

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN



BÁO CÁO PROJECT 1
QUẢN LÝ HỆ THỐNG TẬP TIN TRÊN WINDOWS

Môn học: Hệ điều hành

Giảng viên: ThS.Lê Viết Long

MỤC LỤC

1. Lời cảm ơn	3
2. Giới thiệu.....	3
2.1. Đặt vấn đề	3
2.2. Mục tiêu	3
2.3. Cách tiếp cận	4
3. Thông tin nhóm.....	4
4. Bảng phân công công việc và đánh giá mức độ hoàn thành	4
4.1. Bảng phân công công việc	4
4.2. Đánh giá mức độ hoàn thành.....	6
5. Bước chuẩn bị: tạo ổ đĩa ảo	7
6. Mô tả thuật toán	12
6.1. FAT32.....	12
6.2. NTFS	16
7. Các bước thực hiện: FAT32	22
8. Các bước thực hiện: NTFS.....	28
8.1. Các file header	28
8.2. Các file .cpp và phần thực thi	29
9. Demo chương trình	39
9.1. FAT32.....	39
9.2. NTFS	42
10. Tính ứng dụng của đồ án.....	51
11. Các nguồn tham khảo	52

1. Lời cảm ơn

Lời đầu tiên, cho phép chúng em được gửi lời cảm ơn đến thầy - ThS. Lê Viết Long, người đã trực tiếp giảng dạy chúng em suốt thời gian qua và đưa ra những lời hướng dẫn, góp ý giúp đỡ án ngày càng hoàn thiện.

Trong quá trình thực hiện đồ án này chúng em không thể tránh khỏi những thiếu sót vì chưng có đủ kinh nghiệm, chúng em mong sẽ nhận được những ý kiến đóng góp của thầy để đồ án của chúng em sẽ hoàn thiện hơn và thực hiện tốt hơn các đồ án sau này.

2. Giới thiệu

2.1. Đặt vấn đề

Xây dựng chương trình máy tính đọc các thông tin trên phân vùng FAT32 và NTFS là một đề tài nghiên cứu mang tính ứng dụng cao. Với sự phát triển không ngừng của công nghệ thông tin, việc nghiên cứu và xây dựng các chương trình hỗ trợ cho việc quản lý và xử lý dữ liệu trên máy tính ngày càng được đặt ra yêu cầu cao hơn.

Phân vùng FAT32 và NTFS là hai loại hệ thống tập tin phổ biến trên hệ điều hành Windows. Tuy nhiên, việc đọc các thông tin từ những phân vùng này vẫn là một thách thức đối với nhiều người sử dụng. Để giải quyết vấn đề này, chúng em đã tiến hành nghiên cứu và xây dựng một chương trình máy tính đọc các thông tin trên phân vùng FAT32 và NTFS.

Chương trình này được phát triển bằng ngôn ngữ lập trình C++, sử dụng các thư viện và công nghệ tiên tiến để đọc các thông tin từ phân vùng FAT32 và NTFS một cách nhanh chóng và chính xác. Chương trình có khả năng đọc các thông tin như tên tập tin, kích thước, thời gian sửa đổi, ngày tạo và nhiều thông tin khác từ những phân vùng này.

2.2. Mục tiêu

Mục tiêu của project là tập trung vào việc thu thập thông tin từ khu vực đầu tiên (Boot Sector) của mỗi phân vùng và bảng thư mục gốc (RDET) trong hai hệ thống quản lý tập tin là FAT32 và NTFS. Chương trình sẽ được thiết kế để truy xuất và đọc các thông tin này một cách chính xác và hiệu quả, giúp người dùng dễ dàng quản lý và sử dụng dữ liệu trên các phân vùng này.

Để đạt được mục tiêu thu thập thông tin từ khu vực đầu tiên và bảng thư mục gốc của các phân vùng FAT32 và NTFS, chương trình sẽ tập trung vào việc lấy các thông số quan trọng như byte per sector, sector per cluster, số lượng sector, cluster... thông qua việc phân tích các giá trị trong Boot Sector và các thuộc tính của tập tin trong bảng thư mục gốc (RDET). Những thông tin này rất quan trọng để chương trình có thể xác định được cấu trúc của phân vùng và truy xuất được các tập tin, thư mục trên đó một cách chính xác. Ngoài ra, chương trình cũng sẽ cung cấp cho người dùng các thông tin liên quan đến khối lượng dữ liệu trên phân vùng, dung lượng khả dụng và các thông tin khác để người dùng có thể quản lý tập tin và thư mục trên phân vùng một cách dễ dàng và hiệu quả.

2.3. Cách tiếp cận

Để lấy thông tin của Boot Sector, chương trình sẽ đọc các giá trị trong Boot Sector của phân vùng FAT32 hoặc NTFS. Thông tin này sẽ giúp chương trình hiểu được cấu trúc của phân vùng đang được đọc và tìm kiếm các thông tin khác trong phân vùng đó.

Sau khi có thông tin về Boot Sector, chương trình sẽ tiếp tục đọc bảng thư mục gốc (RDET) của phân vùng để lấy các thuộc tính của tập tin như tên tập tin, kích thước tập tin, địa chỉ cluster đầu tiên của tập tin và nhiều thuộc tính khác. Quá trình này sẽ được thực hiện thông qua việc tìm kiếm thông tin về tập tin trong bảng thư mục gốc (RDET) và lấy các thuộc tính của tập tin đó.

Từ các thông tin lấy được từ Boot Sector và RDET, chương trình có thể phân tích và hiển thị các thông tin cần thiết và phân vùng và các tập tin trong đó. Quá trình lấy thông tin này sẽ giúp người dùng có thể hiểu được cấu trúc của phân vùng và quản lý các tập tin một cách hiệu quả hơn.

3. Thông tin nhóm

STT	MSSV	HỌ VÀ TÊN
1	22127479	Lê Hoàng Linh
2	22127278	Vũ Thu Minh
3	22127103	Lê Thị Hồng Hạnh
4	22127311	Hồ Hà Nhi

4. Bảng phân công công việc và đánh giá mức độ hoàn thành

4.1. Bảng phân công công việc

Công việc	Người thực hiện
Code	
FAT32	
1.Đọc các thông tin chi tiết của một phân vùng	Lê Thị Hồng Hạnh
2.Hiển thị cây thông tin thư mục của phân vùng: + Chia đường dẫn thành các thành phần để xác định các thư mục con. + Đọc bảng RDET để lấy thông tin về tất cả các tập tin/ thư mục trong phân vùng + Duyệt các thư mục con và lấy thông tin tập tin/ thư mục cần truy xuất. + Hiển thị thông tin về tập tin/ thư mục bao gồm: tên, kích thước, đường dẫn, vị trí trên đĩa cứng. + Xác định offset của tập tin trong từng \cluster.	Lê Thị Hồng Hạnh
NTFS	
1.Đọc các thông tin chi tiết của một phân vùng	Hồ Hà Nhi
2.Hiển thị cây thông tin thư mục của phân vùng Chia đường dẫn thành các thành phần để xác định các thư mục con.	
Đọc bảng RDET để lấy thông tin về tất cả các tập tin/ thư mục trong phân vùng	Vũ Thu Minh
Duyệt các thư mục con và lấy thông tin tập tin/ thư mục cần truy xuất.	
Hiển thị thông tin về tập tin/ thư mục bao gồm: tên, kích thước, đường dẫn,	Lê Hoàng Linh

vị trí trên đĩa cứng.	
Xác định offset của tập tin trong từng \cluster.	
BÁO CÁO	
FAT 32	
1.Mô tả thuật toán	Lê Hoàng Linh
2.Các bước thực hiện	Lê Thị Hồng Hạnh
3.Demo	
NTFS	
1.Mô tả thuật toán	Hồ Hà Nhi
2.Các bước thực hiện	
File header	
MFT.cpp	Lê Hoàng Linh
Utils.cpp	
Volume.cpp	Vũ Thu Minh
PartiotionBootSector.cpp	Hồ Hà Nhi
3.Demo	Hồ Hà Nhi
Phần khác của báo cáo	
1.Trang bìa	
2.Định dạng cho phần mục lục	Lê Hoàng Linh
3.Nguồn	
4.Lời giới thiệu & lời cảm ơn	
5.Bước chuẩn bị: tạo ổ đĩa ảo	Vũ Thu Minh
6.Tính ứng dụng của đồ án	

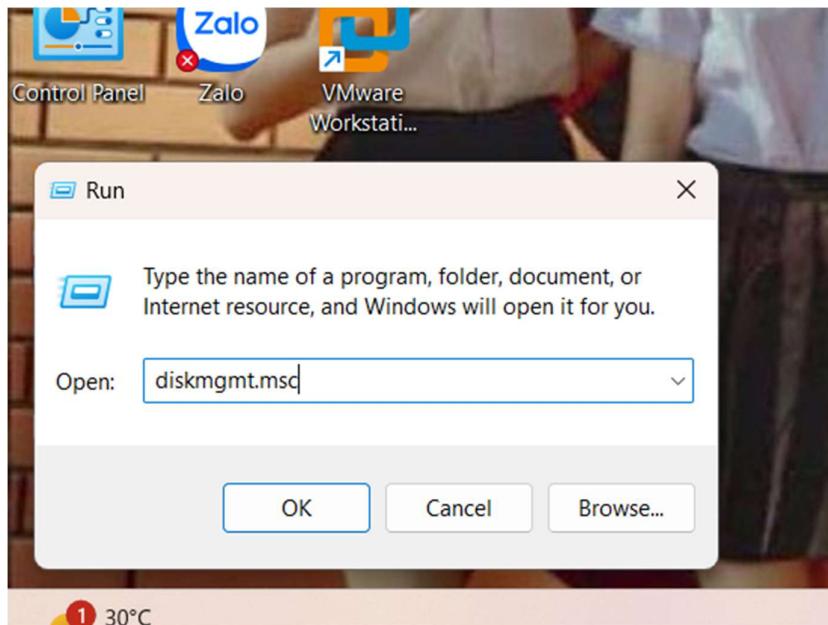
4.2. Đánh giá mức độ hoàn thành

Yêu cầu	Mức độ hoàn thành (100%)
FAT32	
1.Đọc các thông tin chi tiết của một phân vùng	100%
2.Hiển thị cây thông tin thư mục của một phân vùng	100%
NTFS	
1.Đọc các thông tin chi tiết của một phân vùng	100%
2.Hiển thị cây thông tin thư mục của một phân vùng	100%

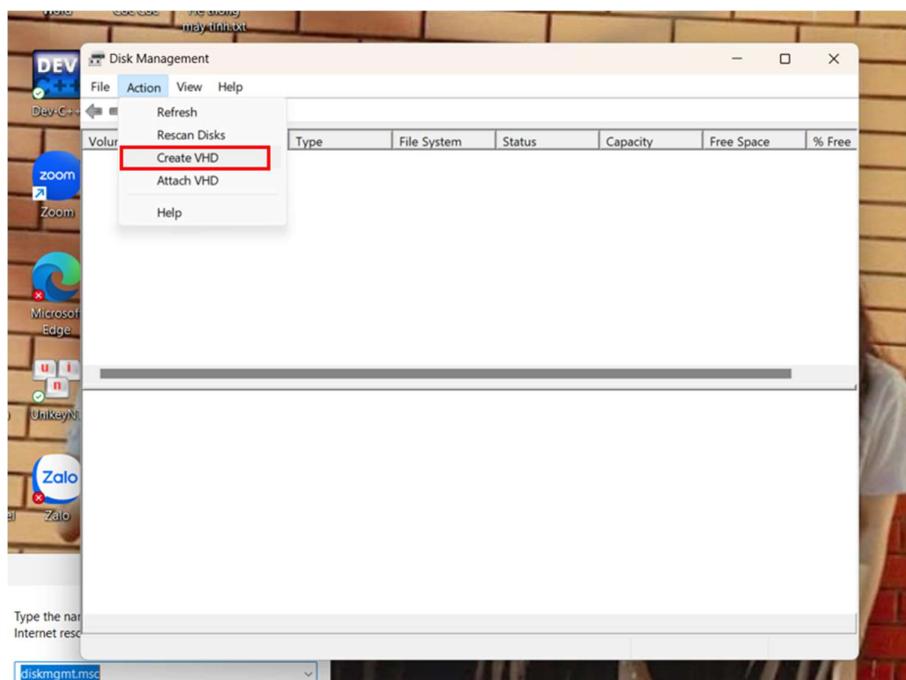
⇒ Hoàn thành 100% yêu cầu đề bài.

5. Bước chuẩn bị: tạo ổ đĩa ảo

- **B1:** Nhấn tổ hợp phím Window + R, sau khi cửa sổ Run xuất hiện. Nhập “diskmgmt.msc” vào ô **Open** và nhấn chọn OK.

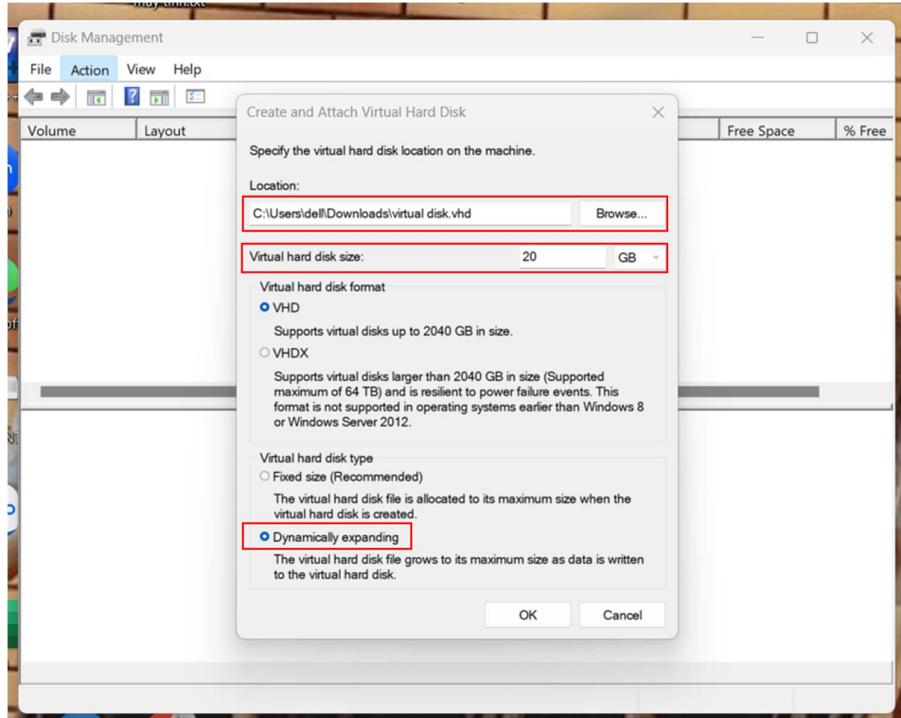


- **B2:** Sau khi giao diện **Disk Management** xuất hiện, nhấn chọn tab **Action** trên thanh công cụ. Chọn **Create VHD** để bắt đầu tạo ổ đĩa ảo.

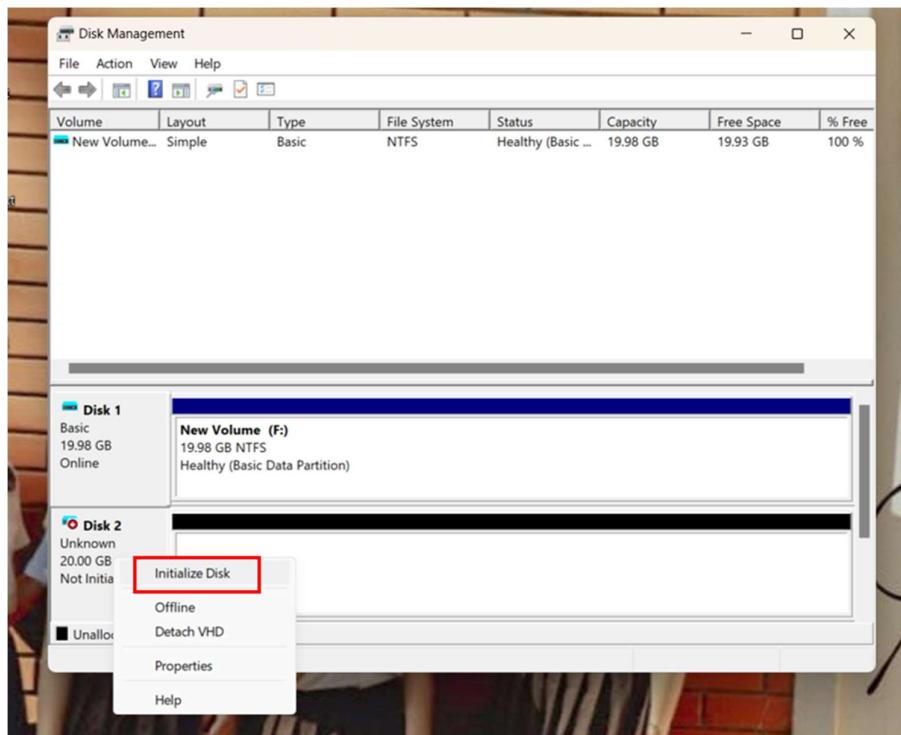


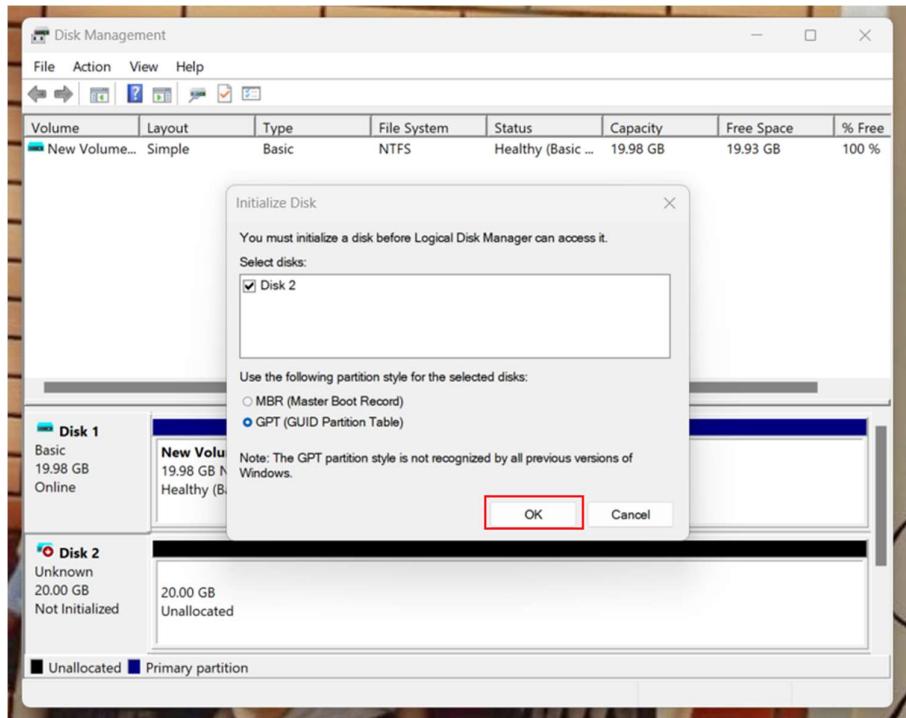
- **B3:** Trong bảng **Create and Attach Virtual Hard Disk**, chọn vị trí ổ đĩa ảo ở mục **Location**. Ở mục **Virtual hard disk size**, nhập kích thước

mong muốn cho ổ đĩa. Sau đó tích chọn ở mục **Dynamically expanding** và nhấn OK.

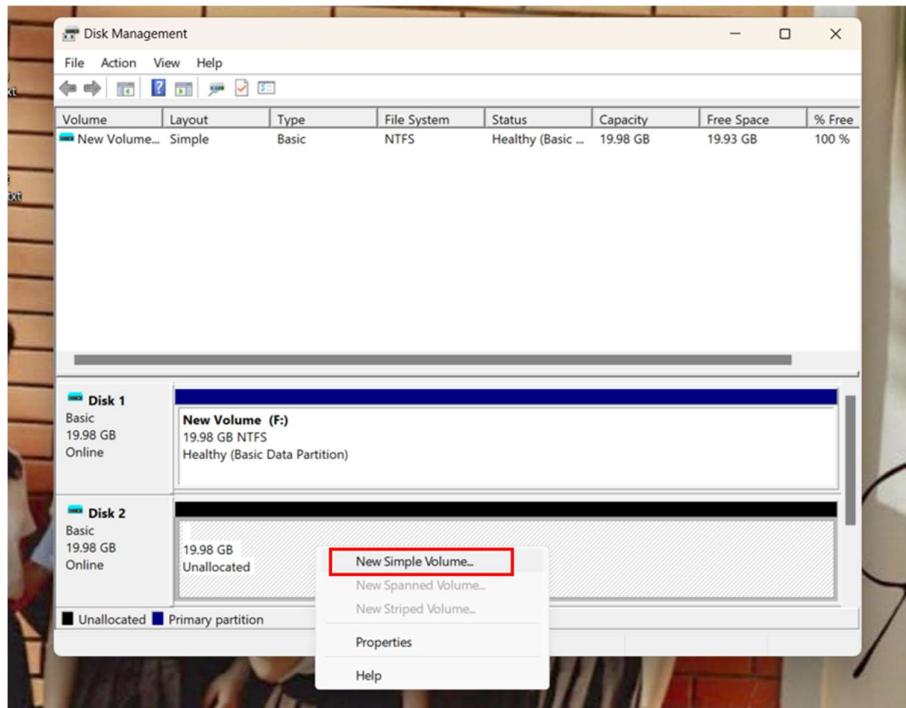


- **B4:** Click chuột phải vào vùng Disk 2 vừa tạo và chọn **Initialize Disk**. Khi bảng thông báo xuất hiện, nhấn OK để tiến hành tối ưu hóa ổ đĩa ảo.

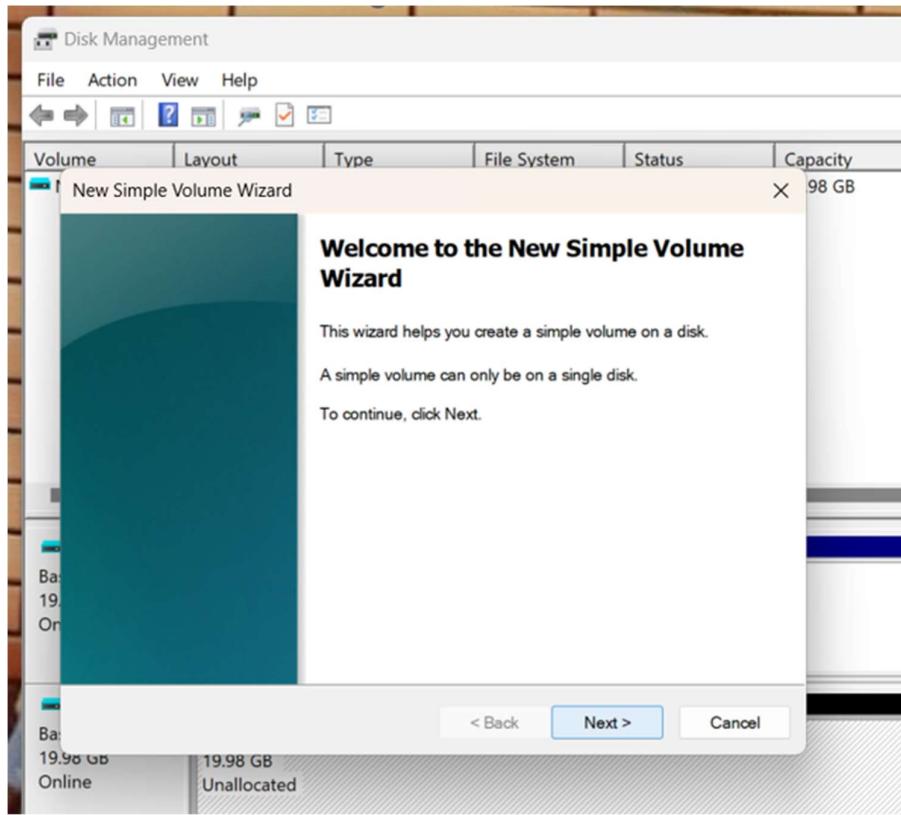




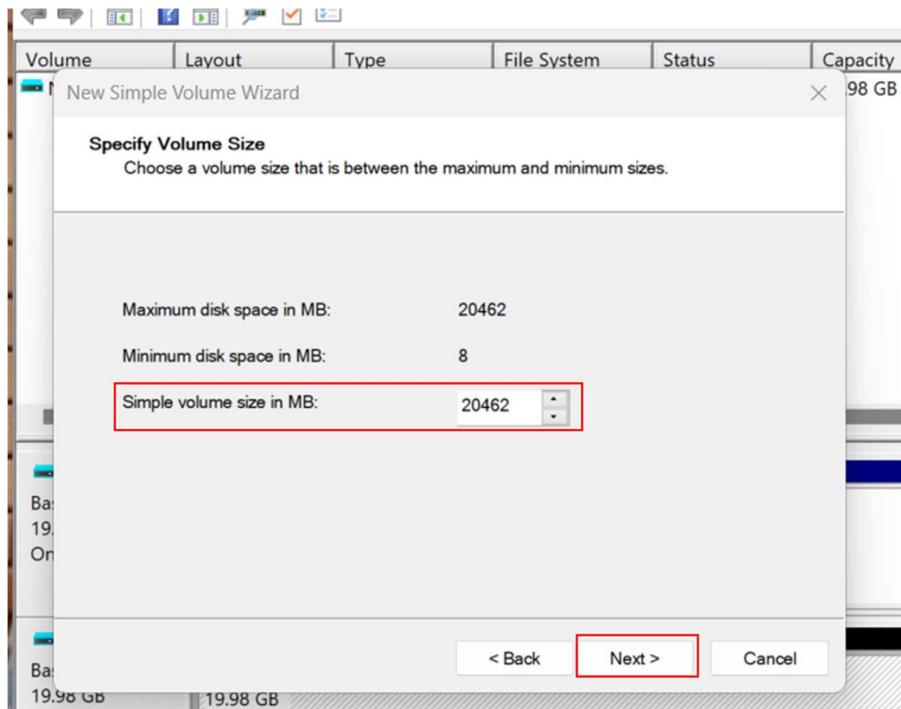
- **B5:** Nháy chuột phải vào vùng ảo tạm thời và chọn **New Simple Volume**.



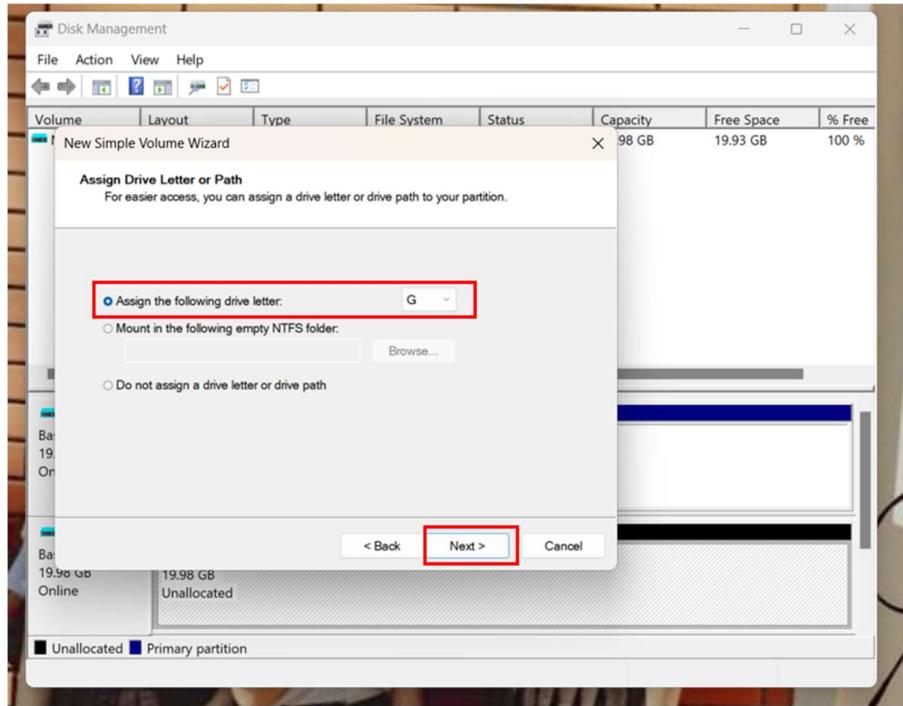
- **B6:** Tại hộp thoại **New Simple Volume Wizard**, chọn Next để tiếp tục thay đổi cài đặt.



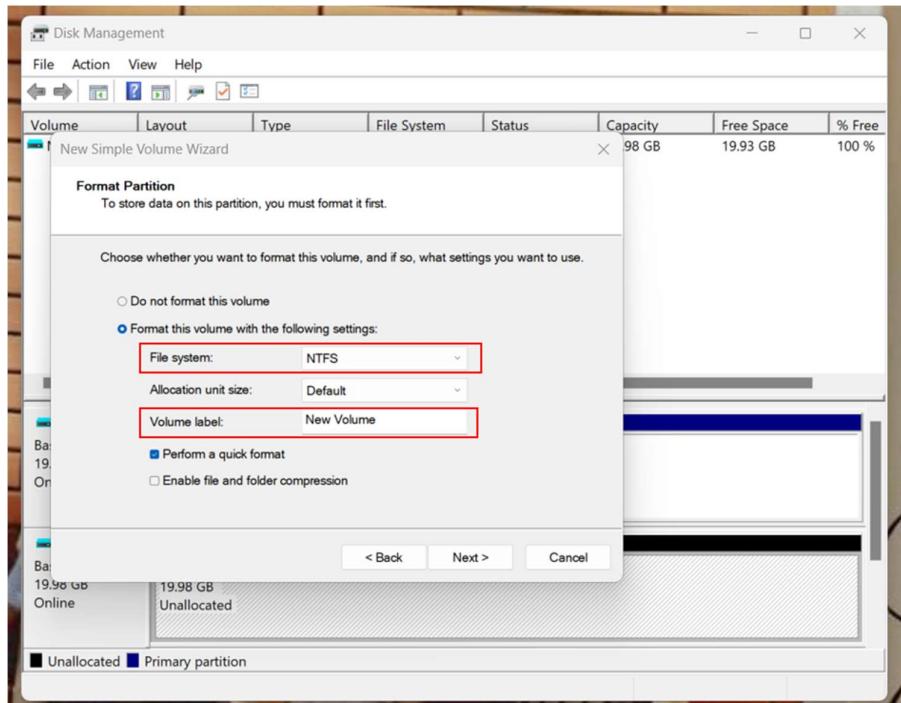
- **B7:** Ở phần **Simple volume size in MB**, chọn kích thước volume phù hợp rồi nhấn Next.



- **B8:** Điều chỉnh chữ cái phù hợp cho Volume ảo ở phần **Assign the following driver letter** và nhấn Next.



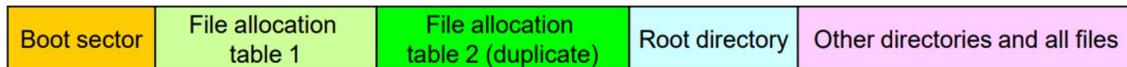
- **B9:** Đặt tên cho volume ảo ở phần **Format this volume with the following settings**, ở mục **File system** chọn NTFS hoặc FAT32 cho ổ đĩa tương ứng muôn tạo và nhấn Next. Cuối cùng nhấn chọn **Finish** để kết thúc cài đặt.



6. Mô tả thuật toán

6.1. FAT32

Cấu trúc của FAT32



PARTITION BOOT SECTOR(PBS)

Đây là một sector đặc biệt nằm ở đầu mỗi partition trên đĩa, chứa thông tin về cấu hình đĩa, kích thước, và loại hệ điều hành được cài đặt cùng với mã lệnh khởi động mới cho hệ điều hành. Để tránh các ứng dụng độc hại như virus hiệu chỉnh nội dung sector này, các máy tính đời mới thường được trang bị BIOS có chức năng bảo vệ boot sector. Khi các ứng dụng cố gắng thay đổi nội dung của boot sector, chúng sẽ phải yêu cầu BIOS làm việc này. Nếu sector bị hiệu chỉnh là boot sector, BIOS sẽ hiển thị thông báo để người dùng quyết định cho phép hoặc cấm chức năng bảo vệ này. Tùy theo nhu cầu sử dụng, người dùng có thể vào BIOS Setup để thay đổi cài đặt bảo vệ boot sector.

Nội dung của boot sector như sau:

Offset	Số byte	Nội dung
0	3	Jump_Code: lệnh nhảy qua vùng thông số (như FAT)
3	8	OEM_ID: nơi sản xuất – version, thường là “MSWIN4.1”
B	2	Số byte trên Sector, thường là 512 (như FAT)
D	1	S_C: số sector trên cluster (như FAT)
E	2	S_B: số sector thuộc vùng Bootsector (như FAT)
10	1	N_F: số bảng FAT, thường là 2 (như FAT)
11	2	Không dùng thường là 0 (số entry của RDET – với FAT)
13	2	Không dùng, thường là 0 (số sector của vol – với FAT)
15	1	Loại thiết bị (F8h nếu là đĩa cứng - như FAT)
16	2	Không dùng, thường là 0 (số sector của bảng FAT – với FAT)
18	2	Số sector của track (như FAT)
1A	2	Số lượng đầu đọc (như FAT)
1C	4	Khoảng cách từ nút mô tả vol đến đầu vol (như FAT)
20	4	S_V: Kích thước volume (như FAT)
24	4	S_F: Kích thước mỗi bảng FAT
28	2	bit 8 bật: chỉ ghi vào bảng FAT active (có chỉ số là 4 bit đầu)
2A	2	Version của FAT32 trên vol này
2C	4	Cluster bắt đầu của RDET
30	2	Sector chứa thông tin phu (về cluster trống), thường là 1
32	2	Sector chứa bản lưu của Boot Sector
34	C	Dành riêng (cho các phiên bản sau)
40	1	Kí hiệu vật lý của đĩa chứa vol (0 : mềm, 80h: cứng)
41	1	Dành riêng
42	1	Kí hiệu nhận diện HĐH
43	4	SerialNumber của Volume
47	B	Volume Label
52	8	Loại FAT, là chuỗi “FAT32”
5A	1A4	Đoạn chương trình khởi tạo & nạp HĐH khi khởi động máy
1FE	2	Dấu hiệu kết thúc BootSector /Master Boot (luôn là AA55h) http://fb.com/tailieuientu

- Máy tính sử dụng boot sector để thực thi các chỉ dẫn trong suốt quá trình khởi động. Quá trình này được mô tả như sau:

- Đầu tiên, BIOS và CPU sẽ thực hiện power-on self test (POST).
- Sau đó, BIOS sẽ tìm một thiết bị khởi động.
- Nếu thiết bị khởi động là ổ cứng, BIOS sẽ nạp Master Boot Record (MBR). Mã lệnh của MBR sẽ nạp boot sector của phân vùng active và chuyển quyền điều khiển cho sector này. Trên Windows 2000, mã thực thi của boot sector sẽ tìm và nạp NTLDR vào bộ nhớ và chuyển quyền điều khiển cho file này.
- Nếu trong ổ đĩa mềm có đĩa có thể khởi động, được định dạng với các tệp hệ thống, thì BIOS sẽ nạp sector đầu tiên (boot sector) của đĩa vào bộ nhớ. Mã lệnh của boot sector sẽ nạp tệp lo.sys - một tệp hệ thống chính của MS-DOS - vào bộ nhớ và chuyển quyền điều khiển cho tệp này.

- Sau khi hệ điều hành được nạp vào bộ nhớ, quyền điều khiển sẽ được chuyển sang hệ điều hành và hệ thống sẽ được điều khiển bởi hệ điều hành đó.

Bảng FAT

Bảng FAT1 và FAT2 chứa thông tin về việc cấp phát và định vị các file, cho phép hệ điều hành truy xuất đến các file một cách chính xác. Bên cạnh đó, bảng này cũng cung cấp thông tin về dung lượng còn trống trên đĩa và đánh dấu các vị trí BAD trên đĩa.

Bảng FAT là một bảng ánh xạ toàn bộ các cluster trên đĩa, nhưng nó chỉ chứa thông tin về vị trí các cluster trên ổ cứng mà không lưu trữ dữ liệu.

Một ví dụ về bảng FAT:

Giá trị	F0	FF	FF	03	40	00	FF	7F	FF	AB	CD	EF
Byte	0	1	2	3	4	5	6	7	8	9	A	B
Giá trị	03 FF FF F0				7F FF 00 40				EF CD AB FF			
Ptử FAT	0				1				2			

Root folder

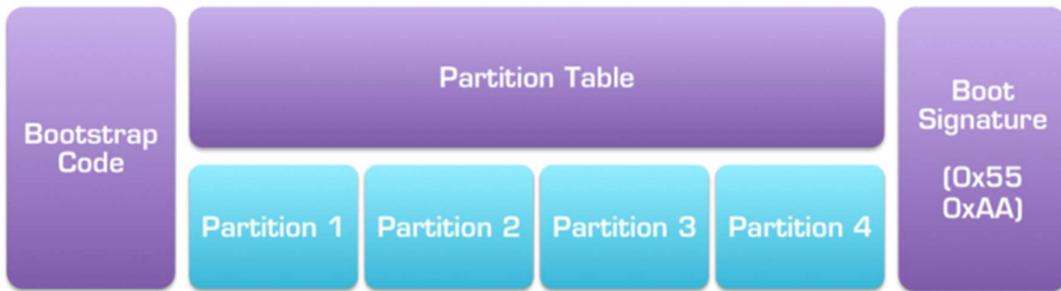
Bảng thư mục gốc là một loại bảng thông tin trong hệ thống tập tin giống như bảng thư mục của một cuốn sách. Nó chứa thông tin liên quan đến các tập tin hoặc thư mục như tên, ngày giờ tạo, thuộc tính và vị trí lưu trữ trên ổ đĩa

Other file or folders

Đây là nơi lưu trữ các file hay thư mục con rong tập tin.

Master Boot Record(MBR)

Master Boot Record



MBR (Master Boot Record) là một phần quan trọng của ổ đĩa cứng, chứa thông tin về các phân vùng trên ổ đĩa. Mỗi hệ điều hành có cách tạo MBR riêng. Ví dụ, trong hệ điều hành WinNT4 và Win2k, MBR tương đương với file boot.ini. Khi khởi động, chương trình bootloader của hệ điều hành sẽ đọc nội dung của MBR để tìm và load hệ điều hành cần thiết cho người dùng.

Tuy nhiên, đối với Windows 98 và Windows ME, chúng không có bootloader như NT bootloader. Thay vào đó, hệ điều hành sẽ mặc định load hệ điều hành đầu tiên trong phân vùng đầu tiên trên ổ đĩa cứng.

Thông thường, MBR được lưu trữ ở đầu ổ đĩa cứng và ở phân vùng nhỏ nhất của nó (phân vùng đầu tiên trên ổ đĩa số 0). MBR rất quan trọng để BIOS tìm đến khi khởi động máy tính.

MBR chứa một phần mồi khởi động cùng với bảng phân vùng (partition table) gồm 4 phần, để lưu trữ thông tin về các phân vùng chính (primary partition) trên ổ đĩa. Thông tin này giúp máy tính xác định được số lượng phân vùng trên ổ đĩa, vị trí và kích thước của chúng.

Bảng thư mục gốc (RDET – Root directory Entry Table)

Trong hệ thống tập tin FAT, có một thông số quan trọng khác liên quan đến cách truy cập vùng FAT (hay các FAT Entry). Mỗi Cluster với số N sẽ có một Entry tương ứng (1-1) được lưu trữ ở đâu đó trong vùng FAT. Với FAT16/FAT32, việc truy cập vùng FAT khá đơn giản, bởi vì FAT Entry được biểu diễn như một mảng 1 chiều các số nguyên 16 bit. Tuy nhiên, so với mảng trên bộ nhớ, FAT Entry có một vài điểm khác biệt. Đầu tiên, các chỉ số không nằm trên một vùng liên tục, ta chỉ có thể chắc chắn từ Sector đầu tiên chứa FAT, thì toàn bộ FAT sẽ nằm trên nhiều Sector liên tục nhau.

Sau đây là dãy các phần tử(entry):

Entry	1	2	...	16	17	18	...	32	33	...	208	209	210	...	224	225	226	...		
Sector	1				2				...				14				...			

Mỗi tập tin hoặc thư mục trong hệ thống tập tin FAT có thể chiếm 1 hoặc nhiều entry. Entry đầu tiên của mỗi tập tin hoặc thư mục cho biết trạng thái hiện tại của entry này, có thể là:

- 0 nếu entry đang trống.
- E5h nếu tập tin hoặc thư mục chiếm entry này đã bị xóa.
- Giá trị khác nếu entry đang chứa thông tin của tập tin hoặc thư mục.

Trong hệ thống tập tin FAT, có hai loại entry bao gồm:

- Entry chính: chứa các thông tin của tập tin hoặc thư mục.
- Entry phụ: chỉ chứa tên của tập tin

Root Directory luôn tồn tại trong ổ đĩa và nằm ở tầng cao nhất trong hệ thống tập tin FAT. Trong FAT12/FAT16, Root Directory nằm ngay sau vùng FAT và có kích thước cố định. Trong khi đó, trong hệ thống tập tin FAT32, cả các tập tin và thư mục con được lưu trữ trong Root Directory

Bảng sau đây chứa các thành phần trong entry chính:

Offset	Bytes	Nội dung
00	8	Tên tập tin
08	3	Phần mở rộng
0B	1	Thuộc tính
1A	2	Cluster bắt đầu
1C	4	Dung lượng tập tin

6.2. NTFS

Tổ chức của ổ đĩa định dạng NTFS

VBR	MFT	Nội dung của tập tin (loại non-resident)	MFT dự phòng	Chưa sử dụng
-----	-----	---	--------------	--------------

PARTITION BOOT SECTOR(PBS)

VBR (Volume Boot Record hay còn gọi là NTFS Partition Boot Sector) là một phần quan trọng trong hệ thống tệp tin NTFS (New Technology File System). Nó đại diện cho sector đầu tiên của một phân vùng NTFS và chứa các thông tin cần thiết để khởi động và điều hướng hệ thống tệp tin NTFS trên phân vùng đó. Nó là một phần quan trọng trong cấu trúc NTFS và đóng vai trò quan trọng trong việc duy trì tính toàn vẹn và khả năng truy cập dữ liệu trong hệ thống tệp tin NTFS.

Phân vùng boot sector NTFS thường được lưu trữ trong sector đầu tiên của phân vùng NTFS và được nạp vào bộ nhớ khi hệ thống khởi động. Nó chứa các thông tin quan trọng như:

- Jump Instruction: Chứa các chỉ thị nhảy (jump instruction) để điều hướng đến mã khởi động (boot code) của phân vùng NTFS.
- OEM Identifier(LONGLONG): Đại diện cho nhãn (OEM identifier) của hệ điều hành sử dụng phân vùng NTFS.
- BIOS Parameter Block (BPB): Lưu trữ thông tin về cấu hình phân vùng NTFS, bao gồm kích thước sector, số sector trên mỗi cụm, số sector ẩn, số cụm trong bảng tệp tin chính (MFT), v.v.
- Master File Table (MFT): Chứa địa chỉ cụm (cluster) đầu tiên của bảng tệp tin chính (Master File Table) và bảng tệp tin chính dự phòng (Mirror Master File Table). MFT lưu trữ thông tin về các tệp tin và thư mục trên phân vùng NTFS.
- Checksum: Giá trị kiểm tra (checksum) được sử dụng để kiểm tra tính toàn vẹn của phân vùng boot sector NTFS.

Các thông tin trên bắt đầu tại offset và có độ dài như sau:

Byte Offset	Field Length	Field Name
0x00	3 bytes	Jump Instruction
0x03	8 bytes	OEM ID
0x0B	25 bytes	BPB
0x24	48 bytes	Extended BPB
0x54	426 bytes	Bootstrap Code
0x01FE	2 bytes	End of Sector Marker

Trong đó BPB (Bios Parameter Block) chứa các thông tin quan trọng của phân vùng gồm 73 bytes(25 bytes BPB và 48 bytes Extended BPB) có cấu trúc như sau:

Địa chỉ (offset)	Kích thước (byte)	Mô tả
0Bh	2	Kích thước một sector. Đơn vị tính là byte.
0Dh	1	Số sector trong một cluster.
0Eh	2	Chưa sử dụng.
10h	1	Với hệ thống NTFS luôn mang giá trị 0.
11h	2	Với hệ thống NTFS luôn mang giá trị 0.
13h	2	Luôn mang giá trị 0, hệ thống NTFS không sử dụng tới trường này.
15h	1	Mã xác định loại đĩa.
16h	2	Với hệ thống NTFS luôn mang giá trị 0.
18h	2	Số sector/track.
1Ah	2	Số mặt đĩa (head hay side).
1Ch	4	Sector bắt đầu của ổ đĩa logic.
20h	4	Luôn mang giá trị 0, hệ thống NTFS không sử dụng tới trường này.
24h	4	Hệ thống NTFS luôn thiết lập giá trị này là “80008000”.
28h	8	Số sector của ổ đĩa logic.
30h	8	Cluster bắt đầu của MFT.
38h	8	Cluster bắt đầu của MFT dự phòng (MFTMirror).
40h	1	Kích thước của một bản ghi trong MFT (MFT entry), đơn vị tính là byte.
41h	3	Luôn mang giá trị 0, hệ thống NTFS không sử dụng tới trường này.
44h	1	Số cluster của Index Buffer.
45h	3	Luôn mang giá trị 0, hệ thống NTFS không sử dụng tới trường này.
48h	8	Số seri của ổ đĩa (volume serial number).
50h	4	Không được sử dụng bởi NTFS.

Như vậy dựa trên nội dung của BPB ta có thể xác định được vị trí cluster bắt đầu của MFT (Master File Table)

MASTER FILE TABLE(MTF)

Master File Table (MFT) là một thành phần quan trọng trong hệ thống tệp tin NTFS (New Technology File System). Nó là một bảng dữ liệu được duy trì bởi NTFS để lưu trữ thông tin về các tệp tin và thư mục trên phân vùng NTFS.

Có ít nhất một mục nhập trong MFT cho mọi tệp trên ổ đĩa NTFS, bao gồm cả chính MFT.

MFT là một cấu trúc dữ liệu có cấu trúc hỗn hợp, được tổ chức dưới dạng các bản ghi (records). Mỗi bản ghi trong MFT đại diện cho một tệp tin, thư mục hoặc một đối tượng hệ thống khác trong hệ thống tệp tin NTFS. MFT bản chất là một tập tin, do vậy cũng có một MFT entry mô tả cho chính nó, đó chính là MFT entry đầu tiên trong MFT, có tên là \$MFT. \$MFT mô tả về kích thước và tổ chức của MFT.

Mỗi bản ghi trong MFT chứa các thông tin quan trọng về một đối tượng như tên, kích thước, quyền truy cập, thời gian tạo và sửa đổi, và các thuộc tính khác của tệp tin hoặc thư mục. Ngoài ra, nó cũng chứa các con trỏ đến vị trí lưu trữ thực tế của dữ liệu tương ứng trong cụm (cluster) trên phân vùng NTFS.

MFT được tổ chức thành các bản ghi có kích thước cố định và được lưu trữ liên tiếp trong phân vùng NTFS. Điều này cho phép hệ thống tệp tin dễ dàng truy cập và quản lý thông tin về các tệp tin và thư mục một cách hiệu quả.

MFT được chia nhỏ thành các phần bằng nhau gọi là MFT entry(record). Kích thước của một MFT entry được quy định trong BPB, thường là 1024 byte.

Cấu trúc của Master File Table (MFT) trong hệ thống tệp tin NTFS (New Technology File System) được tổ chức thành các bản ghi (records) có kích thước cố định. Mỗi bản ghi trong MFT đại diện cho một tệp tin, thư mục hoặc một đối tượng hệ thống khác trong hệ thống tệp tin NTFS.

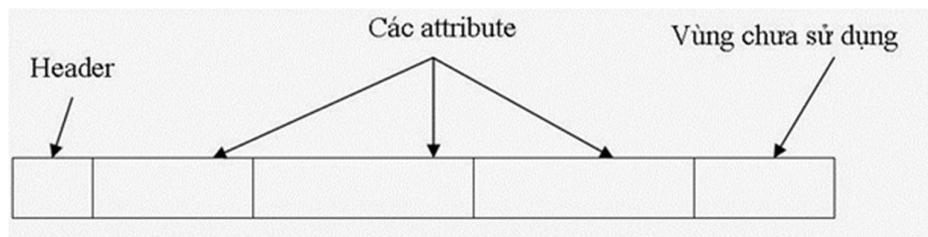
MFT được tổ chức thành các bản ghi (record), và có một số bản ghi đặc biệt trong đó. Bản ghi đầu tiên, được gọi là bản ghi MFT chính, chứa thông tin về chính MFT và là điểm khởi đầu để truy xuất các thông tin khác trong MFT. Một bản sao của MFT cũng được lưu trữ trong 16 bản ghi đầu tiên để đảm bảo tính toàn vẹn của MFT.

Nếu bản ghi MFT chính bị hỏng, hệ thống NTFS sẽ chuyển sang đọc bản ghi tiếp theo, được gọi là Mirror Master File Table (MMFT), để tìm bản sao dự phòng của MFT. MMFT được tạo ra để đảm bảo tính toàn vẹn của MFT trong trường hợp MFT bị hỏng hoặc mất. Vị trí của các phân vùng dữ liệu cho MFT và MMFT được lưu trữ trong boot sector. Một bản sao của boot sector cũng được lưu trữ để đảm bảo tính toàn vẹn của dữ liệu.

Bản ghi thứ ba trong MFT là Log File, được sử dụng để ghi lại các hoạt động thay đổi trong MFT và hỗ trợ khôi phục dữ liệu khi có sự cố xảy ra. Các bản ghi tiếp theo trong MFT (bắt đầu từ bản ghi thứ 17) chứa thông tin về các tệp tin và thư mục trên đĩa, bao gồm các thuộc tính của chúng và vị trí của các phân vùng dữ liệu tương ứng.

Mỗi bản ghi MFT(MFT entry) bao gồm hai phần chính: thông tin cơ bản (header) và các thuộc tính (attributes). Dưới đây là cấu trúc chi tiết của một bản ghi MFT:

Hình dưới đây minh họa tổ chức của một MFT entry, gồm header và ba attribute.



- Header (Thông tin cơ bản): gồm 42 bytes đầu tiên được sử dụng để chứa một số thông tin mô tả cho MFT entry

Offset	Số byte	Mô tả
0x0 – 0x03	4	Dấu hiệu nhận biết MFT entry(Magic number 'FILE')
0x04 – 0x05	2	Địa chỉ (offset) của Update sequence.
0x06 – 0x07	2	Số phần tử của mảng Fixup, mảng này chứa các giá trị bị thay thế trong quá trình thao tác với Update sequence.
0x08 – 0x0F	8	\$LogFile Sequence Number (LSN): mã định danh MFT entry của file log (log record).
0x10 – 0x11	2	Sequence Number: cho biết số lần MFT entry này đã được sử dụng lại. Giá trị này được tăng lên một đơn vị sau mỗi lần tập tin tương ứng với MFT entry này bị xóa. Mang giá trị 0 nếu MFT entry này chưa được sử dụng.
0x12 – 0x13	2	Reference Count: cho biết số thư mục mà tập tin này được hiển thị trong đó, hay nói cách khác là số thư mục tham chiếu đến tập tin này. Trường này còn có tên gọi khác là hard link count.
0x14 – 0x15	2	Địa chỉ (offset) bắt đầu của các attribute.

0x16 – 0x17	2	Flags: - giá trị 0x01: MFT entry đã được sử dụng - giá trị 0x02: MFT entry của một thư mục - giá trị 0x04, 0x08: không xác định
0x18 – 0x1B	4	Số byte đã được sử dụng trong MFT entry.
0x1C – 0x1F	4	Kích thước vùng đĩa đã được cấp cho MFT entry.
0x20 – 0x27	8	Tham chiếu đến MFT entry cơ sở của nó (Base MFT Record). Mang giá trị 0 nếu là MFT entry cơ sở. MFT entry cơ sở dùng để chứa các thông tin về các MFT entry mở rộng (Extension Record).
0x28 – 0x29	2	Next attribute ID: mã định danh của attribute kế tiếp sẽ được thêm vào MFT entry.

- Attributes (Thuộc tính):

Mỗi bản ghi MFT có thể chứa một hoặc nhiều thuộc tính. Mỗi thuộc tính có một loại (type), mô tả các thông tin cụ thể về tệp tin hoặc thư mục. Các loại thuộc tính bao gồm: Standard Information (thông tin tiêu chuẩn), File Name (tên tệp tin), Data (dữ liệu), Security Descriptor (mô tả bảo mật), và nhiều loại thuộc tính khác.

Attribute Header	Attribute Name
0x10	\$STANDARD_INFORMATION
0x30	\$FILE_NAME
0x40	\$OBJECT_ID
0x80	\$DATA

MFT sử dụng phần vùng trống trong các bản ghi để lưu trữ thông tin về các tệp tin và thuộc tính của chúng. Nếu kích thước của một tệp tin nhỏ hơn hoặc bằng 1,5KB thì tệp tin đó có thể được lưu trữ hoàn toàn trong một bản ghi MFT duy nhất. Các tệp tin lớn hơn được lưu trữ trên các vùng dữ liệu riêng biệt và thông tin về tệp tin này được lưu trữ trong các bản ghi MFT tương ứng. Các bản ghi của thư mục cũng được lưu trữ trong MFT, tuy nhiên chúng không chứa dữ liệu của thư mục mà chỉ chứa thông tin về chỉ số của thư mục. Các thư mục có kích thước nhỏ có thể được lưu trữ hoàn toàn trong MFT. Attribute chứa nội dung của một tập tin có thể có kích thước từ vài MB tới hàng GB. Tuy nhiên, kích thước của một MFT entry chỉ là 1024 byte, nên việc chứa toàn bộ nội dung của attribute trong MFT entry là không thực tế. Để giải quyết vấn đề này, hệ thống NTFS cung cấp hai tùy chọn để lưu nội dung của attribute đó là lưu trực tiếp trong MFT entry và lưu ở ngoài MFT entry.

Attribute có phần nội dung được lưu ngay trong MFT entry được gọi là resident attribute (attribute thường trú), thường áp dụng với các attribute có kích thước phần nội dung nhỏ.

Attribute lưu phần nội dung ở các cluster bên ngoài MFT entry được gọi là non-resident attribute (attribute không thường trú).

Trong header của attribute có trường cho biết attribute đó là resident hay non-resident. Nếu attribute thuộc loại resident, phần nội dung sẽ được đặt ngay sau header của attribute, ngược lại, nếu attribute thuộc loại non-resident, header sẽ cung cấp địa chỉ của cluster.

7. Các bước thực hiện: FAT32

Chuẩn bị đối tượng và hàm hỗ trợ

- Chương trình truy cập thông tin của các đối tượng là ổ đĩa và thư mục/tập tin, do đó em sẽ tạo 2 lớp đối tượng là class FAT32 cho đối tượng ổ đĩa và class Component cho đối tượng tập tin/thư mục.
 - Class Component

```
class Component {
private:
    std::string _name;           //ten tap tin/thu muc
    unsigned int _status;         //tinh trang(thu muc = 1, tap tin = 2)
    unsigned int _firstCluster;   //cluster bat dau
    unsigned int _lastCluster;    //cluster ket thuc
    unsigned int _level;          //tap tin/thu muc level 2 nam trong thu muc level 1
                                  //tap tin/thu muc level 3 nam trong thu muc level 2
    unsigned int _size;           //kich thuoc tap tin/thu muc
```

- Class FAT32

```
class FAT32 {
private:
    LPCWSTR _drive;             //ten o dia
    BYTE* BootSector;           //512 byte cua Boot Sector
    BYTE* FAT;                  //bytes cua 1 bang FAT
    std::vector<Component> componentList; //danh sach cac tap tin/thu muc trong o dia

    unsigned int bytes_sector;   //so byte 1 sector
    unsigned int sectors_cluster; //so sector 1 cluster
    unsigned int sectors_bootsector; //so sector cua boot sector
    unsigned int FAT_tables;     //so bang FAT
    unsigned int volume_size;    //kich thuoc o dia
    unsigned int FAT_size;       //kich thuoc 1 bang FAT
    unsigned int first_cluster_RDET; //cluster bat dau cua RDET
```

- Hàm ReadSector() sử dụng source code do thầy cung cấp
- Hàm để lấy giá trị số nguyên ở offset và số byte (number) xác định

```
int Get_Value_At_Offset(BYTE* sector, int offset, int number) {
    int result = 0;
    memcpy(&result, sector + offset, number);
    return result;
}
```

- Hàm lấy chuỗi ký tự ở offset và chiều dài xác định

```
std::string Get_String(BYTE* sector, int offset, int number) {
    std::string result = "";
    for (int i = 0; i < number; i++) {
        if (sector[offset + i] != 0x00 && sector[offset + i] != 0xFF) {
            result += char(Get_Value_At_Offset(sector, offset + i, 1));
        }
    }
    return result;
}
```

Đọc Boot Sector và bảng FAT của FAT32

- Dựa theo mô tả sau để đọc thông tin của Boot Sector

Offset	Số byte	Nội dung
0	3	Jump_Code: lệnh nhảy qua vùng thông số (như FAT)
3	8	OEM_ID: nơi sản xuất – version, thường là “MSWIN4.1”
B	2	Số byte trên Sector, thường là 512 (như FAT)
D	1	S_c: số sector trên cluster (như FAT)
E	2	S_b: số sector thuộc vùng Bootsector (như FAT)
10	1	N_F: số bảng FAT, thường là 2 (như FAT)
11	2	Không dùng, thường là 0 (số entry của RDET – với FAT)
13	2	Không dùng, thường là 0 (số sector của vol – với FAT)
15	1	Loại thiết bị (F8h nếu là đĩa cứng - như FAT)
16	2	Không dùng thường là 0 (số sector của bảng FAT – với FAT)
18	2	Số sector của track (như FAT)
1A	2	Số lượng đầu đọc (như FAT)
1C	4	Khoảng cách từ nơi mô tả vol đến đầu vol (như FAT)
20	4	S_V: Kích thước volume (như FAT)
24	4	S_F: Kích thước mỗi bảng FAT
28	2	bit 8 bật: chỉ ghi vào bảng FAT active (có chỉ số là 4 bit đầu)
2A	2	Version của FAT32 trên vol này
2C	4	Cluster bắt đầu của RDET
30	2	Sector chứa thông tin phụ (về cluster trống), thường là 1
32	2	Sector chứa bản lưu của Boot Sector
34	C	Dành riêng (cho các phiên bản sau)
40	1	Kí hiệu vật lý của đĩa chứa vol (0 : mềm, 80h: cứng)
41	1	Dành riêng
42	1	Kí hiệu nhận diện HĐH
43	4	SerialNumber của Volume
47	B	Volume Label
52	8	Loại FAT, là chuỗi “FAT32”
5A	1A4	Đoạn chương trình khởi tạo & nạp HĐH khi khởi động máy
1FE	CumDongThanCong.com	Dấu hiệu kết thúc BootSector /Master Boot (luôn là AA55h) https://fb.com/tailieuindientucmtt

```

FAT32::FAT32(LPCWSTR drive) {
    _drive = drive;
    BYTE* BootSector = new BYTE[512];
    ReadSector(_drive, 0, BootSector, 512);
    bytes_sector = Get_Value_At_Offset(BootSector, 0x0B, 2);
    sectors_cluster = Get_Value_At_Offset(BootSector, 0x0D, 1);
    sectors_bootsector = Get_Value_At_Offset(BootSector, 0x0E, 2);
    FAT_tables = Get_Value_At_Offset(BootSector, 0x10, 1);
    volume_size = Get_Value_At_Offset(BootSector, 0x20, 4);
    FAT_size = Get_Value_At_Offset(BootSector, 0x24, 4);
    first_cluster_RDET = Get_Value_At_Offset(BootSector, 0x2C, 4);
    FAT = new BYTE[512];
    ReadSector(_drive, sectors_bootsector * bytes_sector, FAT, 512);
    delete[] BootSector;
}

```

- Sau khi nhập tên ổ đĩa, 1 đối tượng FAT32 sẽ được khởi tạo và việc đọc Boot Sector và bảng FAT được thực hiện ngay trong hàm khởi tạo (constructor).

Em để việc đọc FAT vào constructor vì khi đọc hết thông tin Boot Sector thì ta cũng biết được sector đầu tiên của bảng FAT.

- *Sector đầu tiên của FAT = S_B* (vì bảng FAT nằm sau Boot Sector)
- *Sector đầu tiên của vùng DATA = $S_B + N_F * S_F$* (vì trong FAT32 RDET nằm trong vùng DATA và RDET không có kích thước cố định)
- Thông tin về Boot Sector, RDET, cây thư mục và thông tin tập tin sẽ được thể hiện trên màn hình console.

Đọc RDET và SDET

- **Đọc RDET bằng hàm Read_RDET()**
- Sau khi đọc thông tin của Boot Sector, ta có thể xác định cluster bắt đầu của RDET nên từ đó ta có thể xác định sector bắt đầu của RDET:

*sector bắt đầu của RDET = Sector đầu tiên của vùng DATA + (cluster bắt đầu của RDET - 2) * số sector của cluster*

- Trường hợp trong demo, cluster bắt đầu của RDET là 2 nên sector bắt đầu của RDET = sector bắt đầu của vùng DATA. Có được sector bắt đầu của RDET, ta có thể biết được vị trí bắt đầu đọc để sử dụng hàm ReadSector(). Em cài đặt đọc từ ổ đĩa 1 sector/lần đọc và phân tích 32 bytes/lần từ sector được đọc.
- Vòng lặp while() được sử dụng để lặp lại việc phân tích 32 bytes. Các biến để lưu dữ liệu bao gồm

```
//byte đầu tiên của RDET
unsigned int readPoint = (sectors_bootsector + FAT_tables * FAT_size + (first_cluster_RDET - 2) * sectors_cluster) * bytes_sector;
BYTE* RDET = new BYTE[bytes_sector];           //mang cac bytes trong 1 sector
std::string filename = "";                     //ten tap tin/ thu muc
int pCurr = 0;                                //con tro dau moi 32 bytes
ReadSector(_drive, readPoint, RDET, bytes_sector); //doc 1 sector của RDET
```

- Mỗi lần đọc chỉ 1 sector nên ta cần chắc chắn biến pCurr (mỗi cuối vòng lặp sẽ được +32) vẫn chưa vượt ngoài số byte của 1 sector. Nếu pCurr = số byte của 1 sector, đó là lúc cần đọc 1 sector tiếp theo, readPoint và pCurr sẽ được cập nhật lại.

```
if (pCurr == bytes_sector) {
    readPoint += bytes_sector;
    delete[] RDET;
    RDET = new BYTE[bytes_sector];
    ReadSector(_drive, readPoint, RDET, bytes_sector);
    pCurr = 0;
}
```

- Do không biết chính xác kích thước của RDET nên ta chỉ có thể đọc đến khi gặp các dãy byte 0. Vòng lặp sẽ dừng khi byte ở offset 0x0B của 32 bytes là 0x00

```
if (Get_Value_At_Offset(RDET, 0x0B + pCurr, 1) == 0x00)
    break;
```

- Nếu không giá trị tại offset 0x0B khác 0x00 thì ta tiếp tục kiểm tra xem dãy 32 bytes có phải lưu thông tin một tập tin/thư mục hay không. So sánh giá trị tại offset 0x0B lần lượt với 0x0F (entry phụ), 0x10 (thư mục), 0x20(tập tin).
 - Nếu giá trị tại offset 0x0B = 0x0F, tức entry phụ chỉ lưu thông tin tên file, ta cập nhật filename bằng hàm Get_String() tại các offset 0x01(10 bytes), 0x0E (12 bytes), 0x1C (4 bytes) và đọc entry tiếp theo.

```
//entry phu
else if (Get_Value_At_Offset(RDET, 0x0B + pCurr, 1) == 0x0F) {
    filename = Get_String(RDET, pCurr + 0x01, 10)
        + Get_String(RDET, pCurr + 0x0E, 12)
        + Get_String(RDET, pCurr + 0x1C, 4) + filename;
}
```

- Nếu giá trị tại offset 0x0B = 0x10 hay 0x0B = 0x20, tức entry chính, ta đọc các thông tin dựa vào cấu trúc entry chính. Từ đó có được thông tin của 1 đối tượng Component như tên, trạng thái, cluster bắt đầu, cluster kết thúc (sau khi tra bảng FAT), kích thước và level (tập tin trên RDET thì level = 1) và sẽ được thêm vào componentList.
- Khi tra bảng FAT để tìm cluster kết thúc, do đây là cấu trúc FAT32 nên mỗi lần xét 4 bytes đến khi gặp 4 bytes có giá trị 0x0FFFFFFF thì đó là cluster kết thúc.

```
unsigned int first_cluster = Get_Value_At_Offset(RDET, pCurr + 0x14, 2) * pow(16, 4)
    + Get_Value_At_Offset(RDET, pCurr + 0x1A, 2);
unsigned int last_cluster = first_cluster;
unsigned int size = Get_Value_At_Offset(RDET, pCurr + 0x1C, 4);
while (Get_Value_At_Offset(FAT, last_cluster * 4, 4) != 0x0FFFFFFF) {
    last_cluster = Get_Value_At_Offset(FAT, last_cluster * 4, 4);
}
```

- Đọc SDET bằng hàm Read_SDET()**
- Nếu entry lưu thông tin 1 tập tin thì ta tiếp tục xét entry tiếp theo, còn nếu đó là thư mục thì phải đọc SDET để tìm thông tin các tập tin/thư mục con bằng hàm Read_SDET(). Hàm nhận vào đối số là cluster bắt đầu, cluster kết thúc và level của nó

- Từ cluster bắt đầu và cluster kết thúc, ta có thể biết được kích thước cần đọc trên SDET = $(last_cluster - first_cluster + 1) * (sector 1 cluster) * (bytes 1 sector)$
- Cũng giống như RDET, đọc từ ổ đĩa 1 sector/lần đọc và phân tích 32 bytes/lần từ sector được đọc đến khi đọc hết kích thước cần đọc vừa tính được hoặc gấp 1 dãy bytes 0 thì dừng. Các làm và các biến số cần thiết cho vòng while() giống như RDET. Nếu tiếp tục gấp thư mục thì level của tập tin/thư mục con sẽ tiếp tục tăng lên 1 đơn vị và tiếp tục đọc SDET cho đến hết.
- Sau khi đọc xong RDET, ta được componentList hoàn chỉnh lưu danh sách tập tin/thư mục. Giờ chỉ việc gọi in thông tin từng tập tin/thư mục và cây thư mục ra màn hình.

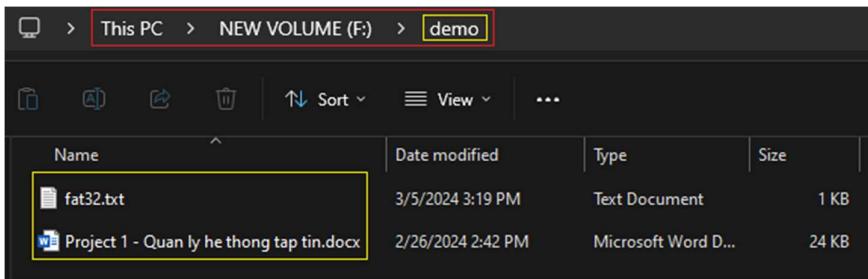
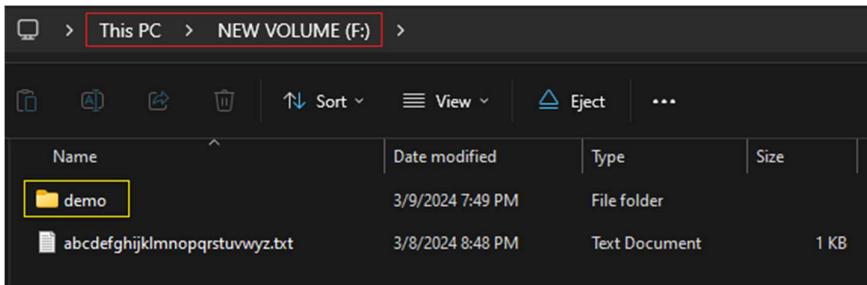
Sau khi có được list các tập tin/thư mục, ta chỉ việc gọi các hàm Print để xuất ra màn hình các thông tin cần thiết

Thông tin tập tin/thư mục (Print_RDET())

- Có được cluster bắt đầu và kết thúc, ta dễ dàng tính được số các sector mà tập tin/thư mục chiếm:
 - $first_sector = S_B + N_F * S_F + (first_cluster - 2) * S_C$
 - $last_sector = S_B + N_F * S_F + (last_cluster + 1 - 2) * S_C - I$
- Các thông tin khác như tên, trạng thái, kích thước đã có sẵn trong mỗi đối tượng Component

Cây thư mục (Print_FolderTree())

- Vì đọc RDET và SDET từ trên xuống và lưu theo level (nghĩa là nếu gấp 1 thư mục thì sẽ lập tức đọc bên trong thư mục trước rồi mới đọc tập tin/thư mục tiếp theo cùng level với nó) nên thứ tự in ra cũng là thứ tự lưu trong componentList. Trong trường hợp demo, các tập tin/thư mục và level của chúng:
 - Level 1: thư mục “DEMO”, tập tin “abcdefghijklmnoprstuvwxyz.txt”
 - Level 2: tập tin “fat32.txt”, tập tin “Project 1 – Quản lý hệ thống tập tin.docx”



Truy xuất thông tin cây thư mục (Print_Data())

- Cũng như in cây thư mục, em sẽ duyệt componentList và xem phần tử là tập tin hay thư mục.
 - TH1: Nếu là tập tin, tiến hành tách extension từ tên file và kiểm tra xem có phải file txt.
 - Nếu là file txt, từ các thông tin của file như cluster bắt đầu và kích thước file, ta dễ dàng tìm đến sector bắt đầu của file và thông tin với kích thước tương ứng vào một chuỗi ký tự. Do đã biết trước kích thước cần đọc nên việc xác định điều kiện dừng vòng lặp dễ hơn so với đọc RDET. Sau khi đọc xong chỉ việc in chuỗi thông tin vừa đọc và tiếp tục với phần tử tiếp theo trong componentList.
 - Nếu không phải file txt, từ extension ta có thể xác định phần mềm tương thích để đọc nội dung file. VD: **Word** đọc file extension *docx*, **PowerPoint** đọc file extension *pptx*, **Excel** đọc file extension *xlsx*, **PDF Reader** đọc file extension *pdf*...
 - TH2: Nếu là thư mục, ta tiến hành in cây thư mục. Thuật toán được sử dụng là in phần tử tiếp theo đến khi hết phần tử trong componentList hoặc gặp phần tử có level bé hơn hoặc bằng level của thư mục hiện tại thì ngừng.

8. Các bước thực hiện: NTFS

8.1. Các file header

- **MFT.h** : chứa các định dạng struct của
 - + Header của MFT entry
 - + Header của MTF attribute

- + Data run của MFT
- + Root index ATTR của MFT
- + Entry index của MFT
- + Block index của MFT
- + Class Filename và ATTRIBUTE
- **Utils.h:** chứa các hàm chức năng cần thiết như
 - + Class Interface của volume
 - + Hàm readfile()
 - + Các hàm convert: DecToDate(), DecToTime(), DecToDate(), ByteToStr(), HexToDec(), StrToDec(), DataSizeFormat(), IsQuals()
 - + Các hàm check entry: isDelete(), isFolder(), isFile(), isSUB_ENTRY(), isDot(), isHiddenEntry(), isVOLUME_ENTRY().
- **PartitionBootSector.h:**

- + Struct PBS(Partition Boot Sector) của NTFS

```
typedef struct PBS { ... };
```

Cấu trúc này chứa các trường dữ liệu như jmpBoot, oemID, bytePerSec, secPerClus, v.v. Mỗi trường dữ liệu trong cấu trúc được khai báo với kiểu dữ liệu tương ứng và có tên. Cấu trúc này được sử dụng để lưu trữ thông tin về một phân vùng hệ thống tệp tin NTFS.

- + Class NTFSPBSector

```
class NTFSPBSector { ... };
```

Lớp này có một thành viên dữ liệu là pbs kiểu PBS, và các phương thức thành viên như Read, getFstMTFSec, getBytePerSec, getSecPerClus, v.v. Lớp này được sử dụng để thực hiện các hoạt động liên quan đến sector NTFS, bao gồm đọc dữ liệu, truy xuất thông tin và in ra thông tin.

- **Volume.h:**
 - + Các class về Entry của NTFS
 - + Class định dạng FILE và FOLDER
 - + Class định dạng Volume

8.2. Các file .cpp và phần thực thi

- **Utils.cpp**

```
void readFile(char* buffer, BYTE byte[], int size)
```

Hàm đọc ‘size’ bytes từ ‘byte’ truyền vào ‘bufer’.

```
string ByteToString(BYTE byte[], int size)
```

Hàm chuyển từ byte sang chuỗi.

```
JINT32 HexToDec(string hexSector, bool mode)
```

Hàm trả về giá trị int ở cơ số 10, hỗ trợ ở việc xuất thông tin Volume.

```
UINT StringToDec(BYTE str[], int size, bool mode)
```

Hàm chuyển từ chuỗi sang int hệ cơ số 10.

```
string DataSizeFormat(UINT64 size)
```

Hàm định dạng cho data.

- nếu ‘size’ < 1024 thì định dạng là “B” (byte).
- nếu $1024 < \text{size} < 1024^2$ thì đổi từ “B” sang “KB”(kilo byte).
- nếu $1024^2 < \text{size} < 1024^3$ thì đổi từ “B” sang “MB”(mega byte).
- nếu $1024^3 < \text{size}$ thì đổi từ “B” sang “GB”(giga byte).

```
bool Equals(wstring str1, wstring str2)
```

Hàm so sánh hai chuỗi str1 và str2, nếu str1 = str2 thì trả về True, ngược lại là False.

```
string DecToDateTme(ULLONG time)
```

Hàm chuyển đổi một số thời gian đại diện dưới dạng ULLONG (unsigned long long) thành một chuỗi đại diện cho ngày và thời gian tương ứng. Chương trình sử dụng hàm `localtime_s` để chuyển đổi thời gian số được tính từ một ngày cụ thể (1 tháng 1 năm 1601) thành cấu trúc `tm` (thời gian cục bộ). Sau đó, nó sử dụng các thành phần của cấu trúc `tm` để tạo ra một chuỗi đại diện cho ngày và thời gian.

```
string DectoTime(WORD Dec)
```

Hàm này chuyển đổi một số được biểu diễn dưới dạng **WORD** thành một chuỗi đại diện cho thời gian (giờ:phút:giây). Cụ thể, nó trích xuất các bit tương ứng để lấy ra giờ, phút và giây, sau đó tạo thành chuỗi kết quả dựa trên các giá trị này, cụ thể là:

- 5 bit đầu lấy giây.
- 6 bit tiếp theo lấy phút.

- 5 bit cuối lấy giờ.

`string DectoDate(WORD Dec)`

Hàm này chuyển đổi một số được biểu diễn dưới dạng **WORD** thành một chuỗi đại diện cho ngày (ngày/tháng/năm). Cụ thể, nó trích xuất các bit tương ứng để lấy ra giờ, phút và giây, sau đó tạo thành chuỗi kết quả dựa trên các giá trị này, cụ thể là:

- 5 bit đầu để lấy ngày.
- 4 bit tiếp theo để lấy tháng.
- 7 bit tiếp theo để lấy năm.

- **MFT.cpp:**

`void INDEX_ROOT_ATTRI::read(BYTE sec[])`

Hàm này có nhiệm vụ đọc dữ liệu từ một sector được truyền vào, chủ yếu là các MFTEntryIndex và FILENAME từ index root attribute của một file trong hệ thống tập tin. Đọc các thông tin này và tạo ra các BLOCKFILE từ chúng, sau đó thêm BLOCKFILE vào danh sách.

`void MFT_RECORD::read(BYTE sec[])`

Hàm này có nhiệm vụ đọc dữ liệu từ một sector chứa MFT Record. Nó đọc thông tin cơ bản của MFT Record, kiểm tra tính hợp lệ của nó, sau đó đọc các MFT Attribute từ danh sách thuộc tính của MFT Record và tạo các đối tượng thuộc tính tương ứng. Cuối cùng, nó thêm các đối tượng thuộc tính vào danh sách thuộc tính của MFT Record.

`DATARUNLIST MFT_RECORD::getDataRun_INDEX()`

Hàm này tìm kiếm trong danh sách các thuộc tính của MFT Record để tìm thuộc tính có kiểu là INDEX_ALLOC_ATTRI. Nếu tìm thấy, nó trả về danh sách Data Run của thuộc tính đó. Nếu không tìm thấy hoặc danh sách thuộc tính rỗng, nó trả về một danh sách Data Run trống.

`DATARUNLIST MFT_RECORD::getDataRun_DATA()`

Hàm này tìm kiếm trong danh sách các thuộc tính của MFT Record để tìm thuộc tính có kiểu là DATA_ATTRIBUTE. Nếu tìm thấy, nó lấy danh sách Data Run

từ thuộc tính đó và thêm vào danh sách Data Run sẽ được trả về. Nếu không tìm thấy hoặc danh sách thuộc tính rỗng, nó trả về một danh sách Data Run trống.

`LIST_INDEX MFT_RECORD::getIndexEntries()`

Hàm này tìm kiếm trong danh sách các thuộc tính của MFT Record để tìm thuộc tính có kiểu là INDEX_ROOT_ATTRI. Nếu tìm thấy, nó trả về danh sách Entry Index của thuộc tính đó. Nếu không tìm thấy hoặc danh sách thuộc tính rỗng, nó trả về một danh sách Entry Index trống. Hàm này tìm kiếm trong danh sách các thuộc tính của MFT Record để tìm thuộc tính có kiểu là INDEX_ROOT_ATTRI. Nếu tìm thấy, nó trả về danh sách Entry Index của thuộc tính đó. Nếu không tìm thấy hoặc danh sách thuộc tính rỗng, nó trả về một danh sách Entry Index trống.

`DATA_ATTRIBUTE* MFT_RECORD::getDataAttr()`

Hàm này tìm kiếm trong danh sách các thuộc tính của MFT Record để tìm thuộc tính có kiểu là DATA_ATTRIBUTE. Nếu tìm thấy, nó trả về con trỏ đến DATA_ATTRIBUTE. Nếu không tìm thấy hoặc danh sách thuộc tính rỗng, nó trả về nullptr.

`vector <MFTDatarun> readDataRun(BYTE datarun[], int size)`

Hàm này nhận một mảng byte datarun[] và kích thước của nó size. Nó đọc các thông tin Data Run từ mảng byte và chuyển đổi chúng thành các đối tượng MFTDatarun, sau đó thêm chúng vào một vector và trả về vector đó.

`void INDEX_ALLOC_ATTRI::read(BYTE sec[])`

Hàm này đọc thông tin cơ bản của INDEX_ALLOC_ATTRI từ một sector (64 bit của sector đó), sau đó lấy offset của dữ liệu đầu tiên trong INDEX_ALLOC_ATTRI và chiều dài của bản ghi. Sau đó, nó gọi hàm readDataRun để đọc và chuyển đổi Data Run từ sector và gán cho danh sách Data Run của INDEX_ALLOC_ATTRI.

`void FILENAME::read(BYTE sec[])`

Hàm này đọc thông tin về tên tập tin từ một sector. Nó đọc thông tin cơ bản của FILENAME từ sector, sau đó đọc tên tập tin từ sector và loại bỏ các ký tự không in được, sau đó lưu trữ tên tập tin kết quả vào biến entryName, cụ thể là:

- Đọc thông tin cơ bản của FILENAME của một sector.

- Đọc tên tập tin của sector, từ offset thứ 66 (phải đọc length*2 byte vì mỗi ký tự unicode chiếm 2 byte)

```
void DATA_ATTRIBUTE::read(BYTE sec[])
```

Hàm này đọc thông tin cơ bản của DATA_ATTRIBUTE từ một sector. Nếu dữ liệu là resident, nó đọc dữ liệu trực tiếp từ sector và chuyển đổi thành chuỗi. Nếu dữ liệu là non-resident, nó đọc thông tin cơ bản của DATA_ATTRIBUTE một lần nữa từ sector, sau đó đọc và chuyển đổi Data Run từ sector và gán cho danh sách Data Run của DATA_ATTRIBUTE.

- **Volume.cpp:**

```
int VOLUME_NTFS::InitVolume(LPCWSTR drive)
```

Hàm InitVolume khởi tạo một đối tượng VOLUME_NTFS để thao tác với ổ đĩa hệ thống NTFS. Với tham số truyền vào là drive - đường dẫn tới ổ đĩa cần mở. Khởi tạo một mảng byte có kích thước 512 byte để lưu thông tin một sector trên ổ đĩa. Sử dụng hàm OpenVolume để tiến hành mở ổ đĩa có đường dẫn được truyền vào hàm với tham số drive và lưu kết quả vào biến status. Nếu kết quả trả về khác 0 hàm trả về -1 để báo lỗi. Đọc sector đầu tiên trên ổ đĩa và lưu dữ liệu vào mảng sector bằng hàm ReadSector(0, sector). Tiếp đó, sử dụng _PBSector.Read(sector) để đọc dữ liệu từ mảng sector vào đối tượng _PBSector (Partition Boot Sector) và _PBSector.getFileSystem() để lấy thông tin về hệ thống tệp của ổ đĩa và kiểm tra xem có phải NTFS không, nếu không phải, hàm trả về 0 để thông báo ổ đĩa không phải NTFS. Sau đó, hàm lấy thông số cấu hình của hệ thống NTFS và lưu vào các biến tương ứng (_secPerClus, _bytePerSec, _fstMTF, _bytePerRecord). Biến _rootEntry được khởi tạo như một thư mục gốc, sau đó, ta xác định sector của thư mục gốc bằng câu lệnh '_rootEntry->sector = _fstMTF + ROOT_FILE_NAME_INDEX * 2;'. Sử dụng hàm readDirectoryTree(_rootEntry) để đọc cấu trúc thư mục bắt đầu từ thư mục gốc. Nếu kết thúc hàm thành công, hàm sẽ trả về 1 để báo hiệu việc khởi tạo ổ đĩa NTFS đã hoàn thành.

```
int VOLUME_NTFS::ReadSector(UINT64 readPoint, BYTE sector[], UINT nByte)
```

Hàm ReadSector được sử dụng để đọc dữ liệu từ 1 sector trên ổ đĩa NTFS và lưu dữ liệu vào mảng byte. Khai báo biến nReadBytes lưu trữ số lượng byte thực sự được đọc từ ổ đĩa. Biến highOffset = readPoint >> 32 để xác định offset cao trong trường hợp readPoint vượt quá 32 bit của một DWORD. Sử dụng hàm

SetFilePointer để đặt con trỏ đọc tại vị trí được chỉ định bởi readPoint. Hàm này đảm bảo rằng dữ liệu được đọc sẽ bắt đầu từ sector cần đọc. Gọi hàm ReadFile để đọc dữ liệu từ ổ đĩa và lưu trữ vào mảng sector, kiểm tra xem nếu đọc dữ liệu không thành công thì in ra thông báo lỗi bằng cách sử dụng hàm GetLastError() và thoát chương trình, ngược lại nếu đọc thành công hàm trả về số byte thực sự đã đọc từ ổ đĩa.

```
void VOLUME_NTFs::printPartitionBootSector()
```

Hàm printPartitionBootsector() được sử dụng để in ra thông tin của Partition Boot Sector của ổ đĩa NTFS bằng cách gọi hàm _PBSector.printBS().

```
void VOLUME_NTFs::readIndexEntriesFromMFTRecord(MFT_RECORD& mftRecord, ENTRY*& folder, ENTRY*& entry)
```

Hàm readIndexEntriesFromMFTRecord có nhiệm vụ đọc các chỉ mục từ một bản ghi Master File Table cụ thể và xây dựng cấu trúc cây thư mục tương ứng. Sử dụng hàm getIndexEntries() để lấy danh sách các chỉ mục từ bản ghi MFT được đưa vào dưới dạng đối tượng MFT_RECORD. Chạy vòng lặp qua danh sách các chỉ mục đã lấy ra từ bản ghi MFT, kiểm tra xem chỉ mục hiện tại có phải là thư mục không bằng cách sử dụng cờ ATTR_FILENAME_FLAG_ENTRYCTORY (0x10000000). Nếu đúng, tạo một đối tượng ‘FOLDER’ mới cho thư mục được tìm thấy trong chỉ mục hiện tại bằng câu lệnh ‘entry = new FOLDER(list[i].curIndex);’. Gọi đệ quy hàm readDirectoryTree để đọc cấu trúc thư mục con hiện tại. Ngược lại, nếu sai kiểm tra xem chỉ mục hiện tại có phải là một tập tin hay không bằng cách tra các thuộc tính của tên tập tin (fname) bằng cách sử dụng cờ ‘ATTR_FILENAME_FLAG_ARCHIVE’ (0x00000020). Khởi tạo một đối tượng ‘FILE_NTFS’ mới cho tập tin được tìm thấy trong chỉ mục hiện tại. Cuối cùng, kiểm tra xem entry có khác NULL không, nếu khác tiến hành đặt tên tập tin hay thư mục cho đối tượng được tạo bằng hàm setFileName, xác định sector của đối tượng vừa được tạo. Thêm đối tượng được tạo vào thư mục cha (‘folder’) bằng hàm addEntry.

```
void VOLUME_NTFs::readIndexEntriesFromIndexAllocationAttribute(MFT_RECORD& mftRecord,  
BYTE*& byte, ENTRY*& folder, ENTRY*& entry)
```

Hàm `readIndexEntriesFromIndexAllocationAttribute` được sử dụng để đọc các chỉ mục từ thuộc tính Index Allocation trong một bản ghi MFT và xây dựng cấu trúc cây thư mục tương ứng. Gọi hàm `getDataRun_INDEX()` để lấy danh sách các datarun từ bản ghi MFT được đưa vào dưới dạng đối tượng ‘`MFT_RECORD`’. Lặp qua danh sách các datarun, cập nhật giá trị offset cho mỗi datarun. Tính toán vị trí bắt đầu của datarun dựa trên offset và kích thước sector và cluster bằng câu lệnh ‘`pos = offset * _secPerClus * _bytePerSec`’. Gọi hàm `ReadSector(pos, byte, 1024)` để đọc dữ liệu từ sector bắt đầu từ vị trí pos vào mảng byte với kích thước 1024 byte. Gọi hàm `readFile` để đọc dữ liệu của 1 khối chỉ mục từ dữ liệu đã đọc được và lưu vào biến `IndexBlock`. Đọc dữ liệu danh sách Update Sequence từ dữ liệu đọc được và lưu vào mảng `LIST_US`. Kiểm tra xem `_entryIndex` hiện tại có phải chỉ mục cuối không. Nếu không, sử dụng hàm `read` để đọc thông tin về tập tin từ dữ liệu được lưu vào `fname`. Tạo một đối tượng entry mới dựa trên loại của tập tin hoặc thư mục và thêm nó vào thư mục cha (folder). Dịch vị trí để tiếp tục đọc các chỉ mục tiếp theo bằng lệnh `pos2 += _entryIndex.Length`. Lặp lại quá trình trên cho tất cả các chỉ mục trong datarun hiện tại và tất cả các datarun.

```
void VOLUME_NTFS::readDirectoryTree(ENTRY* folder)
```

Hàm `readDirectoryTree` được sử dụng để đọc cấu trúc cây thư mục từ một thư mục cụ thể (folder) trong hệ thống tập tin NTFS. Khởi tạo một mảng byte có kích thước 2048 byte để lưu trữ dữ liệu đọc từ ổ đĩa. Tính toán vị trí bắt đầu của thư mục trong bản ghi MFT dựa trên sector đầu tiên của MFT và index của thư mục qua câu lệnh ‘`pos = (_fstMTF * 512 + 1024 * folder->_entryIndex)`’. Khởi tạo một con trỏ ‘`entry`’ bằng `NULL` để lưu trữ thư mục hoặc tập tin được tạo ra từ các chỉ mục. Gọi hàm `ReadSector(pos, byte, 1024)` để đọc dữ liệu từ sector bắt đầu ở vị trí pos vào mảng byte ‘`byte`’ với kích thước 1024 byte. Tạo một đối tượng ‘`MFT_RECORD`’ và đọc dữ liệu từ mảng byte vào đối tượng này thông qua hàm `read`. Gọi hàm `readIndexEntriesFromRecord` để đọc các mục chỉ mục từ MFT và xây dựng cấu trúc cây thư mục tương ứng. Gọi hàm `readIndexEntriesFromIndexAllocationAttribute` để đọc các mục chỉ mục từ thuộc tính Index Allocation trong MFT và xây dựng cấu trúc cây thư mục tương ứng. Cuối cùng giải phóng bộ nhớ của mảng byte sau khi sử dụng xong bằng lệnh `delete[]` byte.

```
void VOLUME_NTFS::printDirectoryTree()
```

Hàm printDirectoryTree được sử dụng để in ra cấu trúc cây thư mục bắt đầu từ thư mục gốc (_rootEntry).

```
void VOLUME_NTFS::printDetailedDirectoryTree()
```

Hàm printDetailedDirectoryTree in ra thông tin chi tiết về cấu trúc cây thư mục bắt đầu từ thư mục gốc (_rootEntry) bằng cách gọi hàm _rootEntry.printEntryInfo().

```
void VOLUME_NTFS::printEntryData(wstring filename)
```

Hàm printEntryData được sử dụng để in ra thông tin về một tập tin hoặc thư mục cụ thể trong cấu trúc cây thư mục, nếu đó là một tập tin, cũng in ra nội dung của tập tin đó. Gọi hàm SearchEntry để tìm kiếm mục với tên được chỉ định trong cây thư mục bắt đầu từ thư mục gốc. Kiểm tra xem mục đó có được tìm thấy hay không. Nếu không, in ra thông báo lỗi và thoát khỏi hàm. Ngược lại, kiểm tra xem mục là tập tin hay thư mục. Nếu đó là tập tin, gọi hàm printEntryInfo để in thông tin chi tiết về tập tin và hàm printFileContent để in ra nội dung của tập tin đó. Nếu đó là thư mục, in ra cây thư mục và thông tin chi tiết về thư mục đó.

```
void VOLUME_NTFS::printFileContent(ENTRY* entry, wstring filename)
```

Hàm printFileContent được sử dụng để in ra nội dung của một tập tin NTFS nếu tập tin có định dạng .txt. Hàm khởi tạo một mảng byte có kích thước 1024 byte để lưu dữ liệu sector đọc từ ổ đĩa. Sử dụng hàm getSize() để lấy kích thước tập tin. Tính toán vị trí bắt đầu của tập tin trong bản ghi MFT dựa trên sector bắt đầu cả MFT và index của tập tin bằng câu lệnh ‘pos = (_fstMFT * 512 + 1024 * entry->_entryIndex)’. Sau đó đọc dữ liệu từ sector có vị trí pos vào mảng byte với kích thước 1024 byte. Khởi tạo biến mft_record, gọi hàm read để đọc dữ liệu từ mảng byte và lưu vào biến này. Gọi hàm getDataAttr để lấy thông tin thuộc tính dữ liệu của tập tin đối tượng mft_record. Dùng hàm transform, chuyển tên tập tin thành chữ in thường để kiểm tra xem có phải tập tin .txt không. Nếu không, in ra thông báo lỗi và thoát khỏi hàm. Kiểm tra xem tập tin có chứa dữ liệu không, nếu không in ra thông báo “No data！”, ngược lại nếu tập tin có dữ liệu, kiểm tra xem dữ liệu có được lưu dưới dạng datarun không, nếu không, in ra dữ liệu tập tin, nếu có, sử dụng vòng lặp để đọc dữ liệu từ các datarun và in ra nội dung tập tin.

```
void ENTRY::printEntryInfo(int x)
```

Hàm printEntryInfo để in thông tin của entry như name, attribute, create, access, modified, file size, entry index, sector thông qua các hàm getName, getAttribute, getCreateTime, getAccessTime, getModifiedTime, getSize, ...

```
void FILE NTFS::printDirectoryTree(int x)
```

Hàm printDirectoryTree(int x) được sử dụng để in ra thông tin của một tập tin NTFS trong cấu trúc cây thư mục. Sử dụng hàm getName để lấy tên tập tin, setw(x) để căn chỉnh khoảng cách từ từ bên trái của đầu dòng in, điều này giúp dễ dàng xem cấu trúc cây thư mục. Cuối cùng gọi hàm DataSizeFormat để in ra kích thước tập tin.

```
ENTRY* FILE_NTFS::SearchEntry(wstring name)
```

Hàm SearchEntry để tìm kiếm một tập tin trong cấu trúc cây thư mục bằng tên tập tin đó.

```
void FOLDER::printDirectoryTree(int x)
```

Hàm printDirectoryTree ở đây được sử dụng để in ra cấu trúc cây thư mục con, trong đó mỗi mục con (thư mục hay tập tin) sẽ được in ra trên một dòng riêng, các mục con sẽ được căn lề so với mục cha của chúng. Đôi số x được sử dụng để xác định mức độ căn lề của mỗi dòng in. Dòng đầu tiên của mỗi thư mục con sẽ bắt đầu bằng ký tự 192 trong bảng mã ASCII, theo sau là tên của mục đó.

```
void FOLDER::printEntryInfo(int x)
```

Hàm printEntryInfo cũng in ra thông tin về thư mục và các mục con của nó trong cây thư mục. Mỗi dòng thông tin sẽ bao gồm tên của mục, thuộc tính của mục, index đầu tiên của mục trong cấu trúc và sector MFT entry. Các thông tin này cũng được căn lề so với cha của chúng. Sau đó, hàm gọi đệ quy để in ra thông tin của các mục con và mỗi mục con sẽ được căn lề thêm 5 khoảng trắng so với mục cha của nó.

```
ENTRY* FOLDER::SearchEntry(wstring name)
```

Hàm SearchEntry trong class FOLDER được sử dụng để tìm kiếm một mục (thư mục hoặc tập tin) trong cấu trúc cây thư mục bằng tên mục. Sử dụng hàm Equals để so sánh tên được truyền vào với tên của thư mục đang xử lý có trùng khớp không. Nếu trùng khớp, tức là tìm thấy mục cần tìm. Lấy số lượng mục con của thư mục hiện tại bằng hàm size(). Duyệt qua danh sách các mục

con của thư mục, gọi phương thức SearchEntry trên từng thư mục con. Nếu không tìm thấy tên trùng khớp với tên của thư mục hiện tại, hàm trả về nullptr, cho biết không tìm thấy mục trong cây thư mục.

```
void InputVolume(VOLUME*& volume, wchar_t& name)
```

Hàm InputVolume được sử dụng để nhập và khởi tạo một đối tượng VOLUME từ ổ đĩa được chỉ định. Đầu tiên, hàm kiểm tra xem đối tượng volume đã được tạo trước đó chưa, nếu tồn tại, giải phóng bộ nhớ và gán volume thành nullptr. Khởi tạo một mảng kí tự driveName với địa chỉ ổ đĩa mặc định là E: . Vòng lặp while được sử dụng để liên tục nhập và kiểm tra tên ổ đĩa cho đến khi người dùng nhập 0 để thoát chương trình. Hàm yêu cầu người dùng nhập tên ổ đĩa và lưu vào biến name. Kiểm tra xem nếu người dùng nhập 0 thì thoát khỏi chương trình và giải phóng bộ nhớ của volume. Gán ký tự tên ổ đĩa được nhập vào phần tử thứ 4 của mảng driveName, khởi tạo một đối tượng VOLUME_NTFS mới. Gọi hàm InitVolume trên đối tượng volume với tham số truyền vào là tên ổ đĩa và lưu giá trị trả về vào biến status. Nếu status != 1, in ra thông báo lỗi và yêu cầu người dùng nhập lại. Nếu không có lỗi, thoát khỏi vòng lặp.

```
void Menu(int& choice, wchar_t name)
```

Hàm Menu để in ra menu của chương trình hệ thống quản lý tập tin NTFS.

- **PartitionBootSector.cpp:**

```
void NTFSPBSector::Read(BYTE sec[512])
```

Hàm là một phương thức thuộc class NTFSPBSector được sử dụng để đọc dữ liệu từ một sector NTFS và lưu trữ vào một mảng sec có kích thước 512 byte và lưu trữ dữ liệu đã đọc được vào đối tượng PBS pbs của class trên với struct PBS chứa các trường dữ liệu tương ứng với thông tin trong Partition Boot Sector (PBS) của hệ thống tệp NTFS(chi tiết ở PartitionBootSector.h). Cụ thể, hàm gọi hàm readFile((char*)&pbs, sec, 512) để thực hiện việc đọc dữ liệu từ sector và lưu trữ vào mảng sec và biến pbs(chi tiết ở Utils.cpp). Khi đó, nội dung của sector sẽ được sao chép vào vùng nhớ tương ứng với biến pbs và có thể truy cập các trường dữ liệu trong biến pbs để sử dụng thông tin đã được đọc.

```
void NTFSPBSector::printBS()
```

NTFSPBSector::printBS() là một phương thức thuộc class NTFSPBSector được sử dụng để in ra các thông tin trong Partition Boot Sector (PBS) của hệ thống

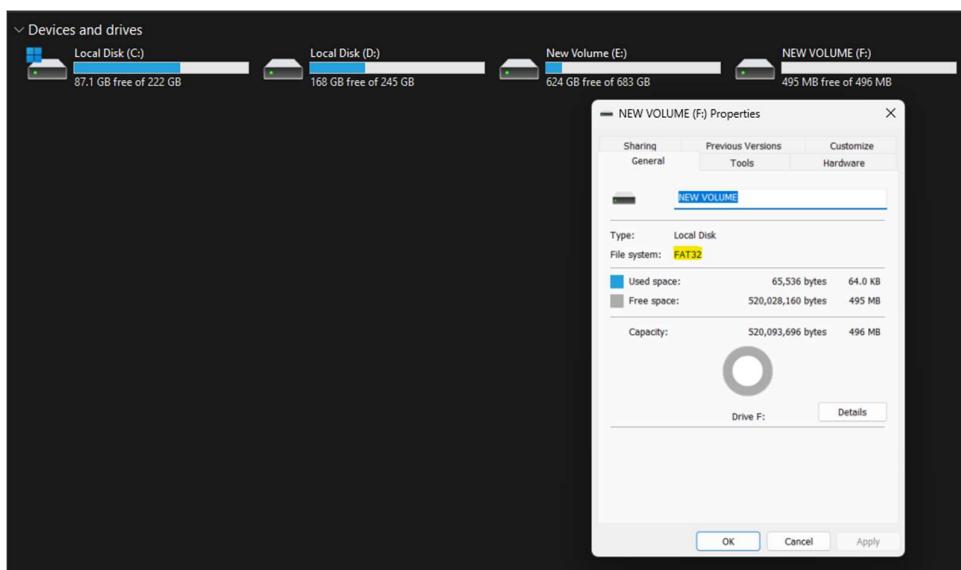
tệp NTFS. PBS là một cấu trúc dữ liệu đặc biệt trong NTFS chứa thông tin cần thiết để khởi động hệ thống tệp. Các lệnh cout được sử dụng để in ra các thông tin của PBS như OEM Name, Bytes Per Sector, Sectors Per Cluster, Sectors Per Track, Number of Heads, Total sectors in volume, Starting Cluster of MTF, Starting Cluster of MFTMirror, MTF Entry Size, Number of Bytes in Index Block.

9. Demo chương trình

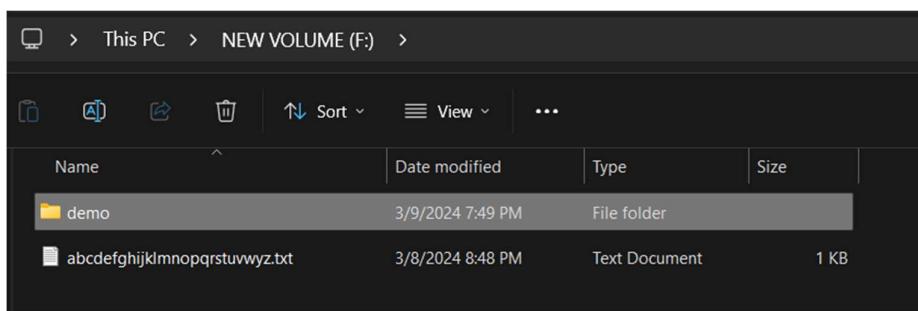
9.1. FAT32

Chuẩn bị ổ đĩa, thư mục và tập tin cho demo

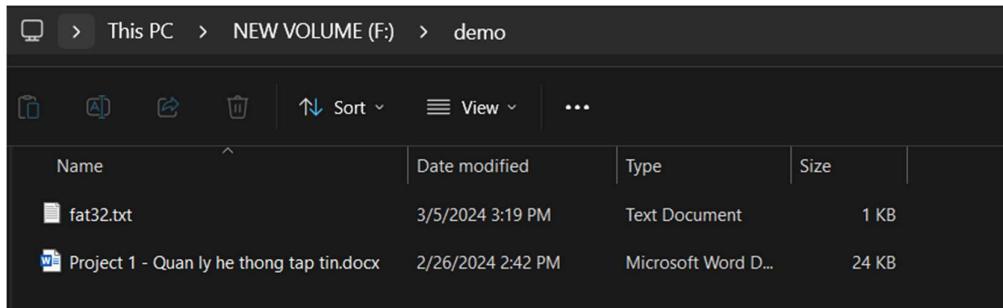
- Tạo ổ đĩa ảo F (FAT32)



- Các thư mục và tập tin được lưu trong ổ F

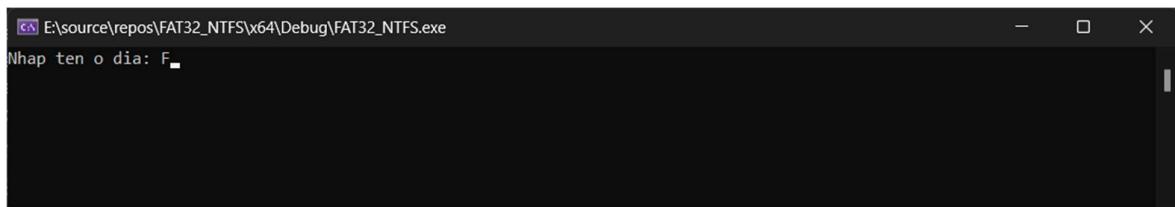


- Các tập tin/thư mục trong thư mục demo



Chạy Demo

- Khi bắt đầu chạy, chương trình yêu cầu nhập tên ổ đĩa cần đọc thông tin



Thông tin Boot Sector

Name	Offset	Value
JMP instruction	000	EB 58 90
OEM ID	003	MSDOS5.0
BIOS Parameter Block	011	
Bytes per sector	011	512
Sectors per cluster	013	8
Reserved sectors	014	6,206
Number of FATs	016	2
(unused)	017	00 00
(unused)	019	00 00
Media descriptor	021	0xF8
(unused)	022	00 00
Sectors per track	024	63
Number of heads	026	255
Hidden sectors	028	32,768
Total sectors	032	1,024,000
Sectors per FAT	036	993
Extended flags	040	0
Version	042	0
Root cluster	044	2
System Information...	048	1
Backup Boot sector	050	6
(reserved)	052	00 00 00 ...

Microsoft Visual Studio Debug Console

```
Nhap ten o dia: F
-----BOOT SECTOR-----
Loai FAT: FAT32
So byte cho 1 sector: 512
So sector cho 1 cluster: 8
So sector tren Bootsector: 6206
So bang FAT: 2
Tong so sector tren dia: 1024000
So sector 1 bang FAT: 993
Sector dau tien cua bang FAT: 6206
Sector dau tien cua vung DATA: 8192
Cluster bat dau cua RDET: 2
```

Sector đầu tiên của FAT = S_B (vì bảng FAT nằm sau Boot Sector)

*Sector đầu tiên của vùng DATA = $S_B + N_F * S_F$ (vì trong FAT32 RDET nằm trong vùng DATA và RDET không có kích thước cố định)*

Thông tin tập tin/thư mục

```

-----THÔNG TIN CÁC TẬP TIN-----
Ten: DEMO
Trang thái: Thủ mục
Chiếncac cluster: 5
Tuong ung cac sector: 8216, 8217, 8218, 8219, 8220, 8221, 8222, 8223
Kich thuoc: 0

Ten: FAT32 TXT
Trang thái: Tập tin
Chiếncac cluster: 6
Tuong ung cac sector: 8224, 8225, 8226, 8227, 8228, 8229, 8230, 8231
Kich thuoc: 33

Ten: Project 1 - Quan ly he thong tap tin.docx
Trang thái: Tập tin
Chiếncac cluster: 7, 8, 9, 10, 11, 12
Tuong ung cac sector: 8232, 8233, 8234, 8235, 8236, 8237, 8238, 8239, 8240, 8241, 8242, 8243, 8244, 8245, 8246, 8247, 8248, 8249, 8250, 8251, 8252, 8253, 8254, 8255, 8256, 8257, 8258, 8259, 8260, 8261, 8262, 8263, 8264, 8265, 8266, 8267, 8268, 8269, 8270, 8271, 8272, 8273, 8274, 8275, 8276, 8277, 8278, 8279
Kich thuoc: 23609

Ten: abcdefghijklmnopqrstuvwxyz.txt
Trang thái: Tập tin
Chiếncac cluster: 13
Tuong ung cac sector: 8280, 8281, 8282, 8283, 8284, 8285, 8286, 8287
Kich thuoc: 91

```

Cây thư mục

```

-----CÂY THU MỤC-----
--DEMO
  --FAT32  TXT
    --Project 1 - Quan ly he thong tap tin.docx
--abcdefghijklmnopqrstuvwxyz.txt

```

Truy xuất thông tin tập tin

```

-----THÔNG TIN TREN CÂY THU MỤC-----
--[DEMO] thư mục
  --FAT32  TXT
    --Project 1 - Quan ly he thong tap tin.docx

--FAT32 [TXT]
do an quan ly tap tin
demo FAT32

--Project 1 - Quan ly he thong tap tin.docx
Doc noi dung file bang: Word

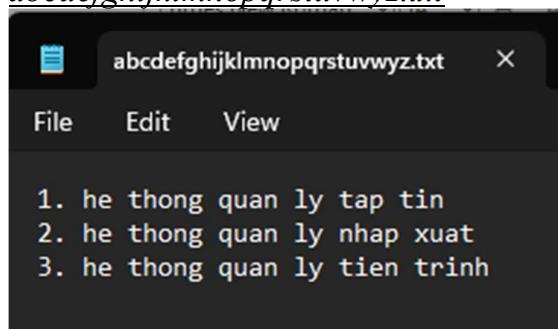
--abcdefghijklmnopqrstuvwxyz.txt
1. he thong quan ly tap tin
2. he thong quan ly nhap xuat
3. he thong quan ly tien trinh

```

Thông tin trong file fat32.txt



Thông tin trong file abcdefghijklmnopqrstuvwxyz.txt



9.2. NTFS

Sau khi tạo ổ đĩa ảo theo hướng dẫn như trên, chạy project NTFS.

Màn hình console như sau sẽ xuất hiện, nhập ổ đĩa muốn đọc. Nếu không tìm được ổ đĩa hoặc không phải là ổ đĩa NTFS thì chương trình sẽ thông báo không tìm được và yêu cầu nhập lại. Trong trường hợp này, ổ đĩa vừa tạo là ổ đĩa G. Nhập tên ổ đĩa → Enter.



The screenshot shows a terminal window with the following text:

```
C:\Users\Admin\Desktop\PRC X + ▾  
Enter volume name (0 to exit): G
```

The window title is "C:\Users\Admin\Desktop\PRC". The command "Enter volume name (0 to exit):" is displayed, followed by the letter "G".

Nếu ổ đĩa NTFS tồn tại, Menu như sau sẽ hiện ra:

```
C:\Users\Admin\Desktop\PRC + ▾  
=====| NTFS VOLUME READER |  
=====| Enter:  
| 1. Change Volume.  
| 2. Print Boot Sector Volume.  
| 3. Print Directory Tree.  
| 4. Print Detailed Directory Tree.  
| 5. Search for File or Folder.  
| 0. Exit.  
=====| Your choice:  
=====
```

Nhập các số và nhấn Enter để thực hiện các yêu cầu:

0. Thoát
1. Đổi ô đĩa khác
2. In ra thông tin về NTFS Boot Sector

```
C:\Users\Admin\Desktop\PRC  X + | v  
=====| NTFS VOLUME READER |=====  
| Enter:  
| 1. Change Volume.  
| 2. Print Boot Sector Volume.  
| 3. Print Directory Tree.  
| 4. Print Detailed Directory Tree.  
| 5. Search for File or Folder.  
| 0. Exit.  
=====  
Your choice: 2  
=====  
PARTITION BOOT SECTOR NTFS  
=====  
OEM Name: NTFS  
Bytes Per Sector: 512  
Sectors Per Cluster: 8  
Sectors Per Track: 63  
Number of Heads: 255  
Total sectors in volume: 2091007(1021.MB)  
Starting Cluster of MTF: 87125  
Starting Cluster of MFTMirror : 2  
MTF Entry Size: 1024 byte  
Number of Bytes in Index Block:4096  
=====  
Press any key to continue . . . |
```

3. In cây thư mục

```
=====| NTFS VOLUME READER |=====
| Enter:
| 1. Change Volume.
| 2. Print Boot Sector Volume.
| 3. Print Directory Tree.
| 4. Print Detailed Directory Tree.
| 5. Search for File or Folder.
| 0. Exit.
=====
```

```
Your choice: 3
```

```
DIRECTORY TREE
```

```
L:
```

```
  L0S:
```

```
    LEXCERCISES:
      └── Baitap.docx(14.91KB)
      └── FAT_EX.pdf(497.2KB)
      └── Hedieuhanh.pdf(237.4KB)
```

```
    LLECTURE:
```

```
      └── bai03.pdf(2.260MB)
          └── bai10.pdf(1.310MB)
```

```
    LTHEORY:
```

```
      └── Final.docx(54.95KB)
          └── Midterm.docx(734.7KB)
              └── Test.txt(1.940KB)
```

4. In cây thư mục chi tiết

```
=====
DETAILED DIRECTORY TREE

Attribute:
First Index: 5
Sector MFT entry: 697010

os
Attribute: Folder
First Index: 41
Sector MFT entry: 697082

EXERCISES
Attribute: Folder
First Index: 42
Sector MFT entry: 697084

Name: Baitap.docx
Attribute: File
Create: -Date: 2024/3/6 Time: 22:23:29
Access: -Date: 2024/3/10 Time: 17:34:33
Modified: -Date: 2024/1/20 Time: 10:6:42
File Size: 15277 Byte
Entry Index: 49
Sector: 697098
Name: FAT_EX.pdf
Attribute: File
Create: -Date: 2024/3/6 Time: 22:23:29
Access: -Date: 2024/3/6 Time: 22:23:29
Modified: -Date: 2024/1/22 Time: 17:32:47
File Size: 509173 Byte
Entry Index: 43
Sector: 697086
Name: Hedieuhanh.pdf
Attribute: File
Create: -Date: 2024/3/6 Time: 22:23:29
Access: -Date: 2024/3/6 Time: 22:23:29
Modified: -Date: 2024/1/20 Time: 10:6:53
```

```
C:\Users\Admin\Desktop\PRC > + >

    Modified: -Date: 2024/1/22           Time: 17:32:47
    File Size: 509173 Byte
    Entry Index: 43
    Sector: 697086
    Name: Hedieuhanh.pdf
    Attribute: File
    Create: -Date: 2024/3/6 Time: 22:23:29
    Access: -Date: 2024/3/6 Time: 22:23:29
    Modified: -Date: 2024/1/20           Time: 10:6:53
    File Size: 243179 Byte
    Entry Index: 48
    Sector: 697096
    -----
    | LECTURE
    | Attribute: Folder
    | First Index: 52
    | Sector MFT entry: 697104
    |
    |     Name: bai03.pdf
    |     Attribute: File
    |     Create: -Date: 2024/3/6 Time: 22:23:30
    |     Access: -Date: 2024/3/6 Time: 22:23:30
    |     Modified: -Date: 2024/1/21           Time: 11:3:32
    |     File Size: 2365136 Byte
    |     Entry Index: 53
    |     Sector: 697106
    |     Name: bai10.pdf
    |     Attribute: File
    |     Create: -Date: 2024/3/6 Time: 22:23:30
    |     Access: -Date: 2024/3/6 Time: 22:23:30
    |     Modified: -Date: 2024/1/21           Time: 9:38:11
    |     File Size: 1372732 Byte
    |     Entry Index: 66
    |     Sector: 697132
    |
    | THEORY
    | Attribute: Folder
    | First Index: 75
    | Sector MFT entry: 697150
    |
    |     Name: Final.docx
    |     Attribute: File
    |     Create: -Date: 2024/3/6 Time: 22:23:30
```

```
C:\Users\Admin\Desktop\PRC + ▾

File Size: 2365136 Byte
Entry Index: 53
Sector: 697106
Name: bai10.pdf
Attribute: File
Create: -Date: 2024/3/6 Time: 22:23:30
Access: -Date: 2024/3/6 Time: 22:23:30
Modified: -Date: 2024/1/21 Time: 9:38:11
File Size: 1372732 Byte
Entry Index: 66
Sector: 697132

THEORY
Attribute: Folder
First Index: 75
Sector MFT entry: 697150

Name: Final.docx
Attribute: File
Create: -Date: 2024/3/6 Time: 22:23:30
Access: -Date: 2024/3/6 Time: 22:23:30
Modified: -Date: 2024/1/20 Time: 10:7:14
File Size: 56269 Byte
Entry Index: 76
Sector: 697152
Name: Midterm.docx
Attribute: File
Create: -Date: 2024/3/6 Time: 22:23:30
Access: -Date: 2024/3/9 Time: 18:39:10
Modified: -Date: 2024/1/20 Time: 10:7:19
File Size: 752338 Byte
Entry Index: 77
Sector: 697154
Name: Test.txt
Attribute: File
Create: -Date: 2024/3/9 Time: 18:22:16
Access: -Date: 2024/3/10 Time: 11:55:16
Modified: -Date: 2024/3/10 Time: 11:55:16
File Size: 1987 Byte
Entry Index: 47
Sector: 697094

Press any key to continue . . . |
```

5. Tìm tên file hoặc folder để xem thông tin chi tiết

- Đọc nội dung với file .txt

```
C:\Users\Admin\Desktop\PRC + v
=====
|      NTFS VOLUME READER      |
=====
| Enter:
| 1. Change Volume.
| 2. Print Boot Sector Volume.
| 3. Print Directory Tree.
| 4. Print Detailed Directory Tree.
| 5. Search for File or Folder.
| 6. Exit.
=====
Your choice: 5
=====
Input entry name:Test.txt
-----FILE INFORMATION-----
Name: Test.txt
Attribute: File
Create: -Date: 2024/3/9 Time: 18:22:16
Access: -Date: 2024/3/10     Time: 11:55:16
Modified: -Date: 2024/3/10    Time: 11:55:16
File Size: 1987 Byte
Entry Index: 47
Sector: 697094
-----DATA-----
The natural course of evolution has seen numerous species come and go, including the iconic dinosaurs. Some argue that it is a futile endeavor for humans to interfere with this natural process and prevent the extinction of animal species. Despite acknowledging the logic behind this suggestion, I firmly disagree with it for several reasons.

Proponents of non-intervention argue that extinction is an inherent part of the evolutionary cycle. They contend that species, including dinosaurs, faced natural selection pressures that led to their demise, making it a fundamental aspect of the Earth's ecological processes. Attempting to prevent such extinctions, according to this view, interferes with the natural order of the planet, meaning that it can cause more disadvantages rather than offering any practical benefit. They further argue that all efforts spent on conserving endangered species may not guarantee desired outcomes while demanding exorbitant expenses that can be allocated to more pressing issues.

Nevertheless, I strongly believe that we should take immediate action to address the issue for reasons related to human impacts and the values that endangered species can offer. Regarding the former, the impact of human activities, such as deforestation, pollution, and climate change, has accelerated the rate of species extinction, affecting biodiversity and potentially causing a domino effect on other interconnected species. Humans, as conscious stewards of the planet, have a moral obligation to mitigate the impacts of their actions and preserve the diversity of life. Another significant aspect is that endangered species often possess unique genetic traits which should be studied thoroughly in order to pave the way for breakthroughs in medicine, biotechnology, and other scientific domains. For instance, a species might have developed resistance to certain diseases, which could inform medical research and contribute to the development of new treatments. This has been a recurring phenomenon throughout Earth's history, I would take the view that we should intervene with it to save threatened species. Preventing the extinction of animal species is not only an ethical imperative but also crucial for groundbreaking discoveries that can greatly benefit humanity. Press any key to continue . . .
```

Dùng các phần mềm khác để đọc các file khác:

```
C:\Users\Admin\Desktop\PRC X + | v  
=====| NTFS VOLUME READER |=====  
| Enter:  
| 1. Change Volume.  
| 2. Print Boot Sector Volume.  
| 3. Print Directory Tree.  
| 4. Print Detailed Directory Tree.  
| 5. Search for File or Folder.  
| 0. Exit.  
=====  
Your choice: 5  
=====  
Input entry name:Final.docx  
-----FILE INFORMATION-----  
Name: Final.docx  
Attribute: File  
Create: -Date: 2024/3/6 Time: 22:23:30  
Access: -Date: 2024/3/6 Time: 22:23:30  
Modified: -Date: 2024/1/20 Time: 10:7:14  
File Size: 56269 Byte  
Entry Index: 76  
Sector: 697152  
Use compatible software to read the content!  
Press any key to continue . . . |
```

Nếu là thư mục:

```
C:\Users\Admin\Desktop\PRC × + | -  
| 3. Print Directory Tree.  
| 4. Print Detailed Directory Tree.  
| 5. Search for File or Folder.  
| 0. Exit.  
=====  
Your choice: 5  
=====  
Input entry name:lecture  
-----FOLDER INFORMATION-----  
DIRECTORY TREE  
|  
└ LECTURE:  
    ├ bai03.pdf(2.260MB)  
    └ bai10.pdf(1.310MB)  
-----  
DETAILED DIRECTORY TREE  
|  
└ LECTURE  
    Attribute: Folder  
    First Index: 52  
    Sector MFT entry: 697104  
    -----  
    Name: bai03.pdf  
    Attribute: File  
    Create: -Date: 2024/3/6 Time: 22:23:30  
    Access: -Date: 2024/3/6 Time: 22:23:30  
    Modified: -Date: 2024/1/21 Time: 11:3:32  
    File Size: 2365136 Byte  
    Entry Index: 53  
    Sector: 697106  
    Name: bai10.pdf  
    Attribute: File  
    Create: -Date: 2024/3/6 Time: 22:23:30  
    Access: -Date: 2024/3/6 Time: 22:23:30  
    Modified: -Date: 2024/1/21 Time: 9:38:11  
    File Size: 1372732 Byte  
    Entry Index: 66  
    Sector: 697132  
Press any key to continue . . . |
```

10. Tính ứng dụng của đồ án

- Có rất nhiều ứng dụng của việc lấy các thuộc tính của tập tin trong bảng thư mục gốc và các thông số của Boot Sector trong việc quản lý và khai thác dữ liệu trên các phân vùng FAT32 và NTFS. Dưới đây là một số ý tưởng về việc áp dụng các thông tin này:

- **Khôi phục dữ liệu:** Trong trường hợp một phân vùng bị hỏng hoặc xảy ra lỗi, việc lấy thông tin từ Boot Sector và bảng thư mục gốc sẽ giúp phục hồi dữ liệu. Dữ liệu có thể bị mất hoặc bị lỗi trong trường hợp phân vùng bị hỏng, vì vậy việc lấy thông tin này có thể giúp định vị và khôi phục lại các tập tin bị mất.
- **Kiểm tra tính toàn vẹn của hệ thống tập tin:** Thông tin về byte per sector, sector per cluster, số lượng sector và cluster của Boot Sector sẽ giúp xác định tính toàn vẹn của hệ thống tập tin. Nếu các thông số này không đúng, có thể làm ảnh hưởng đến khả năng truy cập và lưu trữ dữ liệu trên phân vùng. Việc kiểm tra tính toàn vẹn của hệ thống tập tin giúp đảm bảo an toàn cho dữ liệu.
- **Tối ưu hóa lưu trữ:** Các thông số byte per sector, sector per cluster, số lượng sector và cluster của Boot Sector cũng cung cấp thông tin về cách hệ thống tập tin được tổ chức và lưu trữ dữ liệu. Dựa trên thông tin này, có thể tối ưu hóa việc lưu trữ dữ liệu và cải thiện hiệu suất truy cập dữ liệu trên phân vùng.
- **Phân tích dữ liệu:** Việc lấy các thuộc tính của tập tin trong bảng thư mục gốc cũng cung cấp nhiều thông tin quan trọng về dữ liệu, chẳng hạn như kích thước tập tin, định dạng và thông tin tác giả. Những thông tin này có thể được sử dụng để phân tích dữ liệu và đưa ra các quyết định về quản lý và khai thác dữ liệu.

11. Các nguồn tham khảo

- http://ntfs.com/ntfs_basics.htm
- Slide bài giảng môn học Hệ điều hành – Giảng viên: Lê Việt Long.
- <https://daohocthuat.com/cach-tao-o-dia-ao-tren-windows-10-cuc-nhanh.html>