .txt (FAT32)

2.1.2.2. Cấu trúc RDET và SDET và class ITEM

Bắt đầu bằng đọc FAT1 tìm vị trí sector trên FAT1 để đọc RDET. Sau đó bắt đầu đọc RDET và các thuộc tính cùng lúc sử dụng hàm read_RDET.

Với các cấu trúc RDET và SDET được định nghĩa:

```
typedef struct
{
    uint8_t fileName[8];
    uint8_t fileExtention[3];
    uint8_t fileAttribute;
    uint8_t resevered[10];
    uint8_t lastModifiedTime[2];
    uint8_t lastModifiedDate[2];
    uint8_t startCluster[2];
    uint8_t fileSize[4];
}FAT32_MAINENTRY, * PFAT32_MAINENTRY;
```

```
typedef struct
{
    uint8_t seqNum;
    uint8_t fileName1[10];
    uint8_t fileAttribute;
    uint8_t fileType;
    uint8_t fileCheckSum;
    uint8_t fileName2[12];
    uint8_t startCluster[2];
    uint8_t fileName3[4];
}FAT32_SUBENTRY, * PFAT32_SUBENTRY;
```

Code 10: Struct FAT32_MAINENTRY và FAT32_SUBENTRY

Trong quá trình làm, một danh sách item sẽ được lưu lại phục vụ mục đích in ra cây thư mục. Cấu trúc của danh sách này được định nghĩa như sau:

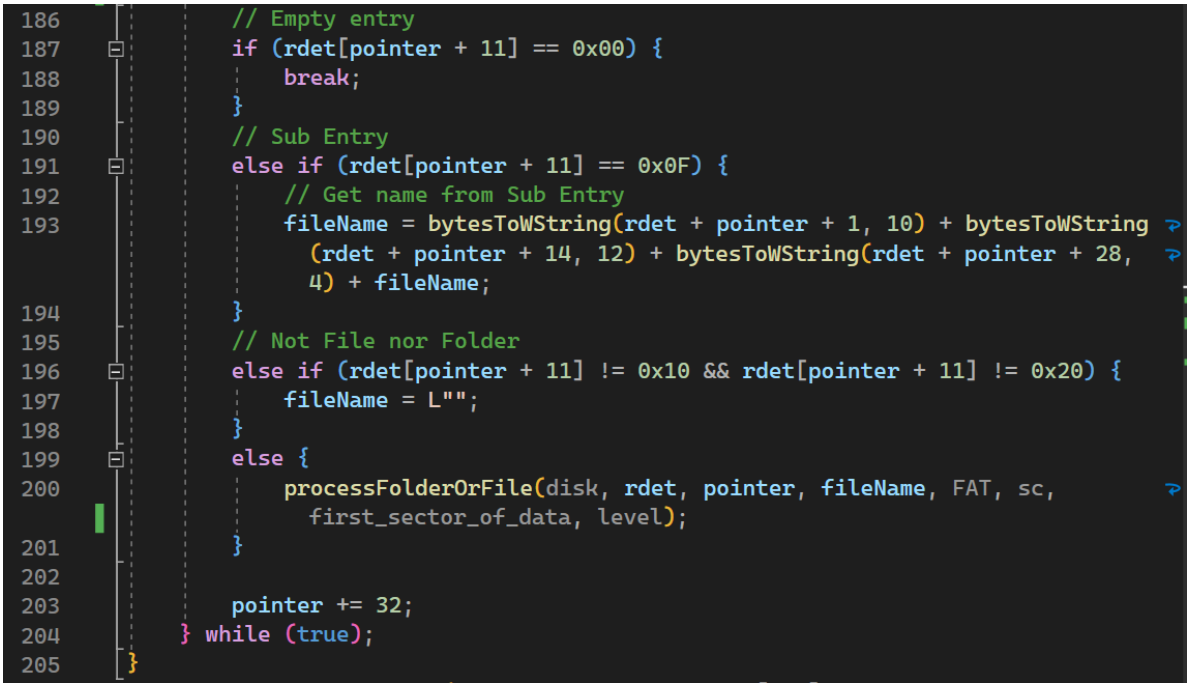
```
class ITEM {
public:
    wstring itemName;
    int itemType;       // 1. Folder, 0. File
    uint32_t itemSize;
    uint32_t firstCluster;
    uint32_t lastCluster;
    int level;
};
```

Code 11: Class ITEM

2.1.2.3. Hàm read_RDET

```
161
162  void read_RDET(LPCWSTR disk, uint32_t sector_index, int level, BYTE FAT[512],
         uint32_t sc, uint32_t first_sector_of_data) {
163      // Load RDET
164      BYTE rdet[512];
165      readSector(disk, sector_index * 512, rdet);
166      wstring fileName = L"";
167
168      // Set pointer to go through sector
169      uint32_t pointer = 0;
170
171      // Skip two first entries -> System
172      pointer = 2 * 32;
173
174      do {
175          // Read another sector
176          if (pointer == 512) {
177              pointer = 0;
178              sector_index += 1;
179              readSector(disk, sector_index * 512, rdet);
180          }
181          // Deleted sector
182          if (rdet[pointer] == 0xE5) {
183              pointer += 32;
184              continue;
185          }
```

Code 12: Hàm read_RDET (01)



```
186         // Empty entry
187         if (rdet[pointer + 11] == 0x00) {
188             break;
189         }
190         // Sub Entry
191         else if (rdet[pointer + 11] == 0x0F) {
192             // Get name from Sub Entry
193             fileName = bytesToWString(rdet + pointer + 1, 10) + bytesToWString
                  (rdet + pointer + 14, 12) + bytesToWString(rdet + pointer + 28,
                  4) + fileName;
194         }
195         // Not File nor Folder
196         else if (rdet[pointer + 11] != 0x10 && rdet[pointer + 11] != 0x20) {
197             fileName = L"";
198         }
199         else {
200             processFolderOrFile(disk, rdet, pointer, fileName, FAT, sc,
                  first_sector_of_data, level);
201         }
202
203         pointer += 32;
204     } while (true);
205 }
```

Code 13: Hàm read_RDET (02)

Cụ thể :

- Bắt đầu đọc RDET, 2 entries đầu tiên là của hệ thống nên sẽ không đọc. (dòng 172)
- Đưa vào vòng lặp để đọc. Trong vòng lặp, đầu tiên ta sẽ kiểm tra nếu con trỏ đọc của ta đang ở cuối sector hiện tại mà vẫn còn dữ liệu ta sẽ tiếp tục đọc sector tiếp theo để lấy hết phần dữ liệu còn lại (dòng 176 – 180).
- Nếu sector đó đã bị xoá thì sẽ bỏ qua sector đó (dòng 182 -185).
- Lệnh if kiểm tra xem đã hết dữ liệu chưa. Nếu hết sẽ thoát chương trình (dòng 186 – 189)
- Lấy tên từ SDET (dòng 190 – 194). Nếu item đó không có thuộc tính là file hoặc folder sẽ bị bỏ qua:
- Ngược lại, ta truyền vào hàm `processFolderOrFile(disk, rdet, pointer, fileName, FAT, sc, first_sector_of_data, level);` để tiếp tục xử lý đến file/ folder đó.

2.1.2.4. Hàm processFolderOrFile

```
206  void processFolderOrFile(LPCWSTR disk, BYTE rdet[512], uint32_t pointer, wstring& fileName,
     BYTE FAT[512], uint32_t sc, uint32_t first_sector_of_data, int level) {
207      FAT32_MAINENTRY mainEnt;
208      memcpy(&mainEnt, rdet + pointer, 32);
209
210      uint32_t first_cluster = bytesToInt(mainEnt.startCluster, 2);
211      uint32_t last_cluster = first_cluster;
212
213      while (true) {
214          uint32_t FATmember = bytesToInt(FAT + last_cluster * 4, 4);
215          if (FATmember == 0x0FFFFFFF || FATmember == 0x0FFFFFF8 || last_cluster == 0) {
216              break;
217          }
218          else if (FATmember == 0x0FFFFFF7 || FATmember == 0) {
219              //wcout << L"    FAT Table deleted or empty.\n";
220              break;
221          }
222          else if (FATmember == 0xCCCCCCCC) {
223              wcout << L"Unable to read FAT Table -> Uathority\nPlease run as administator.";
224              break;
225          }
226          else last_cluster = FATmember;
227      }
```

Code 14: Hàm processFolderOrFile (01)

```
229      // Folder
230      if (mainEnt.fileAttribute == 0x10) {
231          // Main Entry name
232          if (fileName == L"") fileName = bytesToWString(mainEnt.fileName, 8);
233          // Get Item to list -> Easier to mange
234          ITEM item = getNewItem(fileName, 0, (first_cluster - 2) * sc + first_sector_of_data, 1, level);
235          list_item.push_back(item);
236          // Print Item's properties
237          wcout << L" *Item: " << fileName << endl;
238          wcout << L"    Type: Folder" << endl;
239          wcout << L"    First Cluster: " << first_cluster << endl;
240          wcout << L"    Cluster list: ";
241          for (int i = first_cluster; i <= last_cluster; i++) {
242              wcout << i;
243              if (i != last_cluster) wcout << L", ";
244          }
245          wcout << endl << L"    Sector list: ";
246          for (int i = item.firstCluster; i <= (last_cluster - 1) * sc + first_sector_of_data; i++) {
247              wcout << i;
248              if (i != (last_cluster - 1) * sc + first_sector_of_data) cout << ", ";
249          }
250          wcout << endl << endl;
251          fileName = L"";
252
253          // Read Sub Folder
254          if (bytesToInt(rdet + pointer + 28, 4) == 0) {
255              read_RDET(disk, item.firstCluster, level + 1, FAT, sc, first_sector_of_data);
256          }
257      }
```

Code 15: Hàm processFolderOrFile (02)

```
258     // File
259     else if (mainEnt.fileAttribute == 0x20) {
260         // Main Entry Name
261         if (fileName == L"") fileName = bytesToWString(mainEnt.fileName, 8);
262
263         // Get Item to list -> Easier to mange
264         ITEM item = getNewItem(fileName, 0, (first_cluster - 2) * sc +
              first_sector_of_data, 0, level);
265         list_item.push_back(item);
266
267         // Print Item's properties
268         wcout << L" *Item: " << fileName << endl;
269         wcout << L"     Type: File" << endl;
270         wcout << L"     Size: " << bytesToInt(mainEnt.fileSize, 4) << L" B" <<
              endl;
271         wcout << L"     First Cluster: " << first_cluster << endl;
272         wcout << L"     Cluster list: ";
273         for (int i = first_cluster; i <= last_cluster; i++) {
274             wcout << i;
275             if (i != last_cluster) wcout << L", ";
276         }
277         wcout << endl << L"     Sector list: ";
278         for (int i = item.firstCluster; i <= (last_cluster - 1) * sc +
              first_sector_of_data; i++) {
279             wcout << i;
280             if (i != (last_cluster - 1) * sc + first_sector_of_data) cout << ",
                  ";
281         }
282
283         // If file is .txt file
284         if (fileName.substr(fileName.size() - 3, 3) == L"txt") {
285             wcout << endl;
286             printTXTContent(disk, item.firstCluster);
287         }
288         else {
289             //wcout << endl << L"     ID: " << list_item.size() - 1;
290             wcout << endl << L"     Cannot open file content!";
291         }
292
293         wcout << endl << endl;
294         fileName = L"";
295     }
296 }
```

Code 16: Hàm processFolderOrFile (03)

Cụ thể:

- Đọc 32 bytes tiếp theo (dòng 207 - 208).
- Bắt đầu đọc cluster đầu tiên và bắt đầu xác định vị trí cluster cuối. (dòng 210 – 211)
- Đọc nội dung cluster và xác định xem cluster loại nào trong các loại (dòng 213 – 227)
    - End cluster
    - Cluster được đánh dấu là đã bị xoá

- Cluster bị ẩn do không đủ quyền tuy cập

- Nếu item có thuộc tính là folder thì chương trình sẽ đọc thuộc tính của folder và in ra màn hình bao gồm các thông tin: thuộc tính, cluster đầu, danh sách các cluster, danh sách sector. (dòng 229 – 257).
  - Nếu không có SDET thì lấy tên folder từ RDET
  - Thêm item hiện tại vào danh sách item
  - In ra màn hình các thuộc tính của item đó
  - Đọc sub folder nếu có
- Nếu item có thuộc tính là file thì chương trình sẽ đọc thuộc tính của folder và in ra màn hình bao gồm các thông tin: thuộc tính, dung lượng, cluster đầu, danh sách các cluster, danh sách sector và nội dung file (nếu là file ".txt') (dòng 258 – 296).
  - Quá trình đọc item có thuộc tính là file tương tự như đọc item có thuộc tính là folder. Nếu item có thuộc tính file là một file ".txt" thì sẽ in ra màn hình nội dung file đó.
  - Hàm printTXTContent:

```
342     // Print .txt file context
343     void printTXTContent(LPCWSTR disk, uint32_t cluster) {
344         BYTE data[512];
345         readSector(disk, cluster * 512, data);
346         wstring result = L"";
347 
348         for (int i = 0; i < 512; i++) {
349             if ((data[i] == 0 && data[i + 1] == 0) || (data[i] == 0xff && data[i + 1] == 0xff)) {
350                 break;
351             }
352 
353             result += wchar_t(data[i]);
354         }
355 
356         wcout << L"    Content: " << result << endl;
357     }
358 
```

Code 17: Hàm printTXTContent

2.1.2.5. Hàm printTree

In ra cây thư mục sau khi đã đọc được các thông tin.

```
297     // Print Tree
298     void printTree() {
299         int i = 0;
300         while (i < list_item.size()) {
301             ITEM tmp = list_item[i];
302 
303             wcout << L"\t";
304             for (int i = -2; i < tmp.level - 1; i++) {
305                 wcout << L"|   ";
306             }
307             wcout << L"|";
308             wcout << tmp.itemName << endl;
309 
310             i++;
311         }
312     }
313 
```

Code 18: Hàm printTree - in cây thư mục FAT32

2.2. HỆ THỐNG TẬP TIN NTFS

2.2.1. Đọc các thông tin mô tả trong Partition Boot Sector

Những thông tin của PBS (Partition Boot Sector) được lưu trữ trong struct BPB bao gồm các thuộc tính sau:

1. Số bytes mỗi sector
2. Số sectors mỗi cluster
3. Sectors chưa sử dụng
4. Mã xác định loại đĩa
5. Số sectors trên mỗi track
6. Số lượng heads
7. Số lượng sectors ẩn
8. Tổng số lượng sectors
9. Cluster bắt đầu của MFT
10. Cluster bắt đầu của MFT dự phòng (Mirror)
11. Kích cỡ File Record Segment (có thể âm vì chuyển từ hex -> bù 2 -> thập phân)
12. Kích cỡ của Index Buffer trong clusters
13. Số seri của Volume
14. Checksum (NTFS không sử dụng nên mặc định là 0)

```
struct BPB {
    unsigned int bytesPerSector;
    unsigned int sC;
    unsigned int reservedSectors;
    unsigned int mediaDescriptor;
    unsigned int sT;
    unsigned int nHeads;
    unsigned int hiddenSectors;
    unsigned int totalSectors;
    unsigned int mftCNum;
    unsigned int mftCNum_2;
    int CPFRS;
    unsigned int CPIB;
    unsigned int serialNum;
    unsigned int checksum;
};
```

Code 19: Struct BPB

Ta cần tạo một struct **NTFSVolume** để chứa các thuộc tính cần thiết để đọc thông tin của PBS (Partition Boot Sector):

```
struct NTFSVolume {
    BPB sectorInfo = {};
    int load = 0;
    // vector<MFTEntry> entries;
};
```

Code 20: Struct NTFSVolume

Ta cần thêm một số hàm để đọc được thông tin của PBS:

- Hàm **bytesToDecimal**: Hàm chuyển từ bytes sang hệ thập phân để thông tin được load khi đọc các thông tin của PBS ta sẽ lưu nó ở dưới hệ thập phân:

```
unsigned bytesToDecimal(vector<BYTE> vec)
{
    // Little endian format
    unsigned result = 0;

    for (int i = vec.size() - 1; i >= 0; i--) {
        result = result * 256 + vec[i];
    }

    return result;
}
```

Code 21: Hàm bytesToDecimal

- Hàm **readInfo**: đây là hàm để đọc thông tin của các sector, **offset** là chỉ số offset bắt đầu dưới dạng số nguyên, **length** là số byte sẽ đọc tính từ offset đó, điều này sẽ thể hiện rõ hơn ở hàm **loadSectorInfo**

```cpp
vector<BYTE> readInfo(int offset, int length, BYTE* sector)
{
    vector<BYTE> bytes;
    for (int i = 0; i < length; i++) {
        bytes.push_back(sector[offset + i]);
    }

    return bytes;
}
```

Code 22: Hàm readInfo

- Hàm **loadSectorInfo**: hàm này load lên những thông tin của PBS

```cpp
void loadSectorInfo(BYTE* sector, NTFSVolume& vol)
{
    BPB* p = &vol.sectorInfo;
    p->bytesPerSector = bytesToDecimal(readInfo(11, 2, sector));
    p->sC = bytesToDecimal(readInfo(13, 1, sector));
    p->reservedSectors = bytesToDecimal(readInfo(14, 2, sector));
    p->mediaDescriptor = bytesToDecimal(readInfo(21, 1, sector));
    p->sT = bytesToDecimal(readInfo(24, 2, sector));
    p->nHeads = bytesToDecimal(readInfo(26, 2, sector));
    p->hiddenSectors = bytesToDecimal(readInfo(28, 4, sector));
    p->totalSectors = bytesToDecimal(readInfo(40, 8, sector));
    p->mftCNum = bytesToDecimal(readInfo(48, 8, sector));
    p->mftCNum_2 = bytesToDecimal(readInfo(56, 8, sector));

    // Special case: CPFRS hex value -> binary value(two's complement)-> int
    BYTE b = sector[64];
    p->CPFRS = char(b);
    if (p->CPFRS < 0) {
        p->CPFRS *= -1;
        p->CPFRS = 1 << (p->CPFRS);
    }

    p->CPIB = bytesToDecimal(readInfo(68, 1, sector));
    p->serialNum = bytesToDecimal(readInfo(72, 8, sector));
    p->checksum = bytesToDecimal(readInfo(80, 4, sector));

    vol.load = 1;
}
```

Code 23: Hàm loadSectorInfo

- Hàm **asciiCodePrint**: Hàm in theo mã ASCII, chúng ta sẽ cần đến hàm này để xuất ra thông tin Số seri của Volume

```
void asciiCodedPrint(uint64_t n)
{
    size_t loops = sizeof(n) / sizeof(uint8_t);
    // 0x 0F A2 98 44 = (00001111 10100010 10011000 01000100)
    uint8_t* arr = new uint8_t[loops];
    for (int i = 0; i < loops; i++) {
        arr[loops - i - 1] = n & 0xFF;
        n >>= 8;
    }

    for (int i = 0; i < loops; i++) {
        if (isprint(arr[i]))
            printf("%c", arr[i]);
    }
    delete[]arr;
}
```

Code 24: Hàm asciiCodePrint

- Hàm **ntfsAnalysis**: Hàm xuất các thông tin của PBS, ở đầu hàm chúng ta có câu lệnh if để kiểm tra xem thông tin của sector đã được load chưa, nếu chưa thì load và xuất ra thông tin như bình thường:

```
void ntfsAnalysis(BYTE* sector, NTFSVolume& vol)
{
    if (!vol.load) {
        cout << "Boot Sector info not loaded. Proceeding to load." << endl;
        loadSectorInfo(sector, vol);
    }
    BPB* p = &vol.sectorInfo;
    cout << "Bytes per Sector: " << p->bytesPerSector << endl;
    cout << "Sectors per Cluster: " << p->sC << endl;
    cout << "Reserved sectors: " << p->reservedSectors << endl;
    cout << "Media Descriptor:" << p->mediaDescriptor << endl;
    cout << "Sectors per Track: " << p->sT << endl;
    cout << "Number of heads: " << p->nHeads << endl;
    cout << "Hidden Sectors: " << p->hiddenSectors << endl;
    cout << "Total Sectors: " << p->totalSectors << endl;
    cout << "Starting cluster to MFT: " << p->mftCNum << endl;
    cout << "Starting cluster to MFT mirror: " << p->mftCNum_2 << endl;
    cout << "Size of File Record Segment: " << p->CPFRS << endl;
    cout << "Index Buffer size in Clusters: " << p->CPIB << endl;
    cout << "Volume Serial Number: ";
    asciiCodedPrint(p->serialNum);
    cout << endl << "Checksum (unused by NTFS): " << p->checksum << endl;
}
```

Code 25: Hàm ntfsAnalysis

2.2.2. Hiển thị thông tin cây thư mục của phân vùng và đọc tập tin .txt.

- Định nghĩa MFTEntry:

```
// TODO: Loc nhung file co co bao system qua phan $STANDARD_INFORMATION
struct MFTEntry
{
    // Thuoc tinh co ban:
    uint16_t flag; // Trang thai cua Entry
    uint8_t name[50]; // Ten file
    uint64_t size; // in bytes
    uint32_t id;
    uint32_t pId;

    // Co bao system
    uint32_t filePermissionsFlag; // offset 0x70, 4

    // Thuoc tinh nang cao
    uint32_t sign; // Dau hieu nhan biet
    uint16_t startAttribute; // Offset bat dau cua phan attribute
    uint32_t entrySize; // kich thuoc entry (offset 18->1B) - dung de danh dau ket thuc entry
    uint32_t stdInfoSize; // Kich thuoc STANDARD_INFORMATION
    uint16_t nameOffset; // Offset bat dau cua $FILE_NAME
    uint8_t nameLen; // Chieu dai ten file
    uint16_t dataOffset;
    uint16_t nonresFlag;

    uint8_t* data = NULL;
};
```

Code 26: Struct MFTEntry

Trong một entry thuộc MFT thường đi kèm với nhiều thuộc tính khác nhau, ở đây quan tâm đến những giá trị chính của một Entry để phân tích các file và thư mục.

1. Flag: Là trạng thái của entry, dùng để xác định entry có phải là của thư mục hay không
2. Name: Là tên của file ứng với entry đó
3. Size: Là kích thước của file
4. ID: Là định danh của entry.
5. PID: Là định dạnh của entry cha (thư mục cha) tham chiếu tới nó.
6. FilePermissionsFlag: Cho biết một file có những cờ báo nào, dùng để xác định một file có bị ẩn, thuộc hệ thống.
7. Sign: Dấu hiệu nhận biết một entry
8. StartAttribute: bắt đầu của phần thuộc tính
9. EntrySize: Kích thước đã dùng trong một entry
10. StdInfoSize: Kích thước của một thuộc tính $STANDARD_INFORMATION
11. NameOffset: Offset bắt đầu của $FILE_NAME
12. NameLen: Chiều dài tên file
13. DataOffset: Offset bắt đầu của $DATA
14. NonResFlag: Cho biết thuộc tính $DATA có thuộc loại non-resident hay không
15. Data: chứa dữ liệu file .txt (là NULL nếu không là file .txt)

- Hàm **analyze** dùng để phân tích một entry trong MFT và ghi vào những thuộc tính cần thiết có trong mỗi MFTEntry.

```
void analyze(BYTE* block, MFTEntry& entry)
{
    for (int i = 0; i < 4; i++) {
        entry.sign = (entry.sign << 8) | block[i];
    }
    entry.id = bytesToDecimal(readInfo(0x2C, 4, block));
    entry.startAttribute = bytesToDecimal(readInfo(0x14, 2, block));
    entry.flag = bytesToDecimal(readInfo(0x16, 2, block));
    entry.entrySize = bytesToDecimal(readInfo(0x18, 4, block));
    entry.stdInfoSize = bytesToDecimal(readInfo(entry.startAttribute + 4, 4, block));

    entry.filePermissionsFlag = bytesToDecimal(readInfo(0x70, 4, block));


    int nameO = entry.stdInfoSize + entry.startAttribute;
    int fnSize = bytesToDecimal(readInfo(nameO + 4, 4, block));

    entry.nameOffset = bytesToDecimal(readInfo(nameO + 20, 2, block));

    nameO += entry.nameOffset;

    entry.pId = bytesToDecimal(readInfo(nameO, 6, block));
    entry.nameLen = bytesToDecimal(readInfo(nameO + 64, 1, block));

    getFilename(block, entry.name);
```

Code 27: Hàm analyze (01)

```
int nextAttributeOffset = entry.startAttribute + entry.stdInfoSize + fnSize;

entry.size = 0;

bool B1 = (entry.flag & 2) != 2 && (entry.flag & 4) != 4 && (entry.flag & 8) != 8;

if (B1) // Phat hien co bao thu muc
{
    int id = -1;
    do
    {
        id = bytesToDecimal(readInfo(nextAttributeOffset, 4, block));
        if (id == 128) break;

        nextAttributeOffset += bytesToDecimal(readInfo(nextAttributeOffset + 4, 4, block));

    } while (id != 128); // Dung khi tim thay id = 128

    entry.dataOffset = nextAttributeOffset;

    uint8_t nonRes = bytesToDecimal(readInfo(entry.dataOffset + 8, 1, block));

    if (nonRes == 0)
        entry.size = bytesToDecimal(readInfo(entry.dataOffset + 16, 4, block));
    else
    {
        uint16_t offset = entry.dataOffset + 64;
        uint8_t head = bytesToDecimal(readInfo(offset, 1, block)); // 0x21 = 0010 0001
        uint8_t l = (head >> 4) | 0;                               //    l = 0000 0010
```

Code 28:Hàm analyze (02)

```
56          {
57              uint16_t offset = entry.dataOffset + 64;
58              uint8_t head = bytesToDecimal(readInfo(offset, 1, block)); // 0x21 = 0010 0001
59              uint8_t l = (head >> 4) | 0;                               //    l = 0000 0010
60              uint8_t r = head - (l << 4);                               //    r = 0000 0001
61
62              entry.size = bytesToDecimal(readInfo(offset + 1, r, block)) * 4096;
63          }
64      }
65      else entry.dataOffset = -1; // Bo qua data.
```

Code 29: Hàm analyze (03)

- Hàm **printEntry** xuất ra thông tin cần thể hiện đối với một entry.

```
void printEntry(MFTEntry& entry)
{
    cout << "-------------------------------------------------------" << endl;
    cout << "Signature: ";
    asciiCodedPrint(entry.sign);
    cout << endl;
    cout << "Is directory: " << ((entry.flag & 2)==2) << endl;
    cout << "Size: " << entry.size << endl;
    cout << "ID: " << entry.id << endl;
    cout << "Parent ID: " << entry.pId << endl;
    printf("Name: %s\n", entry.name);
}
```

Code 30: Hàm printEntry

- Hàm **dataRunList** chiết xuất data run khi phát hiện $DATA thuộc loại non-resident và trả về mảng các byte.

```
void dataRunList(BYTE* block, BYTE*& list, MFTEntry& entry)
{
    if (entry.dataOffset != -1) {
        uint16_t offset = entry.dataOffset + 64;
        if (bytesToDecimal(readInfo(entry.dataOffset, 4, block)) != 128)
            return;
        uint8_t head = bytesToDecimal(readInfo(offset, 1, block)); // 0x21 = 0010 0001
        uint8_t l = (head >> 4) | 0;                               //    l = 0000 0010
        uint8_t r = head - (l << 4);                               //    r = 0000 0001

        list = new BYTE[1 + l + r];
        for (int i = 0; i < 1 + l + r; i++) {
            list[i] = block[offset + i];
        }
    }
}
```

Code 31: Hàm dataRunList

- Hàm **readData** đọc dữ liệu trên entry nếu là resident, hoặc dựa vào data run list để đi tới phần data thật của file.

```
void readData(LPCWSTR drive, BYTE* block, MFTEntry& entry)
{
    if (strstr(reinterpret_cast<char*>(entry.name), ".txt") == NULL) {
        cout << "Data not readable." << endl;
    }
    if (entry.dataOffset != -1) {
        uint8_t nonRes = bytesToDecimal(readInfo(entry.dataOffset + 8, 1, block));

        uint16_t f = entry.flag;
        bool bits[4] = { f & 1,f & 2, f & 4,f & 8 };

        // Kiem tra entry co phai la cua thu muc?
        if (bits[1]) {
            return;
        }

        if (nonRes == 0) {
            // Doc data thuoc loai resident
            uint16_t offset = entry.dataOffset + bytesToDecimal(readInfo(entry.dataOffset + 10, 2, block));
            uint32_t size = bytesToDecimal(readInfo(entry.dataOffset + 16, 4, block));

            entry.data = new uint8_t[size + 1];
            for (int i = 0; i < size; i++) {
                entry.data[i] = block[offset + i];
            }
            entry.data[size] = '\0';

            entry.size = size;
```

Code 32: Hàm readData (01)

```
            entry.data[i] = block[offset + i];
        }
        entry.data[size] = '\0';

        entry.size = size;
    }
    else
    {
        uint16_t offset = entry.dataOffset + 64;
        uint8_t head = bytesToDecimal(readInfo(offset, 1, block)); // 0x21 = 0010 0001
        uint8_t l = (head >> 4) | 0;                               //    l = 0000 0010
        uint8_t r = head - (l << 4);                               //    r = 0000 0001

        uint32_t size = bytesToDecimal(readInfo(offset + 1, r, block)) * 4096;
        uint32_t firstClus = bytesToDecimal(readInfo(offset + r + 1, l, block));

        entry.data = new uint8_t[size + 1];
        entry.data = ReadBytes(drive, uint64_t(firstClus) * 4096, size);
        entry.data[size] = '\0';
        entry.size = size;
    }
}
else {
    cout << "Entry data not readable." << endl;
}
}
```

Code 33: Hàm readData (02)

- Hàm **printData** in ra data nếu đọc được (khác NULL).

```
void printData(MFTEntry& entry)
{
    if (entry.data != NULL) {
        printf("%s", entry.data);
    }
}
```

Code 34: Hàm printData

- Hàm **loadEntries** để phân tích toàn bộ các entry MFT vào NTFSVolume.

```
void loadEntries(LPCWSTR drive, BYTE* block, NTFSVolume& vol)
{
    // Check if the block passed in is the $MFT
    uint8_t name[50];
    getFilename(block, name);

    BPB* p = &vol.sectorInfo;

    if (strcmp(reinterpret_cast<char*>(name), "$MFT") == 0) {
        // Scan through the MFT entries.
        int jump = 1024; // size of MFT

        int loop = 1;

        uint64_t offset =   vol.sectorInfo.bytesPerSector*
                            vol.sectorInfo.mftCNum*
                            vol.sectorInfo.sC;
        uint8_t filename[50];

        uint64_t limit = endOfMFT(drive, vol) * 8;
        uint64_t rp = offset;
        do {

            BYTE* entry;
            entry = ReadBytes(drive, rp , jump);
            getFilename(entry, filename);
            // Check signature

            // Push any entry that is valid
            if (bytesToDecimal(readInfo(0, 4, entry)) == 0x454C4946) {

                MFTEntry obj;
                analyze(entry, obj);
                readData(drive, entry, obj);
                vol.entries.push_back(obj);
            }
            rp += 1024;
            delete[]entry;
        } while (rp / 512<= limit);
    }
}
```

Code 35: Hàm loadEntries

- Hàm **endOfMFT** tính toán điểm kết thúc của một MFT thông qua kích thước của nó.

```cpp
// Tinh toan diem ket thuc MFT
uint64_t endOfMFT(LPCWSTR drive, NTFSVolume& vol)
{
    BYTE* MFT = new BYTE[1024];
    uint64_t pos = uint64_t(vol.sectorInfo.mftCNum) * vol.sectorInfo.sC * vol.sectorInfo.bytesPerSector;

    MFT = ReadBytes(drive, pos, vol.sectorInfo.CPFRS);

    MFTEntry obj;
    analyze(MFT,obj);

    BYTE* list;
    dataRunList(MFT,list,obj);

    uint8_t head = list[0];
    uint8_t l = (head >> 4) | 0;
    uint8_t r = head - (l << 4);

    uint64_t sz = bytesToDecimal(readInfo(1, r, list));
    return vol.sectorInfo.mftCNum + sz;
}
```

Code 36: Hàm endOfMFT

- Hàm **outputEntries** in ra các thông tin entry đã lưu trong NTFSVolume.

```cpp
void outputEntries(NTFSVolume& vol)
{
    for (int i = 0; i < vol.entries.size(); i++) {
        printEntry(vol.entries[i]);
    }
}
```

Code 37: Hàm outputEntries

- Hàm **constructTree** (chưa hoàn tất) nhằm mục đích tạo ra một cấu trúc cây chứa tổ chức thư mục.

```cpp
void constructTree(Tree& tree, NTFSVolume& vol)
{
    int c_Passed = 0;
    int c_Failed = 0;

    // Tim cac file hien tren nhanh.
    for (int i = 0; i < vol.entries.size(); i++) {
        MFTEntry* p = &vol.entries[i];

        if ((p->filePermissionsFlag & Perms::System) == 0 && p->pId == 5) {
            tree.branches.push_back(new TreeNode{ p->pId,p->id,*p });
        }
    }


    for (int i = 0; i < tree.branches.size(); i++) {
        // Xet tung nhanh, xem co ton tai file nao thuoc nhanh do

        uint32_t init = tree.branches[i]->Id;

        for (int j = 0; j < vol.entries.size(); j++) {
            MFTEntry* p = &vol.entries[j];

            if ((p->filePermissionsFlag & Perms::System) == 0 && p->pId == tree.branches[i]->Id) {
                tree.branches[i]->nodes.push_back(new TreeNode{ p->pId,p->id,*p });
            }
        }
```

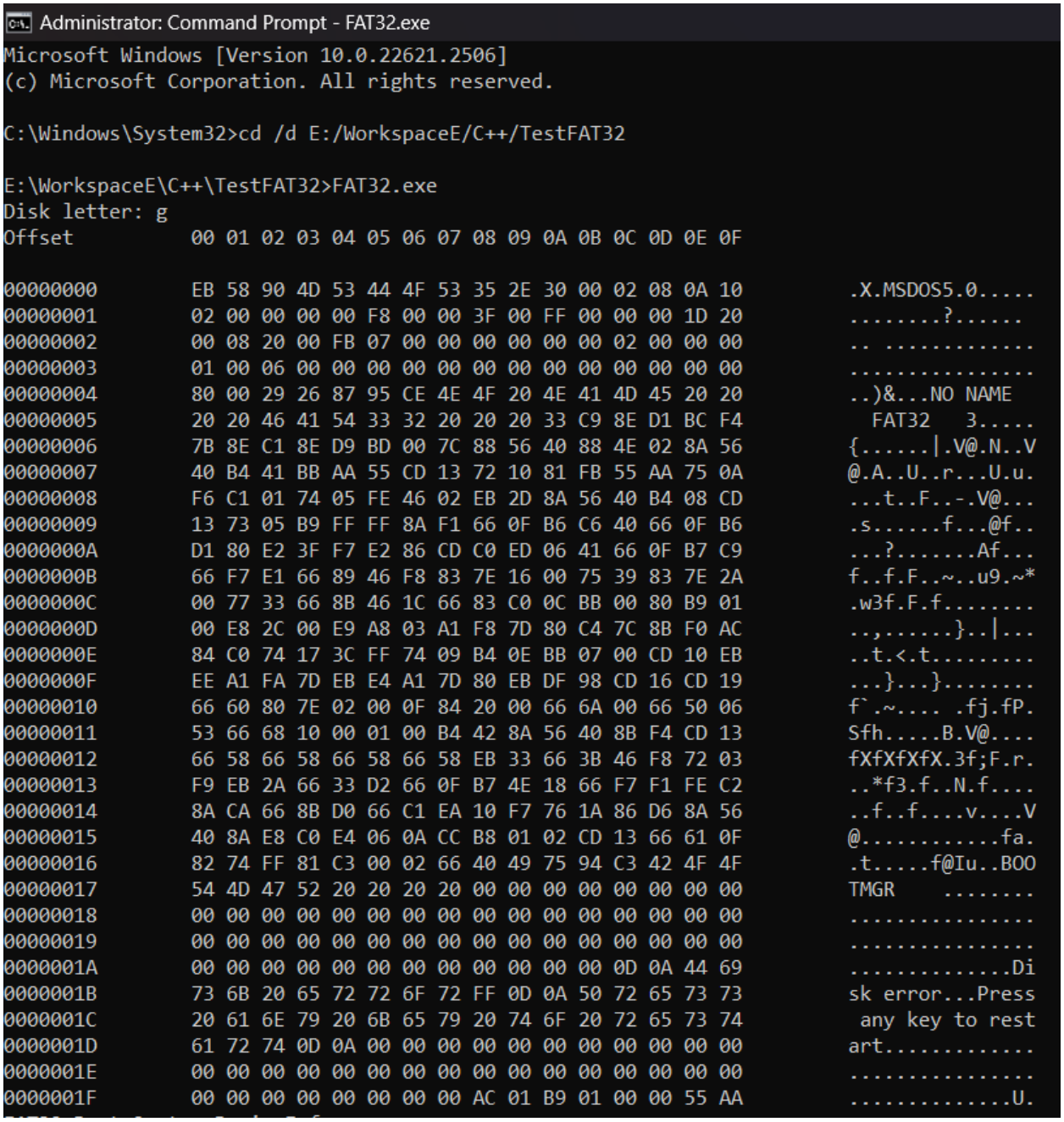
Code 38: Hàm constructTree (chưa hoàn tất)

2.3. DEMO CHƯƠNG TRÌNH

2.3.1. Hệ thống tập tin FAT32

Để chạy chương trình, sử dụng các tệp trong thư mục "Source/FAT32" để tạo tệp thực thi (.exe), hoặc sử dụng tệp 'FAT32.exe' trong thư mục "Release" (đã được chúng em tạo sẵn) bằng cách mở Command Prompt với quyền Admin và gọi tệp thực thi.

2.3.1.1. Đọc các thông tin mô tả trong Boot Sector



Hình 1. BootSector đọc được từ disk G

```
FAT32 Boot Sector Basic Info:
File system type                      :     FAT32
Bytes per sector                      :       512
Sector per cluster (Sc)               :         8 sector(s)
Sector in Boot Sector (Sb)            :      4106 sector(s)
Number of FATs (nF)                   :         2 FAT(s)
Number of root directory entries (Sr) :         0 sector(s)
Size of volume (Sv)                   :   2099200 sector(s)
Size of FAT (Sf)                      :      2043 sector(s)
Start cluster RDET                    :         2
Additional Information Sector         :         1 sector(s)
Boot Sector's Backup Sectors          :         6 sector(s)
```

Hình 2. Thông tin mô tả BootSector

2.3.1.2. Hiển thị thông tin cây thư mục của phân vùng và đọc tập tin .txt.

```
Disk items' simple properies:

*Item: Folder A
   Type: Folder
   First Cluster: 7
   Cluster list: 7
   Sector list: 8232, 8233, 8234, 8235, 8236, 8237, 8238, 8239, 8240

*Item: Cannot Read This.xls
   Type: File
   Size: 0 B
   First Cluster: 0
   Cluster list: 0
   Sector list: 8176, 8177, 8178, 8179, 8180, 8181, 8182, 8183, 8184
   Cannot open file content!

*Item: File 1.txt
   Type: File
   Size: 35 B
   First Cluster: 8
   Cluster list: 8
   Sector list: 8240, 8241, 8242, 8243, 8244, 8245, 8246, 8247, 8248
   Content: Immortal of the priest is hoyohoyo!


*Item: Folder AB
   Type: Folder
   First Cluster: 9
   Cluster list: 9
   Sector list: 8248, 8249, 8250, 8251, 8252, 8253, 8254, 8255, 8256

*Item: File 0.txt
   Type: File
   Size: 18 B
   First Cluster: 10
   Cluster list: 10
   Sector list: 8256, 8257, 8258, 8259, 8260, 8261, 8262, 8263, 8264
   Content: DO IT! JUST DO IT!
```

Hình 3. Thông tin các thư mục và tập tin trong G

```
*Item: Folder B
   Type: Folder
   First Cluster: 11
   Cluster list: 11
   Sector list: 8264, 8265, 8266, 8267, 8268, 8269, 8270, 8271, 8272

*Item: File 2.txt
   Type: File
   Size: 74 B
   First Cluster: 12
   Cluster list: 12
   Sector list: 8272, 8273, 8274, 8275, 8276, 8277, 8278, 8279, 8280
   Content: give me some time, i will make you feel good and bad at the same time :)))


*Item: File 3.txt
   Type: File
   Size: 183 B
   First Cluster: 13
   Cluster list: 13
   Sector list: 8280, 8281, 8282, 8283, 8284, 8285, 8286, 8287, 8288
   Content: - I think I've finish
- What? you've finish? It've just been five minutes.
- What do you expect! Ten minutes of pull-ups? You gotta be kidding!
- Thought you were stronger then :P.


*Item: Magic Chemical Reaction.txt
   Type: File
   Size: 17 B
   First Cluster: 14
   Cluster list: 14
   Sector list: 8288, 8289, 8290, 8291, 8292, 8293, 8294, 8295, 8296
   Content: H2 + O2 -> HOHO
```

Hình 4. Thông tin các thư mục và tập tin trong G (tiếp theo)

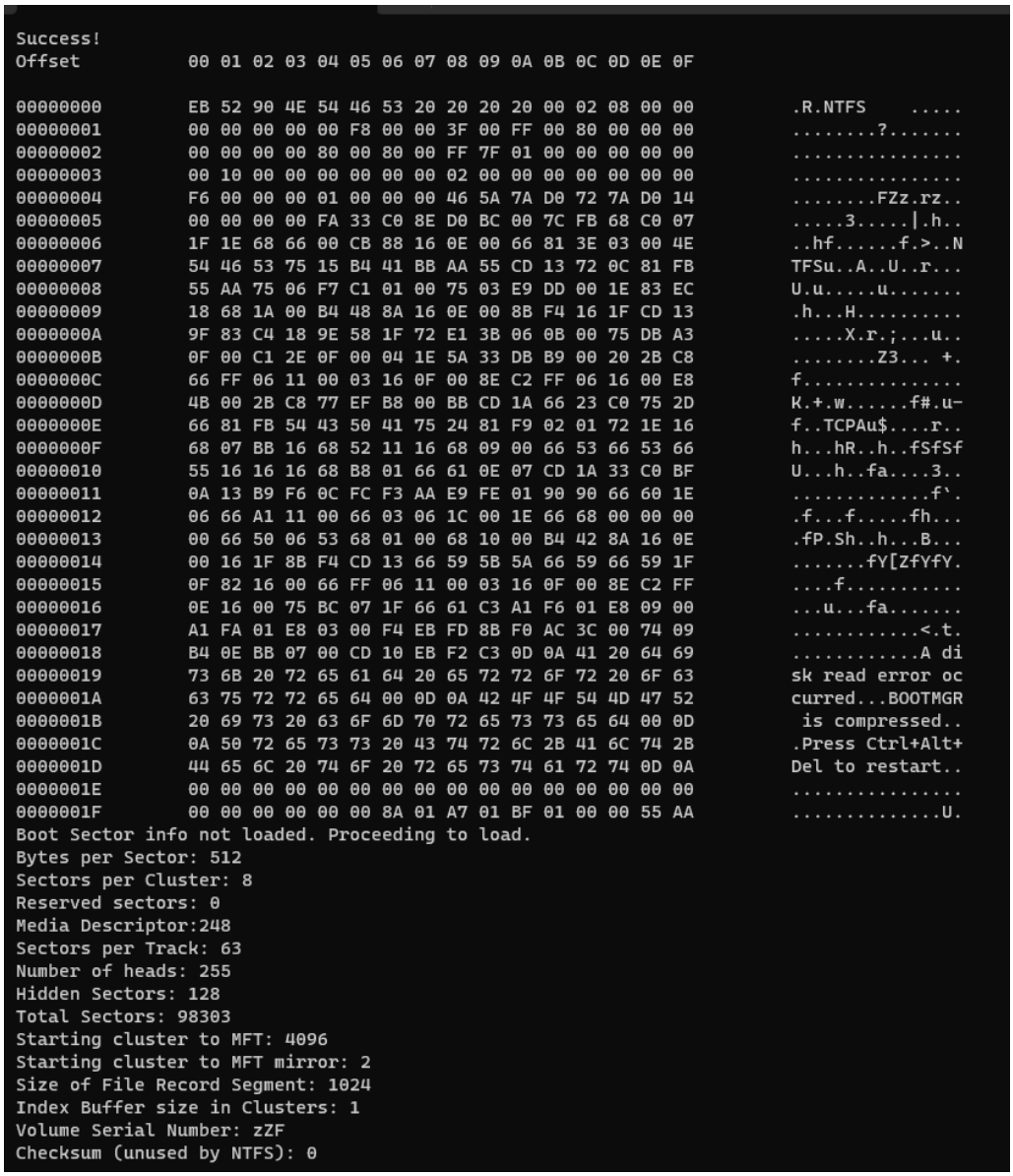
```
Folder Tree:
        |g:
        |   |Folder A
        |   |   |Cannot Read This.xls
        |   |   |File 1.txt
        |   |   |Folder AB
        |   |   |   |File 0.txt
        |   |Folder B
        |   |   |File 2.txt
        |   |   |File 3.txt
        |   |Magic Chemical Reaction.txt
Press any key to continue . . .
```

Hình 5. Cây thư mục - FAT32

- Link các folder trong disk G:
  https://drive.google.com/file/d/1YY04O6tHckKxXAyZjTihed9tZ2zUl4HH/view?usp=sharing

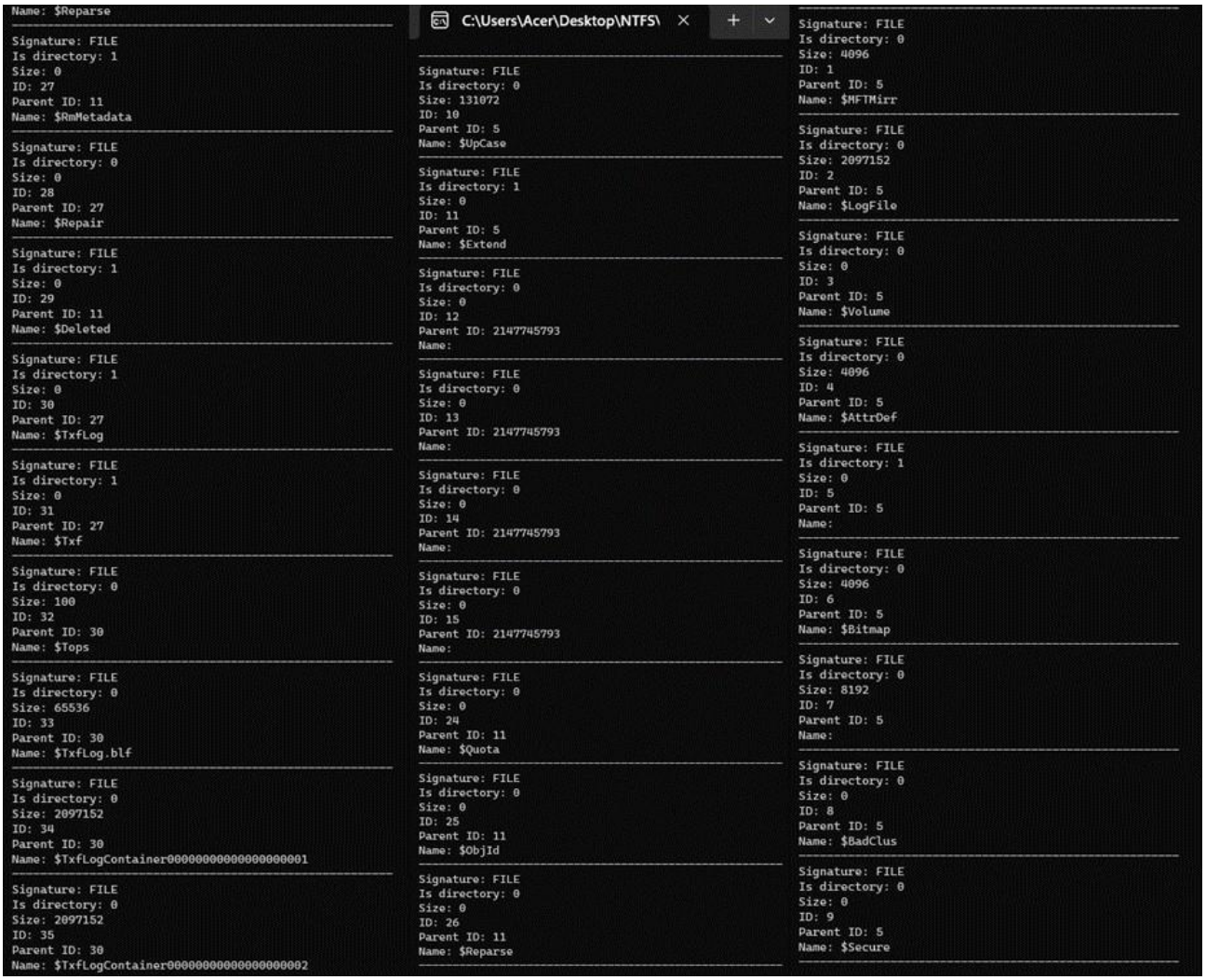
2.3.2. Hệ thống tập tin NTFS

2.3.2.1. Hiển thị thông tin Partition Boot Sector và cụ thể Bios Parameter Block của ổ đĩa NTFS.

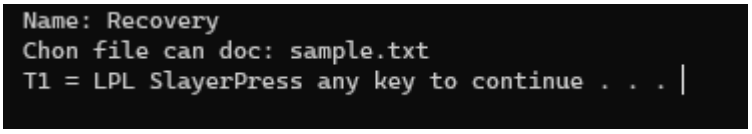


Hình 6. Thông tin Partition BootSector - Bios Parameter Block

2.3.2.2. Hiển thị cây thư mục của phân vùng và nội dung một file .txt:



Hình 7. Cây thư mục của phân vùng NTFS



Hình 8. Đọc nội dung nội dung một file .txt (NTFS)

3. TÀI LIỆU THAM KHẢO

[1] Trần Trung Dũng, Phạm Tuấn Sơn (2022). Giáo trình Hệ điều hành. NXB Khoa học và Kỹ thuật, Hồ Chí Minh.

[2] Lê Viết Long, Bài giảng: Hệ thống tập tin FAT.

[3] Tài liệu giảng viên cung cấp trong quá trình thực hiện đồ án.