

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN  
KHOA CÔNG NGHỆ THÔNG TIN  
BỘ MÔN MẠNG MÁY TÍNH - VIỄN THÔNG**



**BÀI BÁO CÁO**

**Project 1 – Hệ thống quản lý tập tin trên Windows**

**Môn học: Hệ điều hành**

**Lớp: 22CLC06**

**Giáo viên hướng dẫn: Ths. Lê Viết Long**

**Sinh viên thực hiện:**

- + 22127022 - Võ Hoàng Anh
- + 22127154 - Nguyễn Gia Huy
- + 22127210 - Phạm Anh Khôi
- + 22127413 - Phạm Hoàng Tiên

# Mục lục

<b>I. Thông tin nhóm.....</b>	<b>4</b>
<b>II. Bảng phân công công việc.....</b>	<b>4</b>
<b>III. Đánh giá mức độ hoàn thành.....</b>	<b>4</b>
<b>IV. Các bước thực hiện.....</b>	<b>5</b>
A. Phân vùng FAT32.....	5
1. Đọc thông tin phân vùng FAT32.....	5
1.1 Đọc thông tin Boot Sector.....	5
1.2 Đọc thông tin FAT.....	7
1.3 Đọc thông tin RDET.....	7
2. Hiển thị cây thư mục gốc.....	9
3. Truy xuất cây thư mục.....	9
B. Phân vùng NTFS.....	10
1. Đọc thông tin phân vùng NTFS.....	10
1.1 Đọc thông tin BPB.....	11
1.2 Đọc thông tin MFT.....	12
1.2.1 Cấu trúc của MFT Entry.....	12
1.2.2 Attribute.....	13
1.2.3 Một số loại Attribute.....	18
1.2.4 Ví dụ minh họa về MFT Entry.....	27
2. Hiển thị thư mục gốc.....	32
3. Truy xuất cây thư mục.....	34
<b>V. Hình ảnh demo chương trình.....</b>	<b>34</b>
A. FAT32.....	34
1. Đọc thông tin FAT32 (Boot Sector, FAT, RDET).....	34
2. Hiển thị cây thư mục gốc.....	35
3. Truy xuất cây thư mục.....	35
B. NTFS.....	37
1. Đọc thông tin NTFS (BPB, MFT).....	37
2. Hiển thị cây thư mục gốc.....	38
3. Truy xuất cây thư mục.....	38
<b>VI. Nguồn tham khảo.....</b>	<b>41</b>

## I. Thông tin nhóm

- Môn: Hệ điều hành
- Lớp: 22CLC06
- Project: 1 - Quản lý hệ thống tập tin
- Thành viên nhóm
- + 22127022 - Võ Hoàng Anh
- + 22127154 - Nguyễn Gia Huy
- + 22127210 - Phạm Anh Khôi
- + 22127413 - Phạm Hoàng Tiên

## II. Bảng phân công công việc

MSSV	Họ và tên	Công việc
22127022	Võ Hoàng Anh	FAT32
22127154	Nguyễn Gia Huy	NTFS, Viết báo cáo
22127210	Phạm Anh Khôi	FAT32
22127413	Phạm Hoàng Tiên	FAT32

## III. Đánh giá mức độ hoàn thành

STT	Phân vùng	Yêu cầu	Mức độ hoàn thành
1	FAT32	Đọc thông tin Boot Sector	Hoàn thành
2	FAT32	Hiển thị cây thư mục gốc	Hoàn thành
3	FAT32	Truy xuất cây thư mục	Hoàn thành
4	NTFS	Đọc thông tin Partition Boot Sector	Hoàn thành
5	NTFS	Hiển thị cây thư mục gốc	Hoàn thành
6	NTFS	Truy xuất cây thư mục	Hoàn thành

## IV. Các bước thực hiện

### A. Phân vùng FAT32

#### 1. Đọc thông tin phân vùng FAT32

Phân vùng FAT32 được tổ chức thành 2 vùng:

Vùng hệ thống

- + Vùng Boot Sector
- + Bảng FAT

Vùng dữ liệu

- + Bảng thư mục gốc (RDET)
- + Các Cluster dữ liệu

##### 1.1 Đọc thông tin Boot Sector

Vùng **Boot Sector** gồm một số sector đầu tiên của phân vùng (Partition), trong đó quan trọng nhất là **Sector đầu tiên** :

- + Chứa các thông số quan trọng của phân vùng
- + Chứa một đoạn chương trình nhỏ để nạp HĐH khi khởi động máy

Hình dưới đây là cấu trúc chi tiết của Boot Sector trong FAT32

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	<b>S<sub>C</sub>: số sector trên cluster (như FAT)</b>
E	2	<b>S<sub>B</sub>: số sector thuộc vùng Bootsector (như FAT)</b>
10	1	<b>N<sub>F</sub>: số bảng FAT, thường là 2 (như FAT)</b>
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	<b>S<sub>V</sub>: Kích thước volume (như FAT)</b>
24	4	<b>S<sub>F</sub>: Kích thước mỗi bảng FAT</b>
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	<b>Cluster bắt đầu của RDET</b>
30	2	<b>Sector chứa thông tin phụ (về cluster trống), thường là 1</b>
32	2	<b>Sector chứa bản lưu của Boot Sector</b>
34	C	Danh riêng (cho các phiên bản sau)
40	1	Kí hiệu vật lý của đĩa chứa vol (0 : mềm, 80h: cứng)
41	1	Danh riêng
42	1	Kí hiệu nhận diện HĐH
43	4	SerialNumber của Volume
47	B	Volume Label
52	8	<b>Loại FAT, là chuỗi “FAT32”</b>
5A	1A4	Đoạn chương trình khởi tạo & nạp HĐH khi khởi động máy
1FE	2	Đầu hiệu kết thúc BootSector /Master Boot (luôn là AA55h)

Trong đó có một số thông tin quan trọng cần chú ý như sau:

Offset	Số byte	Nội dung
B	2	Số byte trên Sector (Thường là 512)
D	1	<b>SC: Số Sector trên mỗi Cluster</b>
E	2	<b>SB: Số Sector thuộc vùng Boot Sector</b>
10	1	N <sub>F</sub> : Số bảng FAT, thường là 2
20	4	S <sub>V</sub> : Kích thước Volume (Tổng số Sector)
24	4	<b>S<sub>F</sub>: Kích thước mỗi bảng FAT (Số lượng Sector mỗi FAT)</b>
2C	4	<b>Cluster bắt đầu của RDET</b>

## 1.2 Đọc thông tin FAT

Vị trí bắt đầu của FAT: Tại Sector thứ  $S_B$

Mỗi phần tử của bảng FAT có kích thước 4 byte, lưu vị trí (cluster) của thư mục/tập tin theo dạng danh sách liên kết

Giá trị	x	x	3	4	EOF	7	EOF	6
Phần tử	0	1	2	3	4	5	6	7

Bảng trên có nghĩa là: Tập tin/thư mục bắt đầu từ cluster 2 sẽ kết thúc ở cluster 4 (2-3-4); tập tin/thư mục bắt đầu từ cluster 5 sẽ kết thúc ở 6 (5-7-6)

Phần tử thứ K trên bảng FAT (**đánh số từ 0**) cho biết trạng thái của Cluster thứ K trên vùng dữ liệu (**đánh số từ 2**). **2 phần tử đầu của bảng FAT không dùng**

## 1.3 Đọc thông tin RDET

Trong FAT32, RDET là gồm 1 dãy các phần tử (gọi là Entry) nằm trên vùng dữ liệu:

- Mỗi phần tử trên vùng dữ liệu, gọi là **Cluster**, có kích thước  $2^n$  **Sector**, tùy thuộc vào người dùng format
- Cluster trên vùng dữ liệu đánh số từ 2. Công thức tương quan giữa Cluster thứ k trên vùng dữ liệu và Sector thứ i trên phân vùng như sau:

$$i = S_B + S_F * N_F + [S_{RDET}] + (k-2) * S_C$$

Để đọc được thông tin của RDET, ta xác định Cluster bắt đầu của RDET (đã biết), dò bảng FAT để tìm các Cluster của RDET -> Đọc các Cluster đó và lưu vào các Entry

Mỗi Entry có kích thước 32 byte, lưu các thông tin của 1 tập tin. Mỗi tập tin/thư mục có thể chiếm 1 hoặc nhiều Entry. Dựa vào byte đầu tiên của mỗi Entry, ta biết được trạng thái của Entry này:

- 0 - Entry trống
- E5h - tập tin chiếm Entry này đã bị xóa
- Giá trị khác - Đang chứa thông tin của tập tin/thư mục

Có 2 loại Entry, dựa vào byte thứ B của mỗi Entry, ta phân biệt như sau:

- Entry chính: có dạng 0.0.A.D.V.S.H.R khi đổi ra bit, lưu thuộc tính trạng thái
- Entry phụ: có giá trị **0Fh**

Một tập tin có thể chiếm 1 Entry chính và 1 hoặc nhiều Entry phụ. Trong RDET, nó sẽ sắp xếp theo thứ tự như sau (Entry phụ ở trước Entry chính):

- + Entry phụ N
- + Entry phụ N - 1
- + ...
- + Entry phụ 2
- + Entry phụ 1
- + Entry chính

Bảng dưới đây minh họa cấu trúc chi tiết của một Entry chính

Offset (hex)	Số byte	Ý nghĩa
0	8	Tên chính /tên ngắn - lưu bằng mã ASCII
8	3	Tên mở rộng – mã ASCII
B	1	Thuộc tính trạng thái (0.0.A.D.V.S.H.R)
C	1	Dành riêng
D	3	Giờ tạo (mili giây:7; giây:6; phút:5; giờ:5)
10	2	Ngày tạo (ngày: 5; tháng: 4; năm-1980: 7)
12	2	Ngày truy cập gần nhất (lưu như trên)
14	2	Cluster bắt đầu – phần Word (2Byte) cao
16	2	Giờ sửa gần nhất (giây/2:5; phút:6; giờ:5)
18	2	Ngày cập nhật gần nhất (lưu như trên)
1A	2	Cluster bắt đầu – phần Word thấp
1C	4	Kích thước của phần nội dung tập tin

Ý nghĩa của các giá trị thuộc **Byte thứ B** như sau:

- + A - Archive: *tập tin*
- + D - Directory: *thư mục*
- + V - VolLabel
- + S - System: *thuộc về hệ thống*
- + H - Hidden: *ẩn*
- + R - ReadOnly: *chỉ đọc*

Ví dụ một **Entry** có **Byte thứ B** mang giá trị:

- + 0x10, ta biết được nó là Entry chính, đổi sang bit có dạng 0001 0000 -> Directory: Entry đó là thông tin của một thư mục
- + 0x23 - 0010 0011 -> Archive, Hidden, ReadOnly: Tập tin ẩn chỉ đọc

Trong trường hợp tên của tập tin/thư mục quá dài thì thông tin về tên sẽ được lưu vào Entry phụ, cụ thể Entry phụ sẽ có cấu trúc như sau:

Offset	Số byte	Ý nghĩa
0	1	Thứ tự của entry (bắt đầu từ 1)
1	A (10d)	5 ký tự UniCode – bảng mã UTF16
B (11d)	1	Dấu hiệu nhận biết (luôn là 0Fh)
E (14d)	C (12d)	6 ký tự kế tiếp
1C (28d)	4	2 ký tự kế tiếp

## 2. Hiển thị cây thư mục gốc

Để “*Hiển thị cây thư mục gốc*” và “*Truy xuất cây thư mục*”, ta cần xây dựng cấu trúc cây thư mục từ những thông tin có sẵn, vì tính chất tương tự nhau về mặt logic, phần xây dựng cấu trúc cây sẽ được trình bày ở phần “2. *Hiển thị cây thư mục gốc*” trong *phân vùng NTFS*

Để hiển thị cây thư mục gốc (Root Directory), ta đọc bảng RDET để lấy thông tin của các Entry, từ đó biết được thông tin của các tập tin, thư mục, lưu chúng lại và hiển thị lên màn hình

## 3. Truy xuất cây thư mục

Để truy xuất cây thư mục, ta:

- Xác định Entry chính của thư mục đó trong bảng thư mục (RDET nếu thư mục đó nằm trong thư mục gốc, SDET nếu thư mục đó nằm trong thư mục con) chứa thông tin của thư mục dựa vào phần tên
- Từ Entry chính tìm được, ta biết được chỉ số Cluster, suy ra phần tử FAT đầu tiên
- Từ phần tử FAT đầu tiên, ta vào bảng FAT, xác định các phần tử còn lại của tập tin/thư mục, tương ứng có được các Cluster của tập tin/thư mục này. Tính được các Sector của tập tin (Tương tự với việc tìm các Cluster của RDET lúc đọc bảng RDET)

- Đọc các Sector nội dung tìm được theo từng Entry (32 bytes) và hiển thị thông tin của các tập tin và thư mục con của thư mục này. (Tương tự với việc đọc các Entry của RDET)

Để đọc nội dung của tập tin, ta:

- Xác định Entry chính trong bảng thư mục (RDET / SDET) chứa thông tin của tập tin dựa vào phần tên và phần mở rộng (lưu ý trường hợp tên dài)
- Từ Entry chính tìm được, ta có chỉ số Cluster đầu tiên ~ Phần tử FAT đầu tiên
- Từ phần tử FAT đầu tiên này, ta vào bảng FAT, xác định các phần tử còn lại của tập tin, tương ứng có được các Cluster của tập tin này. Tính được các Sector của tập tin
- Đọc các Sector nội dung của tập tin

Trong đồ án lần này, ta chỉ quan tâm đến đọc nội dung của những tập tin file “.txt” nên ta cần kiểm tra phần tên mở rộng của Entry chính của tập tin (Byte 8 - 10) xem nó có phải là “.txt” hay không:

- Nếu là “.txt”: Ta đọc nội dung tập tin như cách trên
- Nếu không là “.txt”: Xuất ra thông báo yêu cầu người dùng dùng phần mềm khác để đọc nội dung tập tin này.

## B. Phân vùng NTFS

### 1. Đọc thông tin phân vùng NTFS

Cấu trúc của phân vùng NTFS (New Technology File System) có dạng như sau

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

Trong đó

- VBR (Volume Boot Record): bản ghi khởi động của ổ đĩa logic, nó luôn nằm ở vị trí đầu tiên trong mỗi ổ đĩa logic. VBR chứa:
  - Mã khởi động
  - BPB (Bios Parameter Block): chứa các thông tin quan trọng của phân vùng (73 byte từ 0Bh đến 53h)
  - Thông báo lỗi và một số thông tin khác
- MFT (Master File Table)
- Nội dung của tập tin (loại non-resident): chứa nội dung của những tập tin phân mảnh

## 1.1 Đọc thông tin BPB

Để đọc thông tin của BPB, ta đọc nội dung của sector đầu tiên (thường là 512Byte), sau đó đọc thông tin của 73 byte của BPB, gồm một số thông tin quan trọng như sau:

Địa chỉ (Offset)	Kích thước (Byte)	Mô tả
0Bh	2	Kích thước một SECTOR (Byte)
0Dh	1	Số SECTOR trên một CLUSTER
30h	8	CLUSTER bắt đầu của MFT
38h	8	CLUSTER bắt đầu của MFT dự phòng
40h	1	Kích thước một bảng ghi (MFT Entry) trong MFT (Byte), thường là 1024

**Lưu ý:** Giá trị ở byte 40h lưu số ở dạng bù 2. Giả sử ta đọc được giá trị của byte 40h là 0xF6 = 1111 0110 (dạng bù 2), ta tính ra giá trị hệ thập phân là -10. Kích thước của MFT Entry là  $2^{10}$  = 1024 byte

Như vậy, sau khi đọc thông tin của BPB, ta biết được một số thông tin quan trọng của phân vùng.

Hình bên dưới là ví dụ về BPB và các giá trị trường thuộc tính của nó (Sử dụng phần mềm *Disk Editor*)

Name	Offset	Value	Offset	00 01 02 03 04 05 06 07	08 09 0A 0B 0C 0D 0E 0F	ASCII
JMP instruction	000	EB 52 90	000004000	EB 52 90 4E 54 46 53 20	20 20 20 00 02 08 00 00	ER.NTFS
OEM ID	003	NTFS	000004010	00 00 00 00 F8 00 00	3F 00 FF 00 20 00 00 00	...z.2.y. ....
BIOS Parameter Block	00B	512	000004020	00 00 00 00 80 00 00 00	DF DF 94 03 00 00 00 00	...BB.....
Bytes per sector	00B	512	000004030	00 00 0C 00 00 00 00 00	02 00 00 00 00 00 00 00	....
Sectors per cluster	00D	8	000004040	F6 00 00 00 01 00 00 00	F1 BF 87 34 EB 87 34 22	0...ñ.4e.4"
Reserved sectors	00E	0	000004050	00 00 00 00 FA 33 C0 8E	D0 BC 00 7C FB 68 C0 07	...ù3À.Đ4.  ûHÀ.
(always zero)	010	00 00 00	000004060	1F 1E 68 66 00 CB 88 16	0E 00 66 81 3E 03 00 4E	..hf.È.....f.>..N
(unused)	013	00 00	000004070	54 46 53 75 15 B4 41 BB	AA 55 CD 13 72 0C 81 FB	TFSu.'À»^Úí.r..Ù
Media descriptor	015	248	000004080	55 AA 75 06 F7 C1 01 00	75 03 E9 DD 00 1E 83 EC	Uº.à.À.ú.éý..í
(unused)	016	00 00	000004090	18 68 1A 00 B4 48 8A 16	0E 00 8B F4 16 1F CD 13	..h..H.....ö..í.
Sectors per track	018	63	0000040A0	9F 83 C4 18 9E 58 1F 72	E1 3B 06 0B 00 75 DB A3	.À..X.rá;...uÙé
Number of heads	01A	255	0000040B0	0F 00 C1 2E 0F 00 04 1E	5A 33 DB 99 00 20 2B C8	..A.....zÙ¹. +È
Hidden sectors	01C	32	0000040C0	66 FF 06 11 00 03 16 0F	00 8E C2 FF 06 16 00 E8	fý.....Ây.....è
(unused)	020	00 00 00 00	0000040D0	4B 00 2B C8 77 EF B8 00	BB CD 1A 66 23 C0 75 2D	K.+ÈwÍ.,»Í.f#Àu-
Signature	024	80 00 00 00	0000040E0	66 81 FB 54 43 50 41 75	24 81 F9 02 01 72 1E 16	f.úTCFAu\$.ù..r..
Total sectors	028	60,088,287	0000040F0	68 07 BB 16 68 52 11 16	68 09 00 66 53 66 53 66	h..»h..R..fsfsf
SMFT cluster number	030	786,432	000004100	55 16 16 68 B8 01 66	61 0E 07 CD 1A 33 C0 BF	U..h..fa..í.3À;
SMFTMirr cluster number	038	2	000004110	0A 13 B9 F6 0C FC F3 AA	E9 FE 01 90 90 66 60 1E	..í.ò.ú.éý.é.ò.è..
Clusters per File Record Se...	040	246	000004120	06 66 A1 11 00 66 03 06	1C 00 1E 66 68 00 00 00	.fj..f.....fh...
Clusters per Index Block	044	1	000004130	00 66 50 06 53 68 01 00	68 10 00 B4 42 8A 16 0E	.fP.Sh..h..'B...
Volume serial number	048	F1 BF 87 34 ...	000004140	00 16 1F 8B F4 CD 13 66	59 5B 5A 66 59 66 59 1F	....óí.fý[ZfYfY.
Checksum	050	0	000004150	0F 82 16 00 66 FF 06 11	00 03 16 0F 00 8E C2 FF	....fy.....Ây
Bootstrap code	054	FA 33 C0 8E...	000004160	0E 16 00 75 BC 07 1F 66	61 C3 A1 F6 01 E8 09 00	...u4..faã.ø.è..
Signature (55 AA)	1FE	55 AA	000004170	A1 FA 01 E8 03 00 F4 EB	FD 8B F0 AC 3C 00 74 09	ú.è..ðéý.ð-<.t.
			000004180	B4 0E BB 07 00 CD 10 EB	F2 C3 0D 0A 41 20 64 69	.»..í.éð.À di
			000004190	73 6B 20 72 65 61 64 20	65 72 72 6F 72 20 6F 63	sk read error oc
			0000041A0	63 75 72 72 65 64 00 0D	0A 42 4F 4F 54 4D 47 52	curred...BOOTMGR
			0000041B0	20 69 73 20 63 6F 6D 70	72 65 73 73 65 64 00 0D	is compressed..
			0000041C0	0A 50 72 65 73 73 20 43	74 72 6C 2B 41 6C 74 2B	.Press Ctrl+Alt+
			0000041D0	44 65 6C 20 74 6F 20 72	65 73 74 61 72 74 0D 0A	Del to restart..
			0000041E0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	.....
			0000041F0	00 00 00 00 00 00 8A 01	A7 01 BF 01 00 00 55 AA	.....\$..?..Ua

## 1.2 Đọc thông tin MFT

Dựa vào thông tin đã đọc ở BPB, ta tìm được “CLUSTER bắt đầu của MFT”. Từ đó ta tìm SECTOR bắt đầu của MFT bằng công thức:

$$\text{Chỉ số Sector} = \text{Chỉ số Cluster} * \text{Số Sector trong mỗi Cluster}$$

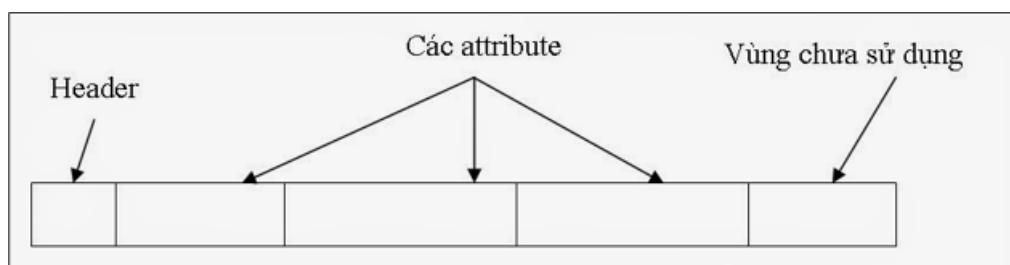
**Lưu ý:** Số Sector trong mỗi Cluster được lưu trong BPB

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

MFT bản chất là một tập tin, do vậy cũng có một MFT Entry mô tả chính nó, đó chính là MFT Entry đầu tiên trong MFT, có tên \$MFT.\$MFT mô tả về kích thước và tổ chức của MFT

### 1.2.1 Cấu trúc của MFT Entry

MFT Entry gồm 2 phần: Header và các Attribute



**Header** gồm 42 byte đầu tiên để chứa thông tin mô tả cho MFT Entry, có cấu trúc cụ thể như sau (những thông tin quan trọng được in đậm)

Offset	Số byte	Mô tả
0x0 – 0x03	4	Dấu hiệu nhận biết MFT entry.
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.
<b>0x14 – 0x15</b>	<b>2</b>	<b>Địa chỉ (offset) bắt đầu của các attribute.</b>
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.
<b>0x20 – 0x27</b>	<b>8</b>	<b>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).</b>
0x28 – 0x29	2	Next attribute ID: mã định danh của attribute kế tiếp sẽ được thêm vào MFT entry.
<b>0x2C - 0x2F</b>	<b>4</b>	<b>ID của Entry</b>

### 1.2.2 Attribute

#### a. Tổng quan về Attribute

Attribute là một cấu trúc dữ liệu, được sử dụng để chứa nội dung của tập tin, chứa các thông tin liên quan đến tập tin, thư mục,...v.v trong hệ thống NTFS.

Có nhiều loại attribute, mỗi loại có cấu trúc tổ chức riêng, có một mã loại (type ID) riêng. Mã loại là một số nguyên. Microsoft sắp xếp thứ tự các attribute trong mỗi MFT entry theo chiều tăng dần của mã loại, nghĩa là, attribute nào có mã loại nhỏ sẽ đứng trước, attribute nào có mã loại lớn sẽ đứng sau.

Các attribute quan trọng thường sử dụng mã loại mặc định, tuy nhiên mã này có thể được định nghĩa lại trong siêu tập tin \$AttrDef.

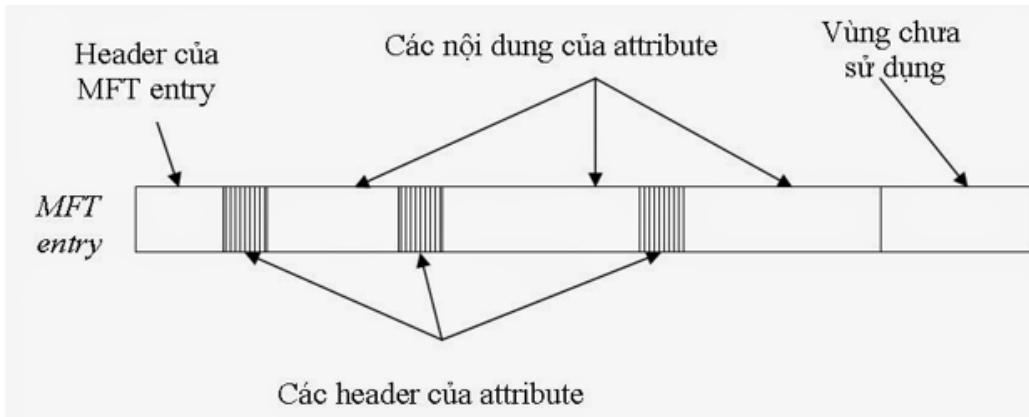
Mỗi loại attribute cũng có một tên gọi riêng, tên gọi được viết hoa toàn bộ, bắt đầu bằng kí hiệu \$. Bảng sau liệt kê một số loại attribute:

Mã loại (hệ 10)	Loại attribute	Mô tả
16	\$STANDARD_INFORMATION	<i>Chứa thông tin chung, ví dụ: các cờ, thời gian tạo, thời gian truy cập mới nhất, thời gian ghi mới nhất, người sở hữu, định danh bảo mật (security ID).</i>
32	\$ATTRIBUTE_LIST	Cho biết vị trí các attribute của một tập tin.
48	\$FILE_NAME	<i>Chứa tên tập tin (dạng Unicode), thời gian tạo, thời điểm ghi tập tin mới nhất, thời điểm truy cập mới nhất.</i>
64	\$VOLUME_VERSION	Chứa thông tin về ổ đĩa. Chỉ có ở phiên bản 1.2
64	\$OBJECT_ID	Chứa định danh duy nhất của tập tin hoặc thư mục. Chỉ có ở phiên bản 3.0 trở về sau.
80	\$SECURITY_DESCRIPTOR	Chứa thông tin về bảo mật và thông tin kiểm soát truy cập của tập tin.
96	\$VOLUME_NAME	Chứa tên ổ đĩa logic.

112	\$VOLUME_INFORMATION	Chứa thông tin về phiên bản của hệ thống quản lý tập tin và các cờ hiệu.
128	<b>\$DATA</b>	<b>Chứa nội dung của tập tin.</b>
144	\$INDEX_ROOT	Chứa nút gốc (root node) của cây chỉ mục (index tree).
160	\$INDEX_ALLOCATION	Chứa các nút của cây chỉ mục (index tree) có gốc thuộc attribute \$INDEX_ROOT.
176	\$BITMAP	Chứa bitmap cho siêu tập tin \$MFT và cho các chỉ mục.
192	\$SYMBOLIC_LINK	Chứa thông tin liên kết mềm. Chỉ có ở phiên bản 1.2
192	\$REPARSE_POINT	Chứa thông tin liên kết mềm. Có ở các phiên bản 3.0 về sau.
208	\$EA_INFORMATION	Chứa thông tin đảm bảo việc tương thích với các ứng dụng trên nền OS/2.
224	\$EA	Chứa thông tin đảm bảo việc tương thích với các ứng dụng trên nền OS/2.
256	\$LOGGED.Utility_STREAM	Chứa khóa (key) và thông tin mã hóa attribute (encrypted attribute) trong các phiên bản từ 3.0 về sau.

### a. Cấu trúc của Attribute

Cấu trúc của một Attribute gồm 2 phần: Header và nội dung



### Header của Attribute

Header của attribute là phần đầu của mỗi attribute, có kích thước 16 byte, chứa các thông tin về: mã loại, kích thước, tên của attribute, ... Cấu trúc cụ thể như sau:

Byte	Mô tả	Ví dụ	
		Giá trị (Hệ 16 - Hệ 10)	Ý nghĩa
0 – 3	Mã loại của attribute (type ID)	0x00000010 – 16	Mã loại là 16: \$STANDARD_INFORMATION
4 – 7	Kích thước của attribute	0x00000060 – 96	Kích thước của attribute là 96 byte
8 – 8	Cờ báo non-resident	0x00 – 0	Attribute thuộc kiểu resident
9 – 9	Chiều dài của tên attribute	0x00 – 0	Attribute này không được đặt tên, nên không có giá trị chiều dài của tên.
10 – 11	Vị trí (offset) chứa tên của attribute	0x0000 – 0	Attribute này không được đặt tên, nên không có thông tin về vị trí của tên.
12 – 13	Các cờ báo	0x0000 – 0	Giá trị cờ báo
14 – 15	Định danh của attribute (định danh này là duy nhất trong phạm vi một MFT entry)	0x0000 – 0	Định danh của attribute (attribute ID) là 0.

### Nội dung của Attribute

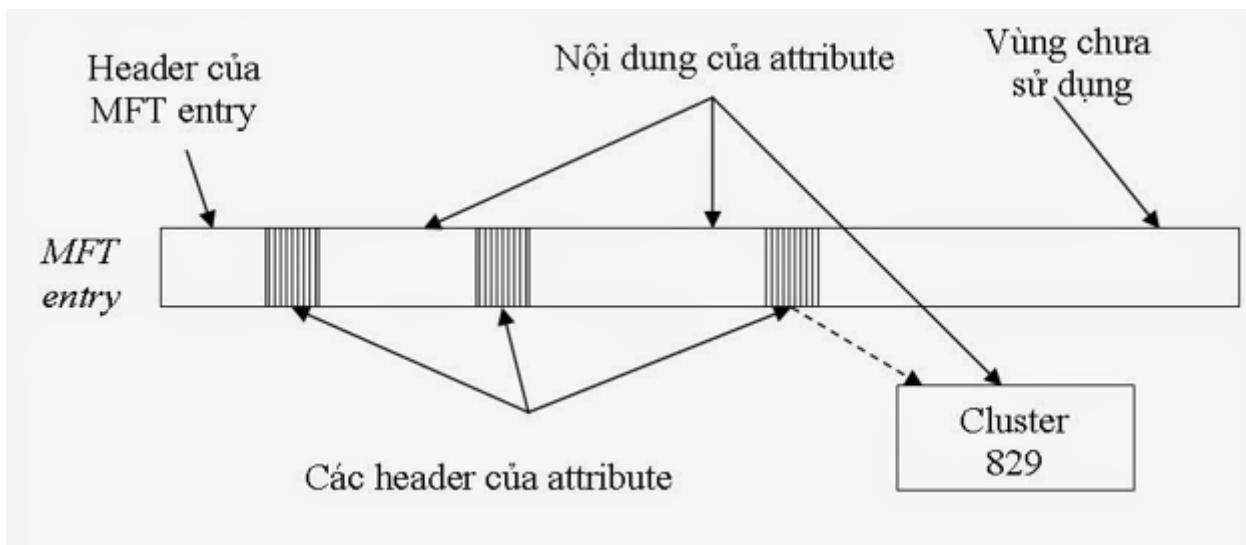
Phần nội dung của attribute được sử dụng để chứa dữ liệu ở định dạng bất kì, với kích thước bất kì. Ví dụ, attribute chứa nội dung của một tập tin có thể có kích thước từ vài MB tới hàng GB. Tuy nhiên, kích thước của một MFT entry chỉ là 1024 byte, nên việc chứa toàn bộ nội dung của attribute trong MFT 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: Lưu trực tiếp bên trong Entry và lưu ở ngoài 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. Xem hình minh họa sau đây, Hình bên dưới cho biết: attribute thứ nhất, thứ hai thuộc loại resident, attribute thứ ba thuộc loại non-resident.



Cấu trúc của Attribute kiểu Resident:

Byte thứ	Mô tả
0 - 15	Cấu trúc header chuẩn (có trong tất cả các loại attribute – Hình 25).
16 – 19	Cho biết kích thước phần nội dung của attribute.
20 - 21	Cho biết nơi bắt đầu (offset) của phần nội dung.

Tùy thuộc vào mỗi loại attribute, phần nội dung của attribute sẽ có cấu trúc tổ chức khác nhau.

Để hiểu rõ hơn về cấu trúc của attribute, phần tiếp theo sẽ trình bày chi tiết về các attribute: \$STANDARD\_INFORMATION, \$FILE\_NAME, \$DATA.

### 1.2.3 Một số loại Attribute

#### a. *Attribute \$STANDARD\_INFORMATION*

Trong một MFT entry, các attribute được sắp xếp theo thứ tự tăng dần của mã loại (type ID). Attribute này có mã loại nhỏ nhất nên luôn luôn nằm ở vị trí đầu tiên.

Cấu trúc của attribute \$STANDARD\_INFORMATION được thể hiện ở bảng sau.

Byte thứ	Mô tả
0 – 15	Header của attribute \$STANDARD_INFORMATION.
16 – 19	Kích thước phần nội dung của attribute \$STANDARD_INFORMATION. Ví dụ: 0x00000048 = 72 (byte).
20 – 21	Nơi bắt đầu (offset) của phần nội dung attribute \$STANDARD_INFORMATION. Ví dụ: 0x0018 = 24 (byte thứ 24 tính từ đầu attribute).

Bảng sau là ví dụ về nội dung header của attribute \$STANDARD\_INFORMATION.

Byte thứ	Giá trị (Hệ 16 – Hệ 10)	Mô tả
0 – 3	0x00000010 – 16	Mã loại là 16.
4 – 7	0x00000060 – 96	Kích thước của attribute \$STANDARD_INFORMATION là 96 byte.
8 – 8	0x00 – 0	Attribute thuộc kiểu resident.

9 – 9	0x00 – 0	Attribute này không được đặt tên, nên không có giá trị chiều dài của tên.
10 – 11	0x0000 – 0	Attribute này không được đặt tên, nên không có thông tin về vị trí của tên.
12 – 13	0x0000 – 0	Giá trị cờ báo.
14 – 15	0x0000 – 0	Định danh của attribute (attribute ID) \$STANDARD_INFORMATION là 0.

Phần nội dung của attribute \$STANDARD\_INFORMATION bắt đầu tại byte 24 (tính từ đầu attribute \$STANDARD\_INFORMATION), kích thước của phần nội dung là 72 byte.

Cấu trúc của phần nội dung attribute \$STANDARD\_INFORMATION và ví dụ được thể hiện ở bảng sau.

Byte thứ	Giá trị (Hệ 16 – Hệ 10)	Mô tả
0 – 7	0x01CF 352F 00BB 73E4 -130381390209053668	Thời gian tạo tập tin: Saturday, March 1, 2014 4:17:01AM UTC.
8 – 15	0x01CF 352E EDCA 04D0 – 130381389891241168	Thời gian thay đổi mới nhất nội dung hai attribute \$DATA hoặc \$INDEX: Saturday, March 1, 2014 4:16:29AM UTC.
16 – 23	0x01CF 352F 0488 3354 – 130381390272803668	Thời gian thay đổi mới nhất thông tin mô tả tập tin: Saturday, March 1, 2014 4:17:07AM UTC.
24 – 31	0x01CF 352F 00BB 73E4 – 130381390209053668	Thời gian truy cập nội dung tập tin mới nhất: Saturday, March 1, 2014 4:17:01AM UTC.
32 – 35	0x00000020	<i>Giá trị cờ báo, tập tin được đánh dấu là archive. (xem bảng về các giá trị của cờ ở bên dưới).</i>

36 – 39	0x00000000	Maximum number of versions
40 – 43	0x00000000	Version number
44 – 47	0x00000000	Class ID
48 – 51	0x00000000	Định danh sở hữu - Owner ID (từ phiên bản 3.0 về sau).
52 – 55	0x000002D9	Định danh bảo mật - Security ID (từ phiên bản 3.0 về sau). Lưu ý: đây không phải là SID trong Windows.
56 – 63	0x00000000 00000000	Thông tin về hạn ngạch - Quota charged (từ phiên bản 3.0 về sau)
64 – 71	0x0000 0000 00BA EC10	Giá trị của USN (update sequence number) (từ phiên bản 3.0 về sau)

Trong các thông tin trên, để đơn giản, ta quan tâm đến giá trị của cờ tại offset 32, bảng sau là giá trị và ý nghĩa của cờ:

Giá trị cờ	Ý nghĩa
0x0000	<i>Thư mục (Directory)</i>
0x0001	<i>Chỉ đọc (read only).</i>
0x0002	<i>Ẩn (hidden).</i>
0x0004	<i>Thuộc hệ thống (system).</i>
0x0020	Thông tin phục vụ việc lưu dự phòng. Tập tin được đánh dấu là archive.

0x0040	Thuộc thiết bị (Device).
0x0080	#Normal.
0x0100	Temporary.
0x0200	Tập tin ‘thưa’ - Spares file.
0x0400	Reparse point.
0x0800	Nén (compressed).
0x1000	Offline
0x2000	Nội dung không được tạo chỉ mục để tăng tốc độ tìm kiếm.
0x4000	Mã hóa (encrypted).

### b. Attribute \$FILE\_NAME

Mỗi tập tin hoặc thư mục (từ đây gọi tắt là tập tin) luôn có ít nhất một attribute \$FILE\_NAME trong MFT entry của nó. Một bản sao của attribute \$FILE\_NAME cũng được lưu trong index của thư mục cha (chứa nó), hai phiên bản này không nhất thiết phải giống nhau hoàn toàn về nội dung.

Mã loại của attribute này là 48. Kích thước của attribute này không cố định, tùy thuộc vào chiều dài của tên tập tin. Cụ thể, kích thước của attribute là: 66 + chiều dài của tên tập tin.

Tên của tập tin là một chuỗi kí tự kiểu UTF-16 Unicode, được định dạng theo kiểu DOS 8.3, Win32 hoặc POSIX. Windows thường yêu cầu một tập tin ít nhất phải có định dạng tên kiểu DOS 8.3, do đó trong attribute \$FILE\_NAME sẽ có cả hai loại định dạng là: DOS và dạng tên đầy đủ. Tùy thuộc vào kiểu định dạng tên, sẽ có quy định những kí tự nào gọi là hợp lệ khi đặt tên cho tập tin.

Nói chung, attribute \$FILE\_NAME chứa rất nhiều các thông tin tương tự như trong attribute \$STANDARD\_INFORMATION. Trong đó có hai thông tin quan trọng là tên của tập tin, tên này cũng được sử dụng để tạo chỉ mục trong thư mục và địa chỉ của thư mục cha, địa chỉ này giúp xác định đường dẫn.

Trong MFT entry, thông thường attribute \$FILE\_NAME nằm ở vị trí thứ hai và là attribute kiểu resident. Tuy nhiên, nếu một tập tin cần nhiều hơn một MFT entry thì sẽ có attribute \$ATTRIBUTE\_LIST nằm giữa attribute \$STANDARD\_INFORMATION và attribute \$FILE\_NAME.

Cũng như tất cả các attribute khác, cấu trúc của attribute \$FILE\_NAME gồm các phần sau.

Byte thứ	Mô tả
0 – 15	Cấu trúc header chuẩn của attribute \$FILE_NAME.
16 – 19	Kích thước phần nội dung của attribute \$FILE_NAME.
20 – 21	Nơi bắt đầu (offset) của phần nội dung attribute \$FILE_NAME.

Để đọc được nội dung của attribute \$FILE\_NAME, trước hết cần tính byte bắt đầu của attribute này.

Attribute \$FILE\_NAME nằm ngay sau attribute \$STANDARD\_INFORMATION. Giả sử nếu Attribute \$STANDARD\_INFORMATION bắt đầu tại offset 56, kích thước của \$STANDARD\_INFORMATION là 96 byte, vậy attribute \$FILE\_NAME sẽ bắt đầu tại vị trí (byte):  $56 + 96 = 152$ .

Bảng sau là ví dụ về nội dung Header của Attribute \$FILE\_NAME

Byte thứ	Giá trị (Hệ 16 - Hệ 10)	Mô tả
0 – 3	0x00000030 – 48	Mã loại là 48.
4 – 7	0x00000070 – 112	Kích thước của attribute \$FILE_NAME là 112 byte.
8 – 8	0x00 – 0	Attribute thuộc kiểu resident.
9 – 9	0x00 – 0	Attribute này không được đặt tên, nên không có giá trị chiều dài của tên.

10 – 11	0x0000 – 0	Attribute này không được đặt tên, nên không có thông tin về vị trí của tên.
12 – 13	0x0000 – 0	Giá trị cờ báo.
14 – 15	0x0002 – 2	Định danh của attribute (attribute ID) \$FILE_NAME là 2.

Đọc tiếp byte thứ 16 -> 19 để biết kích thước phần nội dung, 0x00000052 = 82 byte.

Đọc byte thứ 20 -> 21 để biết vị trí bắt đầu của phần nội dung, 0x0018 = 24, vậy phần nội dung sẽ được lưu bắt đầu từ byte thứ 24 tính từ đầu attribute.

Cấu trúc các trường của phần nội dung attribute \$FILE\_NAME và ví dụ được mô tả trong bảng sau.

Byte thứ	Giá trị (Hệ 16 – hệ 10)	Mô tả
0 – 7	<b>“05 00 00 00 00 00 05 00”</b>	<b>Địa chỉ MFT entry của thư mục cha (file reference).</b>
8 – 15	0x01CF352F00BB73E4 – 130381390209053668	Thời gian tạo tập tin: Saturday, March 1, 2014 4:17:01AM UTC.
16 – 23	0x01CF352F00BB73E4 – 130381390209053668	Thời gian tập tin có thay đổi: Saturday, March 1, 2014 4:17:01AM UTC.
24 – 31	0x01CF352F00BB73E4 – 130381390209053668	Thời gian MFT entry có thay đổi: Saturday, March 1, 2014 4:17:01AM UTC.
32 – 39	0x01CF352F00BB73E4 – 130381390209053668	Thời gian truy cập tập tin mới nhất: Saturday, March 1, 2014 4:17:01AM UTC.
40 - 47	0x0000000000000000	Kích thước cấp phát cho tập tin. NTFS không sử dụng đến trường này, giá trị luôn là 0.
48 – 55	0x0000000000000000	Kích thước thật của tập tin. NTFS không sử dụng đến trường này, giá trị luôn là 0. Giá trị thật được lưu trong attribute \$DATA.
56 – 59	0x00000020	Giá trị cờ báo, tập tin được đánh dấu là archive. (Xem thêm về giá trị và ý nghĩa của cờ đã đề cập trong phần attribute \$STANDARD_INFORMATION).

60 – 63	“00 00 00 00”	Giá trị Reparse
64 – 64	0x08 – 8	<i>Chiều dài của tên tập tin: 8 kí tự</i>
65 – 65	0x03 – 3	<i>Giá trị cho biết định dạng tên tập tin (Namespace). Xem thêm bên dưới.</i>
66+ (8)	“54 00 65 00 73 00 74 00”	<i>Tên của tập tin: Test.txt</i>

Bảng sau mô tả một số kiểu định dạng tên tập tin.

Giá trị	Kiểu định dạng tên	Mô tả
0	POSIX	Tên có phân biệt chữ hoa, chữ thường. Cho phép sử dụng tất cả các kí tự Unicode ngoại trừ ‘/’ và NULL.
1	Win32	Tên có phân biệt chữ hoa, chữ thường. Cho phép sử dụng tất cả các kí tự Unicode ngoại trừ ‘/’, ‘\’, ‘:’, ‘>’, ‘<’ và ‘?’.
2	DOS	Tên không phân biệt chữ hoa, chữ thường, tất cả đều được chuyển sang dạng chữ hoa. Số kí tự của phần tên phải ít hơn hoặc bằng tám kí tự, phần mở rộng phải ít hơn hoặc bằng ba kí tự.
3	Win32 & DOS	Định dạng kép Win32 và DOS. Khi định dạng DOS có thể lưu đầy đủ tên tập tin rồi, thì mặc định hiểu là nó cũng được lưu ở định dạng Win32 mà không cần lưu ở hai định dạng riêng biệt.

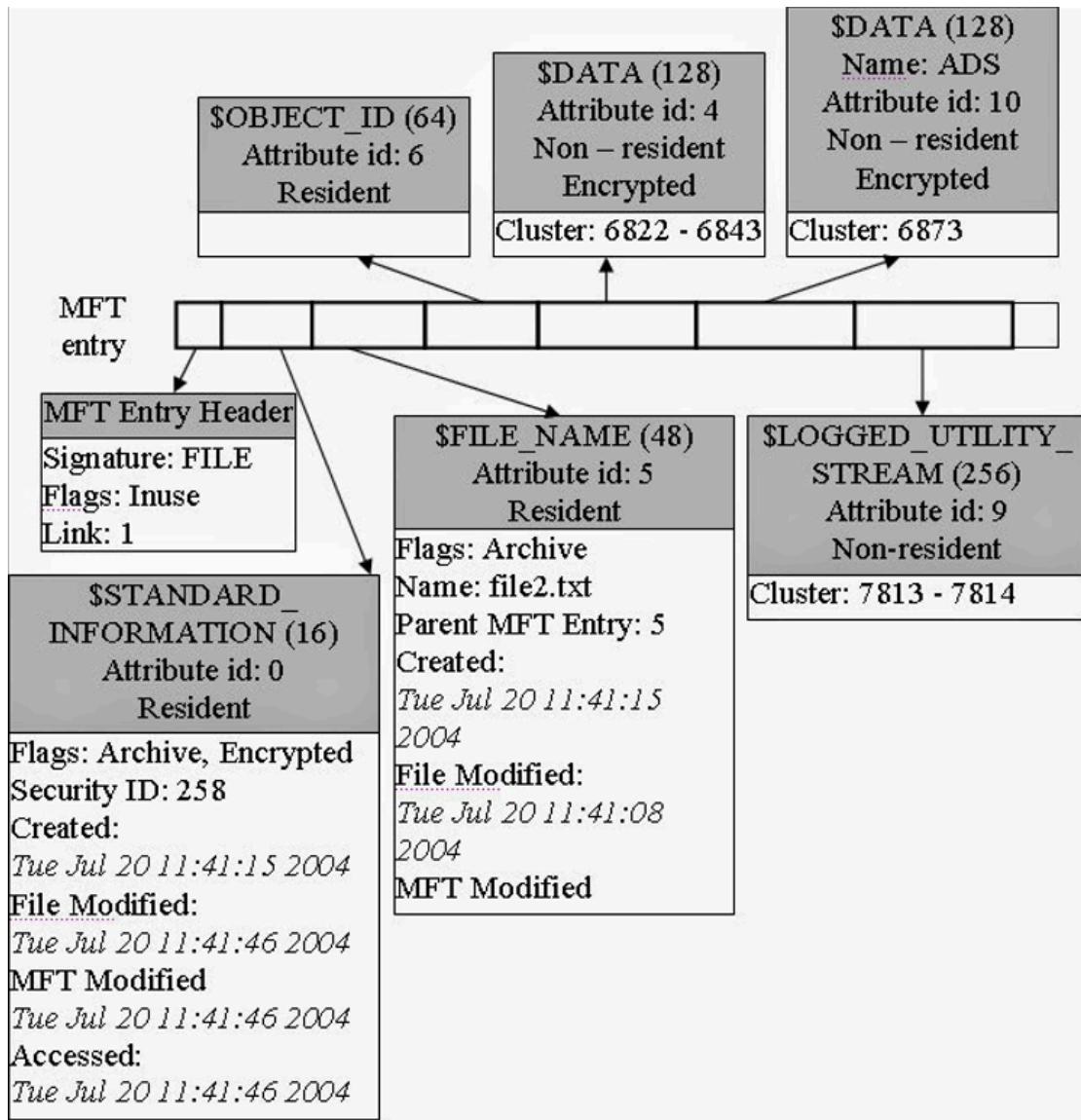
### c. Attribute \$DATA

Attribute \$DATA được sử dụng để lưu trữ tất cả các loại dữ liệu. Mã loại của attribute này là 128. Attribute \$DATA không có kích thước cố định. Mỗi tập tin có thể gồm một hoặc nhiều attribute \$DATA. Attribute \$DATA đầu tiên không có tên, các attribute \$DATA tiếp sau phải được đặt tên cụ thể.

Trong hệ thống Windows, attribute \$DATA cũng được tạo thêm khi người dùng nhập thông tin vào mục “Summary” (chuột phải vào tập tin\ chọn properties\ Summary), được tạo thêm do trình anti-virus, hoặc được tạo thêm do chương trình sao lưu dự phòng...v.v.

Attribute \$DATA có thể được mã hóa để đảm bảo an toàn thông tin. Khi được mã hóa, “cờ báo” trong header của attribute sẽ được thiết lập, “khóa” của quá trình mã hóa được lưu trong attribute \$LOGGED.Utility\_Stream.

Hình bên dưới minh họa một tập tin với hai attribute \$DATA đã được mã hóa



Cấu trúc của attribute \$DATA khá đơn giản. Sau phần header của attribute là phần nội dung, đây là dữ liệu ở dạng thô của một tập tin. Attribute \$DATA không có kích thước tối thiểu và tối đa. Nếu kích thước phần nội dung vượt quá 700 byte, attribute sẽ được chuyển từ loại resident sang loại non-resident. Trong hầu hết các tập tin, attribute \$DATA luôn nằm ở vị trí sau cùng trong MFT entry.

Phần sau đây trình bày quá trình đọc attribute \$DATA, xem hình minh họa bên dưới.

Đọc header của attribute nằm kế sau attribute \$FILE\_NAME, gồm 16 byte. Xét mã loại, byte 0 → 3 có giá trị là  $0x00000040 = 64$ , kết luận: đây là attribute \$OBJECT\_ID. Tính kích thước của attribute \$OBJECT\_ID để nhảy qua attribute kế tiếp, byte 4 → 7 có giá trị là  $0x00000028 = 40$  byte.

Vậy ta sẽ bỏ qua 40 byte của attribute \$OBJECT\_ID, để đọc header của attribute kế tiếp.

Physical Sector: Absolute Sector 6,312,933		
00000000: 46 49 4C 45 30 00 03 00 - 7B 35 34 09 00 00 00 00		FILE0...{54.....
00000010: 0B 00 01 00 38 00 01 00 - 68 01 00 00 00 04 00 00		....8...h.....
00000020: 00 00 00 00 00 00 00 00 - 04 00 00 00 D3 29 00 00		.....).
00000030: 06 00 00 00 00 00 00 00 - 10 00 00 00 E0 00 00 00		.....
00000040: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00		.....
00000050: E4 73 BB 00 2F Attribute \$FILE_NAME 35 CF 01		.....H.....
00000060: 54 33 88 04 2F 35 CF 01 - E4 73 BB 00 2F 35 CF 01		.s.../5.....5..
00000070: 20 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00		T3.../5...s.../5..
00000080: 00 00 00 00 D9 02 00 00 - 00 00 00 00 00 00 00 00		.....
00000090: 10 EC BA 00 00 00 00 00 - 30 00 00 00 70 00 00 00		.....0...p..
000000A0: 00 00 00 00 00 00 02 00 - 52 00 00 00 18 00 01 00		.....R.....
000000B0: E4 73 BB 00 2F Attribute \$OBJECT_ID 35 CF 01		.....s.../5..
000000C0: E4 73 BB 00 2F 35 CF 01		.s.../5...s.../5..
000000D0: E4 73 BB 00 2F 35 CF 01		.s.../5.....
000000E0: 00 00 00 00 00 00 00 00 - 20 00 00 00 00 00 00 00		.....
000000F0: Attribute \$DATA 74 00 2E 00 74 00 78 00		..T.e.s.t..t.x.
00000100: 74 00 2E 00 74 00 78 00 - 40 00 00 00 28 00 00 00		t.....@...(
00000110: 00 00 00 00 00 00 03 00 - 10 00 00 00 18 00 00 00		.....
00000120: A0 B9 07 6F 59 26 E3 11 - BC E2 00 0C 29 02 BC 73		...oY&.....)..s
00000130: 80 00 00 00 30 00 00 00 - 00 00 18 00 00 00 01 00		.....0.....
00000140: 15 00 00 00 18 00 00 00 - 44 75 20 6C 69 65 75 20		.....Du lieu
00000150: 64 61 6E 67 20 76 61 62 - 20 62 61 6E 21 00 00 00		dang van ban!...
00000160: FF FF FF FF 82 79 47 11 - 00 00 00 00 00 00 00 00		.....yG.....
00000170: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00		.....

Xét mã loại, byte 0 -> 3 có giá trị 0x00000080 = 128, kết luận: đây là attribute \$DATA.

Bảng sau là nội dung header của attribute \$DATA.

Byte thứ	Giá trị (Hệ 16 – Hệ 10)	Mô tả
0 – 3	0x00000080 – 128	Mã loại là 128
4 – 7	0x00000030 – 48	Kích thước của attribute \$DATA là 48 byte
8 – 8	0x00 – 0	Attribute thuộc kiểu resident
9 – 9	0x00 – 0	Attribute này không được đặt tên, nên không có giá trị chiều dài của tên.
10 – 11	0x0018 – 24	Attribute này không được đặt tên, nhưng vẫn có thông tin về vị trí của tên?

12 – 13	0x0000 – 0	Giá trị cờ báo.
14 – 15	0x0001 – 1	Định danh của attribute (attribute ID) \$DATA là 1.

Đọc tiếp byte thứ 16 -> 19 để biết kích thước phần nội dung, 0x00000015 = 21 byte.

Đọc byte thứ 20 -> 21 để biết vị trí bắt đầu của phần nội dung, 0x0018 = 24, tức là bắt đầu từ byte thứ 24 tính từ đầu attribute.

Nội dung của tập tin văn bản là “Du lieu dang van ban!”.

Cuối cùng, do chưa sử dụng hết 1024 byte của MTF entry, hệ thống sử dụng giá trị 0xFFFFFFFF để đánh dấu kết thúc phần nội dung của MFT entry.

**Lưu ý:** Trong trường hợp kích thước phần nội dung của \$DATA quá lớn, cờ Non-resident sẽ được bật lên 1. Khi đó nội dung phần \$DATA sẽ được lưu trong các cluster bên ngoài. Để biết thông tin các cluster lưu dữ liệu, ta đọc thông tin Data run của phần nội dung. Ví dụ về cấu trúc của 1 Data run có dạng

### 21 01 88 15

Trong đó:

- + Byte đầu tiên (**0x21**): Mô tả cấu trúc của Data run, 4 bit đầu tiên (có giá trị 2) là số Byte cần để mô tả chỉ số Offset của Cluster bắt đầu dữ liệu, 4 bit sau (có giá trị 1) là số Byte cần để mô tả kích thước của dữ liệu (số lượng Cluster)
- + 1 Byte tiếp theo (**0x01**): Kích thước của dữ liệu (có giá trị là 1)
- + 2 Byte tiếp theo (**0x8815**): Chỉ số Offset của Cluster bắt đầu dữ liệu (lưu dưới dạng Little Endian, có giá trị là 5512)

Ví dụ dưới đây là 1 Data run có dạng **31 07 4C A1 02** của file 22127154@6.sql

Attribute \$80	110		46 49 4C 45   30 00 03 00   F8 49 00 01 00 00 00 00   ...   FILE0...øI.....
Attribute type	110	0x80	01 00   01 00   38 00   01 00   60 01 00 00   00 04 00 00   ...   ... .8...`....
Length (including header)	114	72	00 00 00 00 00 00 00 00   03 00 00 00 2A 00 00 00   ...   ..... ....*
Non-resident flag	118	1	02 00   00 00 00 00 00 00   10 00 00 00 60 00 00 00   ...   .. .... ....`...
Name length	119	0	00 00 00 00 00 00 00 00   48 00 00 00 18 00 00 00   ...   .....H.....
Name offset	11A	0x00	56 64 01 3F B5 75 DA 01   E2 7E 08 6F A5 73 DA 01   ...   Vd.?muÚ.â~.oÙsÚ.
Flags	11C	00 00	BB D1 0B 6F A5 73 DA 01   OE BE 02 3F B5 75 DA 01   ...   »Ñ.oÙsÚ..ñ. ?muÚ.
Attribute ID	11E	1	20 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00   ...   ..... ..... ..... .....
First VCN	120	0	00 00 00 00 08 01 00 00   00 00 00 00 00 00 00 00   ...   0...x... ..... .....
Last VCN	128	6	00 00 00 00 00 00 00 00   30 00 00 00 78 00 00 00   ...   ^..... ..... .....
Data runs offset	130	0x40	00 00 00 00 00 00 02 00   5E 00 00 00 18 00 01 00   ...   .Vd.?muÚ.
Compression unit size	132	0	05 00 00 00 00 00 05 00   56 64 01 3F B5 75 DA 01   ...   Vd.?muÚ.Vd.?muÚ.
Padding	134	00 00 00 00	56 64 01 3F B5 75 DA 01   56 64 01 3F B5 75 DA 01   ...   Vd.?muÚ..p.....
Allocated size	138	28,672	56 64 01 3F B5 75 DA 01   00 70 00 00 00 00 00 00   ...   ..2.2.1.2.7.1.5.
Real size	140	25,990	00 00 00 00 00 00 00 00   20 00 00 00 00 00 00 00   ...   4.0.6...s.q.1...
Initialized size	148	25,990	0E 00 32 00 32 00 31 00   32 00 37 00 31 00 35 00   ...   ....H.....
\$DATA	150		34 00 40 00 36 00 2E 00   73 00 71 00 6C 00 00 00   ...   @..... ..... .....
Data run	150		80 00 00 00 48 00 00 00   01 00 00 00 00 00 01 00   ...   ..e..... ..... .....
Size	150	0x31	00 00 00 00 00 00 00 00   06 00 00 00 00 00 00 00   ...   @..... ..... .....
Cluster count	151	7	40 00 00 00 00 00 00 00   00 70 00 00 00 00 00 00   ...   ..e..... ..... .....
First cluster	152	172,364	86 65 00 00 00 00 00 00   86 65 00 00 00 00 00 00   ...   1[E1]...ÿÿÿÿ.yG.
End marker	158	0xFFFFFFFF	31 07 4C A1 02 00 00 00   FF FF FF FF 82 79 47 11   ...

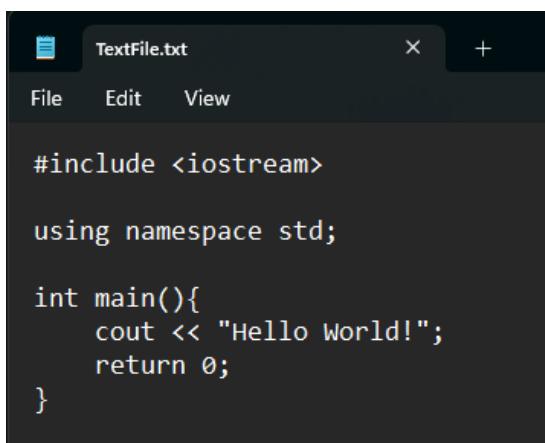
Từ thông tin về chỉ số Cluster bắt đầu, ta có thể tìm đến Cluster đó và đọc nội dung file với kích thước bằng số lượng Cluster được mô tả trong Data run. Hình bên dưới là nội dung của sector đầu tiên thuộc chuỗi Cluster nội dung.

02A150000	EF BB BF 2D 20 51 35 38 3A 20 43 68 6F 20 62	i»¿-- Q58: Cho b
02A150010	69 E1 BA BF 74 20 74 C3 AA 6E 20 67 69 C3 A1 6F	iáº¿t tÃªn giÃ¡o
02A150020	20 76 69 C3 AA 6E 20 6E C3 A0 6F 20 6D C3 A0 20	viÃªn nÃ o mÃ
02A150030	74 68 61 6D 20 67 69 61 20 C4 91 E1 BB 81 20 74	tham gia Ä.á». t
02A150040	C3 A0 69 20 C4 91 E1 BB A7 20 74 E1 BA A5 74 20	Ä i Ä.á»S tÃºt
02A150050	63 E1 BA A3 20 63 C3 A1 63 20 63 68 E1 BB A7 20	cáº£ cÃ¡c chÃ¢»S
02A150060	C4 91 E1 BB 81 0D 0A 0D 0A 2D 2D 20 45 58 43 45	Ä.á».....-- EXCE
02A150070	50 54 20 0D 0A 0D 0A 53 45 4C 45 43 54 20 0D 0A	PT ....SELECT ..
02A150080	09 47 56 2E 4D 41 47 56 20 41 53 20 4E 27 4D C3	.GV.MAGV AS N'MÃ
02A150090	A3 20 47 56 27 2C 0D 0A 09 47 56 2E 48 4F 54 45	£ GV',...GV.HOTE
02A1500A0	4E 20 41 53 20 4E 27 48 E1 BB 8D 20 76 C3 A0 20	N AS N'HÃ». vÃ
02A1500B0	74 C3 AA 6E 27 0D 0A 46 52 4F 4D 20 54 48 41 4D	tÃªn'..FROM THAM
02A1500C0	47 49 41 44 54 20 41 53 20 54 47 31 0D 0A 4A 4F	GIADT AS TG1..JO
02A1500D0	49 4E 20 47 49 41 4F 56 49 45 4E 20 41 53 20 47	IN GIAOVIE AS G
02A1500E0	56 20 4F 4E 20 47 56 2E 4D 41 47 56 20 3D 20 54	V ON GV.MAGV = T
02A1500F0	47 31 2E 4D 41 47 56 0D 0A 57 48 45 52 45 20 4E	G1.MAGV..WHERE N
02A150100	4F 54 20 45 58 49 53 54 53 20 28 0D 0A 09 28 53	OT EXISTS (...(S
02A150110	45 4C 45 43 54 20 4D 41 44 54 20 46 52 4F 4D 20	ELECT MADT FROM
02A150120	44 45 54 41 49 29 0D 0A 09 45 58 43 45 50 54 20	DETAI)...EXCEPT
02A150130	0D 0A 09 28 53 45 4C 45 43 54 20 4D 41 44 54 20	... (SELECT MADT
02A150140	0D 0A 09 46 52 4F 4D 20 54 48 41 4D 47 49 41 44	...FROM THAMGIAD
02A150150	54 20 41 53 20 54 47 32 0D 0A 09 57 48 45 52 45	T AS TG2...WHERE
02A150160	20 54 47 32 2E 4D 41 47 56 20 3D 20 54 47 31 2E	TG2.MAGV = TG1.
02A150170	4D 41 47 56 29 0D 0A 09 29 0D 0A 0D 0A 2D 2D 20	MAGV)...)--
02A150180	4E 4F 54 20 45 58 49 53 54 53 0D 0A 0D 0A 53 45	NOT EXISTS....SE
02A150190	4C 45 43 54 20 0D 0A 09 47 56 2E 4D 41 47 56 20	LECT ...GV.MAGV
02A1501A0	41 53 20 4E 27 4D C3 A3 20 47 56 27 2C 0D 0A 09	AS N'MÃ GV',...
02A1501B0	47 56 2E 48 4F 54 45 4E 20 41 53 20 4E 27 48 E1	GV.HOTEN AS N'HÃ
02A1501C0	BB 8D 20 76 C3 A0 20 74 C3 AA 6E 27 0D 0A 46 52	».. vÃ tÃªn'..FR
02A1501D0	4F 4D 20 54 48 41 4D 47 49 41 44 54 20 41 53 20	OM THAMGIADT AS
02A1501E0	54 47 31 0D 0A 4A 4F 49 4E 20 47 49 41 4F 56 49	TG1..JOIN GIAOVI
02A1501F0	45 4E 20 41 53 20 47 56 20 4F 4E 20 47 56 2E 4D	EN AS GV ON GV.M

**Lưu ý:** Mỗi Entry có thể có nhiều Attribute \$DATA, tương đương với việc có nhiều Data run

#### 1.2.4 Ví dụ minh họa về MFT Entry

Trong phần này, ta xét tập tin TextFile.txt có nội dung như sau



```
#include <iostream>

using namespace std;

int main(){
    cout << "Hello World!";
    return 0;
}
```

Trong ổ địa vật lí, TextFile.txt được lưu trong 1 MFT Entry như sau (1 MFT Entry gồm 1024 byte ~ 2 sector nhưng vì phần nội dung của Entry này không vượt quá sector đầu tiên nên hình bên dưới chỉ gồm nội dung sector đầu thuộc MFT Entry)

Offset	00 01 02 03 04 05 06 07	08 09 0A 0B 0C 0D 0E 0F	ASCII	Unicode
0C000E400	46 49 4C 45 30 00 03 00	AE 2D 80 01 00 00 00 00	FILE0...Ø-	..0...b..
0C000E410	01 00 01 00 38 00 01 00	C8 01 00 00 00 04 00 00	.L...8...È...Ø...	..8.Ij.È..
0C000E420	00 00 00 00 00 00 00 00	05 00 00 00 29 00 00 00	.....Ø...)...	.....).
0C000E430	25 00 00 00 00 00 00 00	10 00 00 00 60 00 00 00	Ø...Ø...Ø...`...	Ø.....`..
0C000E440	00 00 00 00 00 00 00 00	48 00 00 00 18 00 00 00	.....H...	....H...
0C000E450	A7 EE 5C BD 7B 74 DA 01	C8 CF E7 69 07 75 DA 01	\$i\{tÚ.ÈÍç.i.uÚ.	....ü...ü
0C000E460	C8 CF E7 69 07 75 DA 01	3F EE 38 AA 07 75 DA 01	ÈÍç.i.uÚ.?í8ª.uÚ.	....ü...ü
0C000E470	20 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	.....	.....
0C000E480	00 00 00 00 09 01 00 00	00 00 00 00 00 00 00 00	.....	.....
0C000E490	00 00 00 00 00 00 00 00	30 00 00 00 78 00 00 00	0...x...	....0.x..
0C000E4A0	00 00 00 00 00 03 00	5A 00 00 00 18 00 01 00	.....Z...	....z...
0C000E4B0	05 00 00 00 00 05 00	A7 EE 5C BD 7B 74 DA 01	\$i\{tÚ.Øi\{tÚ.	....ü...ü
0C000E4C0	A7 EE 5C BD 7B 74 DA 01	A7 EE 5C BD 7B 74 DA 01	\$i\{tÚ.Øi\{tÚ.	....ü...ü
0C000E4D0	A7 EE 5C BD 7B 74 DA 01	00 00 00 00 00 00 00 00	\$i\{tÚ.....	....ü....
0C000E4E0	00 00 00 00 00 00 00 00	20 00 00 00 00 00 00 00	.....	.....
0C000E4F0	0C 00 54 00 65 00 78 00	74 00 46 00 69 00 6C 00	.T.e.x.t.F.i.l.	.TextFil
0C000E500	65 00 2E 00 74 00 78 00	74 00 00 00 00 00 00 00	e....t.x.t.....	e.txt...
0C000E510	40 00 00 00 28 00 00 00	00 00 00 00 00 00 04 00	@...(......	@.(.....
0C000E520	10 00 00 00 18 00 00 00	CF D7 F4 E2 B5 DF EE 11	.....Í×ôåµßi.	.....
0C000E530	95 E5 E0 25 28 BB 5C 5A	80 00 00 00 88 00 00 00	åà%»\Z.....	.....
0C000E540	00 00 18 00 00 00 01 00	69 00 00 00 18 00 00 00	.....i.....	....i...
0C000E550	23 69 6E 63 6C 75 64 65	20 3C 69 6F 73 74 72 65	#include <iostre	.....
0C000E560	61 6D 3E 0D 0A 0D 0A 75	73 69 6E 67 20 6E 61 6D	am>....using nam	.....
0C000E570	65 73 70 61 63 65 20 73	74 64 3B 0D 0A 0D 0A 69	espace std;....i	.....
0C000E580	6E 74 20 6D 61 69 6E 28	29 7B 0D 0A 20 20 20 20	nt main(){...	.....tt
0C000E590	63 6F 75 74 20 3C 3C 20	22 48 65 6C 6C 6F 20 57	cout << "Hello W	....!!....
0C000E5A0	6F 72 6C 64 21 22 3B 0D	0A 20 20 20 20 72 65 74	orld!";... ret	.....t..
0C000E5B0	75 72 6E 20 30 3B 0D 0A	7D 00 00 00 00 00 00 00	urn 0;...}.....	.....}...
0C000E5C0	FF FF FF FF 82 79 47 11	00 00 00 00 00 00 00 00	ÿÿÿÿ.yG.....	.....
0C000E5D0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	.....	.....
0C000E5E0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	.....	.....
0C000E5F0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 25 00	.....%	.....%

Nhìn hình trên, ta thấy được MFT Entry được chia thành 5 vùng theo thứ tự lần lượt là:

- Header của Entry
- Attribute \$10 (\$STANDARD\_INFORMATION)
- Attribute \$30 (\$FILE\_NAME)
- Attribute \$40
- Attribute \$80 (\$DATA)

Cụ thể, thông tin thuộc tính từng vùng như sau:

a. *Header của Entry*

Name	Offset	Value
... Signature (must be 'FILE')	000	FILE
... Offset to the update sequence	004	0x30
... Update sequence size in words	006	3
... \$LogFile Sequence Number (L...	008	25,177,518
... Sequence number	010	1
... Hard link count	012	1
... <b>Offset to the first attribute</b>	<b>014</b>	<b>0x38</b>
Flags	016	01 00
In use	:0	1
Directory	:1	0
Real size of the FILE record	018	456
Allocated size of the FILE record	01C	1,024
Base FILE record	020	0
Next attribute ID	028	5
ID of this record	02C	41
Update sequence number	030	25 00
Update sequence array	032	00 00 00 00
<b>Attribute \$10</b>	<b>038</b>	
<b>Attribute \$30</b>	<b>098</b>	
<b>Attribute \$40</b>	<b>110</b>	
<b>Attribute \$80</b>	<b>138</b>	
End marker	1C0	0xFFFFFFFF

Trong các thông tin trên, thông tin quan trọng nhất là “Offset to the first attribute” tại Offset 0x14 có giá trị là **0x38** (là giá trị Offset của Attribute đầu tiên - Attribute \$10)

b. Attribute \$10 (\$STANDARD\_INFORMATION)

Attribute \$10	038	
Attribute type	038	0x10
Length (including header)	03C	96
Non-resident flag	040	0
Name length	041	0
Name offset	042	0x00
Flags	044	00 00
Compressed	:0	0
Encrypted	:14	0
Sparse	:15	0
Attribute ID	046	0
Length of the attribute	048	72
Offset to the attribute data	04C	0x18
Indexed flag	04E	0
Padding	04F	0
\$STANDARD_INFORMATION	050	
File created (UTC)	050	3/12/2024 12:49 PM
File modified (UTC)	058	3/13/2024 5:29 AM
Record changed (UTC)	060	3/13/2024 5:29 AM
Last access time (UTC)	068	3/13/2024 5:31 AM
File Permissions	070	20 00 00 00
Read-Only	:0	0
Hidden	:1	0
System	:2	0
Archive	:5	1
Device	:6	0
Normal	:7	0
Temporary	:8	0
Sparse File	:9	0
Reparse Point	:10	0
Compressed	:11	0
Offline	:12	0
Not Content Indexed	:13	0
Encrypted	:14	0
Maximum number of ...	074	0
Version number	078	0
Class Id	07C	0
Owner Id	080	0
Security Id	084	265
Quota Charged	088	0
Update Sequence Num...	090	0

Attribute **\$STANDARD\_INFORMATION** sẽ gồm phần Header và nội dung của Attribute (được đánh dấu bằng chữ **\$STANDARD\_INFORMATION** như trên hình). Trong đó thông tin quan trọng nhất là giá trị cờ *File Permissions* tại *Offset 0x70*, có giá trị là **20 00 00 00** tương ứng với ý nghĩa đây là **Tập tin (Archive)**

### c. Attribute \$30 (\$FILE\_NAME)

Attribute \$30	098	
Attribute type	098	0x30
Length (including header)	09C	120
Non-resident flag	0A0	0
Name length	0A1	0
Name offset	0A2	0x00
Flags	0A4	00 00
Attribute ID	0A6	3
Length of the attribute	0A8	90
Offset to the attribute data	0AC	0x18
Indexed flag	0AE	1
Padding	0AF	0
\$FILE_NAME	0B0	
Parent directory file record number	0B0	5
Parent directory sequence number	0B6	5
File created (UTC)	0B8	3/12/2024 12:49 PM
File modified (UTC)	0C0	3/12/2024 12:49 PM
Record changed (UTC)	0C8	3/12/2024 12:49 PM
Last access time (UTC)	0D0	3/12/2024 12:49 PM
Allocated size	0D8	0
Real size	0E0	0
File attributes (used by EAs and reparse)	0E8	20 00 00 00
File name length	0F0	12
File name namespace	0F1	0
File name	0F2	TextFile.txt

Tương tự như trên, Attribute \$30 này cũng gồm Header và nội dung. Trong đó một số thông tin quan trọng thuộc vùng **\$FILE\_NAME** được đánh dấu trên hình như:

- + Parent directory file record number (**0xB**): chứa địa chỉ record của thư mục cha của file này
- + File name length (**0xF**): độ dài của tên
- + File name namespace (**0xF1**): namespace của tên file
- + File name (**0xF2**): tên file (lưu ý, mỗi kí tự trong filename có kích thước 2 byte)

#### d. Attribute \$40

<input type="checkbox"/> Attribute \$40	110	
Attribute type	110	0x40
Length (including header)	114	40
Non-resident flag	118	0
Name length	119	0
Name offset	11A	0x00
<input type="checkbox"/> Flags	11C	00 00
Attribute ID	11E	4
Length of the attribute	120	16
Offset to the attribute data	124	0x18
Indexed flag	126	0
Padding	127	0
<input type="checkbox"/> \$OBJECT_ID	128	
GUID Object Id	128	CF D7 F4 E2 B5 DF EE 11 95 E5 E0 25 28 BB 5C 5A

Trong Attribute \$40 này, ta chỉ cần quan tâm đến thông tin **Length** tại Offset **0x114** (có giá trị là 40 ) để bỏ qua Attribute này và đến Attribute kế tiếp

#### e. Attribute \$80 (\$DATA)

<input type="checkbox"/> Attribute \$80	138	
Attribute type	138	0x80
Length (including header)	13C	136
Non-resident flag	140	0
Name length	141	0
Name offset	142	0x18
<input type="checkbox"/> Flags	144	00 00
Attribute ID	146	1
Length of the attribute	148	105
Offset to the attribute data	14C	0x18
Indexed flag	14E	0
Padding	14F	0
<input type="checkbox"/> \$DATA	150	
Data	150	23 69 6E 63 6C 75 64 65 20 3C 69 6F 73 74 72 6...
End marker	1C0	0xFFFFFFFF

95 E5 E0 25 28 BB 5C 5A	80 00 00 00 00 88 00 00 00	.à à à (»\Z.....
00 00 18 00 00 00 01 00	69 00 00 00 18 00 00 00	.....i.....
23 69 6E 63 6C 75 64 65	20 3C 69 6F 73 74 72 65	#include <iostre
61 6D 3E 0D 0A 0D 0A 75	73 69 6E 67 20 6E 61 6D	am>....using nam
65 73 70 61 63 65 20 73	74 64 3B 0D 0A 0D 0A 69	espace std;....i
6E 74 20 6D 61 69 6E 28	29 7B 0D 0A 20 20 20 20	nt main(){..
63 6F 75 74 20 3C 3C 20	22 48 65 6C 6C 6F 20 57	cout << "Hello W
6F 72 6C 64 21 22 3B 0D	0A 20 20 20 20 72 65 74	orld!";... ret
75 72 6E 20 30 3B 0D 0A	7D 00 00 00 00 00 00 00 00	urn 0;...}.....
FF FF FF FF	FF 82 79 47 11 00 00 00 00 00 00 00 00 00 00 00 00	ÿÿÿÿ.yG.....
00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....

Trong Attribute \$80 (\$DATA) này lưu nội dung của TextFile.txt, dựa vào các thông tin ở Header: **Length of the Attribute** tại **0x148** và **Offset to the attribute data** tại **0x14C**, ta đọc được nội dung của file trong phần \$DATA

Cuối cùng là 4 byte đánh dấu kết thúc Entry (**0xFFFFFFFF**) để biết được phần nội dung Entry đã kết thúc

## 2. Hiển thị thư mục gốc

Để thực hiện yêu cầu “Hiển thị thư mục gốc” và “Truy xuất cây thư mục”. Vì mỗi MFT Entry đều có một cặp **ID** và **ParentID** phân biệt nên một trong những cách làm đơn giản là xây dựng cấu trúc cây dựa trên các MFT Entry.

Dưới đây là hình ảnh minh họa về cấu trúc của 1 Entry bằng C++

```
class Entry
{
public:
    uint16_t startOffsetOfAttribute; // offset from the start of the file to the attribute
    uint16_t flags; // 0x01 = used, 0x02 = directory, 0x04, 0x08 = unknown
    uint64_t baseEntryRef; // reference to the base entry
    uint32_t id; // id of the entry
    std::vector<Attribute *> attributes; // attributes of the entry
public:

    uint64_t GetParentDirectory() const;
};
```

Class Entry sẽ chứa một số thông tin của phần Header và danh sách các Attribute. Trong đó thông tin giá trị ID nằm ở Header của Entry, còn ParentID nằm ở Attribute \$FILE\_NAME

Dựa vào đó, ta có thể xây dựng cấu trúc EntryNode tham chiếu đến Entry như sau:

```
struct EntryNode
{
    std::string name;
    std::vector<EntryNode *> children;
    Entry *entry;
    EntryNode *parent;

    EntryNode(Entry *entry, EntryNode *parent)
    {
        this->entry = entry;
        this->name = entry->GetFileName();
        this->parent = parent;
    }
    uint32_t GetID() const
    {
        return entry->id;
    }
    uint32_t GetParentID() const
    {
        return parent->entry->id;
    }
    bool IsDirectory() const
    {
        return entry->GetStandardInformationFlags() == 0x0000;
    }
    bool IsFile() const
    {
        return entry->GetStandardInformationFlags() == 0x20;
    }
};
```

Mỗi EntryNode sẽ giữ thông tin của 1 Entry, đồng thời có liên kết để dẫn tới Entry cha và các Entry con nằm hỗ trợ cho thao tác truy cập thư mục con và quay về thư mục cha

Như vậy, dựa vào các xây dựng trên, sau khi đọc toàn bộ thông tin ở mục 1. Và xây dựng được các cấu trúc Entry và EntryNode, để truy xuất thư mục gốc (hoặc bất kì một thư mục nào), ta chỉ cần tìm đến EntryNode có tên là dữ liệu đầu vào, sau đó in ra danh sách các EntryNode con.

### 3. Truy xuất cây thư mục

Dựa vào các thông tin bên trên cùng các cấu trúc đã được xây dựng, ta có thể biết được mỗi EntryNode là tập tin hay thư mục (dựa vào các hàm kiểm tra trong cấu trúc).

Để truy xuất nội dung của 1 tập tin trong thư mục hiện tại, ta tìm EntryNode con có tên bằng tên tập tin đầu vào. Sau đó kiểm tra xem đuôi tập tin có phải là ".txt" không, nếu có ta sẽ in ra nội dung tập tin được lưu trong Attribute \$DATA của Entry, nếu không phải thì in ra thông báo yêu cầu người dùng dùng phần mềm khác để đọc.

Để truy xuất nội dung của 1 thư mục trong thư mục hiện tại, ta cũng tìm đến EntryNode con có tên bằng tên thư mục đầu vào. Sau đó kiểm tra xem EntryNode này có phải thư mục hay không, nếu có thì chuyển hướng chương trình đến thư mục con đó và cho phép người dùng thực hiện các thao tác tương tự (có thể quay lại thư mục cha trước đó)

## V.Hình ảnh demo chương trình

### A. FAT32

#### 1. Đọc thông tin FAT32 (Boot Sector, FAT, RDET)

Khi bắt đầu chương trình, người dùng sẽ nhập ổ đĩa cần đọc thông tin

```
=====
[System] Restart
=====
What disk do you want to choose ?
 [0]: Refresh disk drive
 [1]: C:\ 
 [2]: D:\ 
 [3]: E:\ 
3|
```

## 2. Hiển thị cây thư mục gốc

Chương trình chính sẽ có 5 tùy chọn để người dùng thao tác được mô tả như bên dưới. Nếu ta chọn nút 2, chương trình sẽ liệt kê cây thư mục hiện tại (gốc)



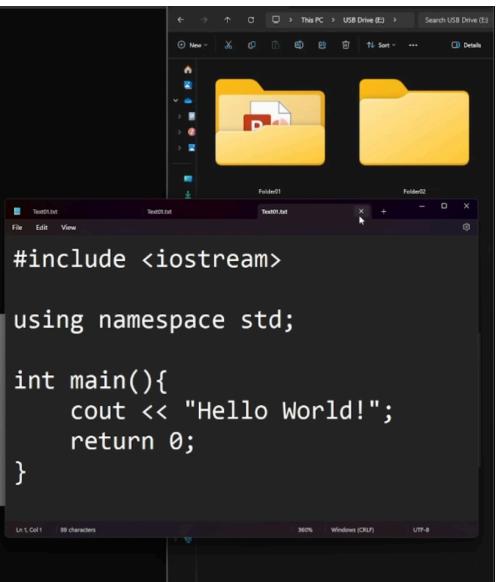
```

=====
[System] Help
=====
-----A CODE.CPP
-----D- FOLDER01
-----D- FOLDER02
-----D- FOLDER03
-----A PowerPoint.pptx
-HS-D- System Volume Information
-----A TEXT01.TXT
What do you want to do ?
[0]: </>          typing commands by your own
[1]: -restart      restart the app to choose another d
rive
[2]: -dir          to show all files and folders
[3]: -open filename.txt   read the content inside the txt fil
e named "filename"
[4]: -cd ./path/to/folder  change the current directory into t
he file named "folder" of given path

```

## 3. Truy xuất cây thư mục

Khi chọn nút 3 và nhập vào tên tập tin cần đọc, nếu đó là tập tin .txt, chương trình sẽ hiển thị nội dung tập tin. Các kí hiệu -HS-DA tương ứng với (Hidden, System, Directory, Archive)



```

=====
[System] Open
=====
-----A CODE.CPP
-----D- FOLDER01
-----D- FOLDER02
-----D- FOLDER03
-----A PowerPoint.pptx
-HS-D- System Volume Information
-----A TEXT01.TXT
Input File Full Name: TEXT01.TXT
#include <iostream>
using namespace std;

int main(){
    cout << "Hello World!";
    return 0;
}
Press any key to continue

```

Khi chọn nút 4 và nhập vào tên thư mục / đường dẫn, chương trình sẽ chuyển hướng đến thư mục con đã được nhập, các thao tác còn lại tương tự như thư mục gốc (vẫn có thể tập tin đọc nội dung và vào các thư mục bên trong)

```
=====
[System] Cd
=====
----A CODE.CPP
----D- FOLDER01
----D- FOLDER02
----D- FOLDER03
----A PowerPoint.pptx
-HS-D- System Volume Information
----A TEXT01.TXT
Input Path/To/File: FOLDER01
----D- .
----D- ..
----A PowerPoint02.pptx
----D- SUB01
----D- SUB02
----A TEXT02.TXT
Press any key to continue
```

Các thao tác còn lại tương tự như thư mục gốc (vẫn có thể in ra cây thư mục, đọc nội dung tập tin và chuyển hướng đến các thư mục con, quay lại thư mục cha)

Ngoài ra, chương trình còn hỗ trợ môi trường dòng lệnh.

```
./E/FOLDER01>> cd .
Directory path: "."
Successfully changed directory path
./E/FOLDER01/.>> -dir
----D- .
----D- ..
----A PowerPoint02.pptx
----D- SUB01
----D- SUB02
----A TEXT02.TXT
./E/FOLDER01/.>> cd ..
Directory path: ".."
Successfully changed directory path
./E/FOLDER01>> -dir
----A CODE.CPP
----D- FOLDER01
----D- FOLDER02
----D- FOLDER03
----A PowerPoint.pptx
-HS-D- System Volume Information
----A TEXT01.TXT
./E/FOLDER01>> |
```

```

Successfully changed directory path
./E/FOLDER01>> -dir
-----A CODE.CPP
-----D- FOLDER01
-----D- FOLDER02
-----D- FOLDER03
-----A PowerPoint.pptx
-HS-D- System Volume Information
-----A TEXT01.TXT
./E/FOLDER01>> -open TEXT01.TXT
file_name: TEXT01 | file_extension: TXT
#include <iostream>

using namespace std;

int main(){
    cout << "Hello World!";
    return 0;
}

```



## B. NTFS

### 1. Đọc thông tin NTFS (BPB, MFT)

Khi bắt đầu chương trình, thông tin của BPB sẽ được in ra sau đó in ra bảng các nút chức năng

```

Welcome to NTFS Explorer
* NTFS BPB information:
+ Bytes per sector: 512
+ Sectors per cluster: 8
+ Start cluster of MFT: 786432
+ Bytes per entry: 1024   *

=====
=====

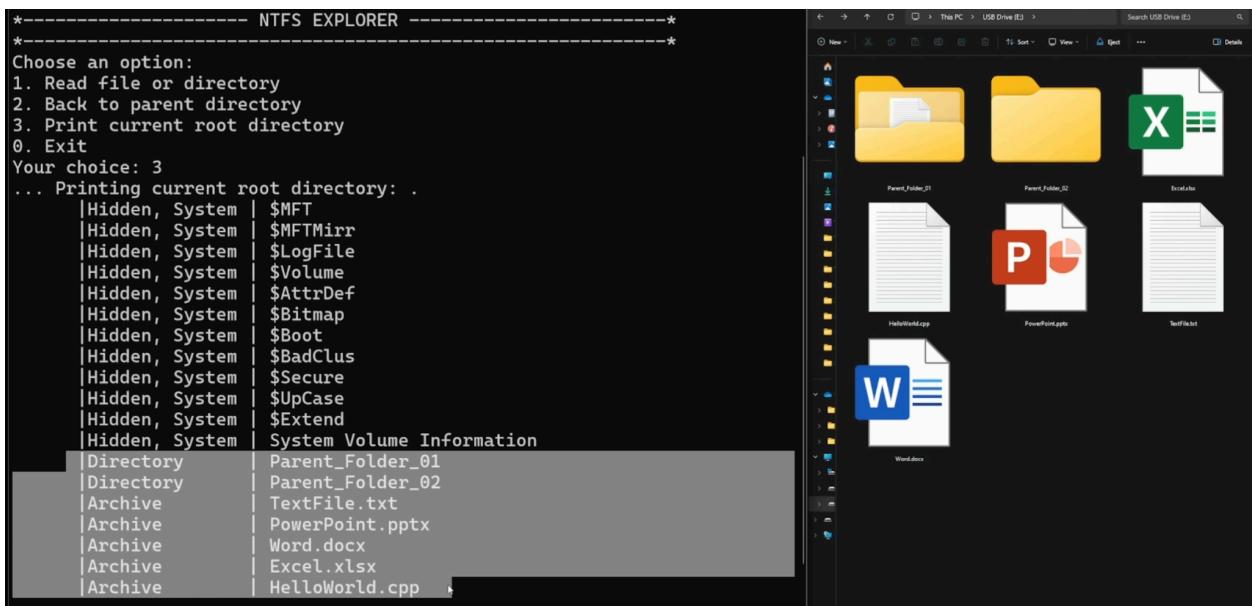
*-----*
*----- NTFS EXPLORER -----*
*-----*

Choose an option:
1. Read file or directory
2. Back to parent directory
3. Print current root directory
0. Exit
Your choice:

```

## 2. Hiển thị cây thư mục gốc

Lúc này, ta đang ở thư mục gốc, nếu chọn chức năng “3. Print current root directory” Kết quả thư mục được in ra từ thư mục gốc, bao gồm cả những tập tin, thư mục ẩn của hệ thống



The screenshot shows the NTFS Explorer interface. On the left, a terminal window displays the output of the 'Print current root directory' command. The output lists various system files and folders, including hidden ones like \$MFT and \$Volume, along with user-created files like TextFile.txt, PowerPoint.pptx, Word.docx, Excel.xlsx, and HelloWorld.cpp. On the right, a file explorer window shows the same directory structure with icons for each file and folder.

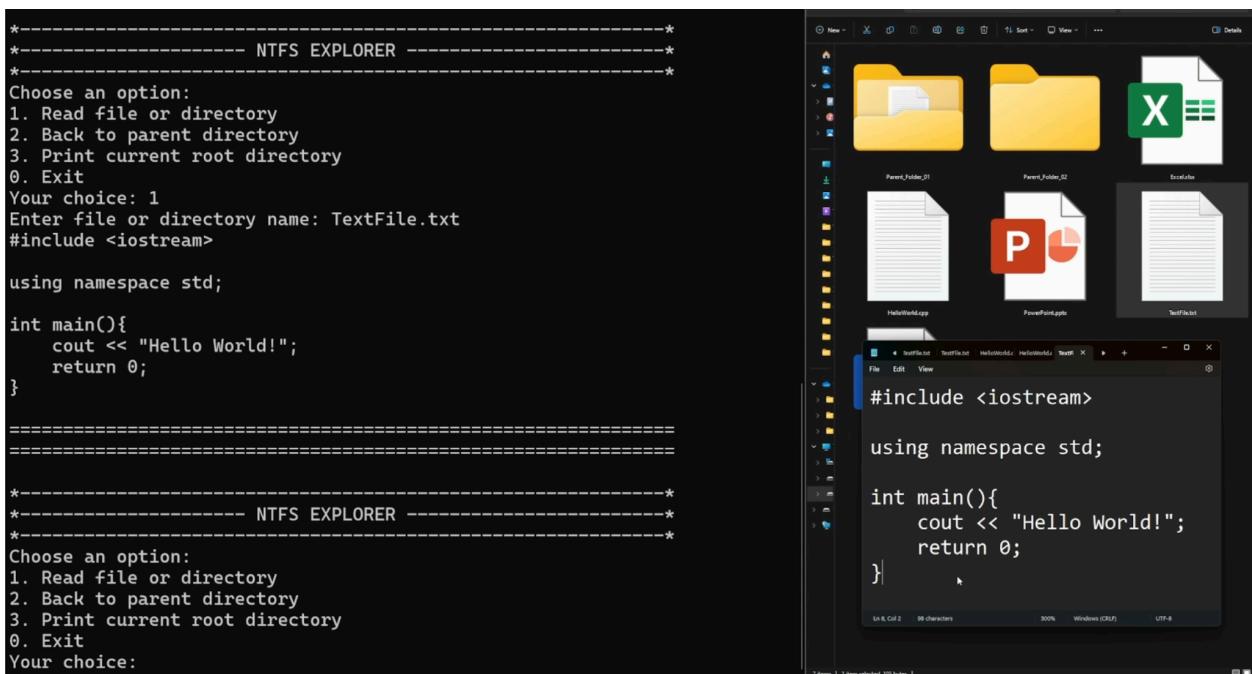
```

*----- NTFS EXPLORER -----
Choose an option:
1. Read file or directory
2. Back to parent directory
3. Print current root directory
0. Exit
Your choice: 3
... Printing current root directory: .
|Hidden, System | $MFT
|Hidden, System | $MFTMirr
|Hidden, System | $LogFile
|Hidden, System | $Volume
|Hidden, System | $AttrDef
|Hidden, System | $Bitmap
|Hidden, System | $Boot
|Hidden, System | $BadClus
|Hidden, System | $Secure
|Hidden, System | $UpCase
|Hidden, System | $Extend
|Hidden, System | System Volume Information
|Directory      | Parent_Folder_01
|Directory      | Parent_Folder_02
|Archive        | TextFile.txt
|Archive        | PowerPoint.pptx
|Archive        | Word.docx
|Archive        | Excel.xlsx
|Archive        | HelloWorld.cpp

```

## 3. Truy xuất cây thư mục

Khi yêu cầu đọc tập tin “TextFile.txt”, nội dung tập tin sẽ được in ra trên màn hình  
Nếu yêu cầu đọc tập tin “HelloWorld.cpp” thì chương trình sẽ in ra thông báo yêu cầu dùng phần mềm khác để đọc



The screenshot shows the NTFS Explorer interface on the left and a code editor window on the right. The code editor contains a simple C++ program that reads the content of TextFile.txt. The terminal window shows the command 'Read file or directory' followed by 'TextFile.txt'. The code editor displays the file's content, which is "Hello World!".

```

*----- NTFS EXPLORER -----
Choose an option:
1. Read file or directory
2. Back to parent directory
3. Print current root directory
0. Exit
Your choice: 1
Enter file or directory name: TextFile.txt
#include <iostream>

using namespace std;

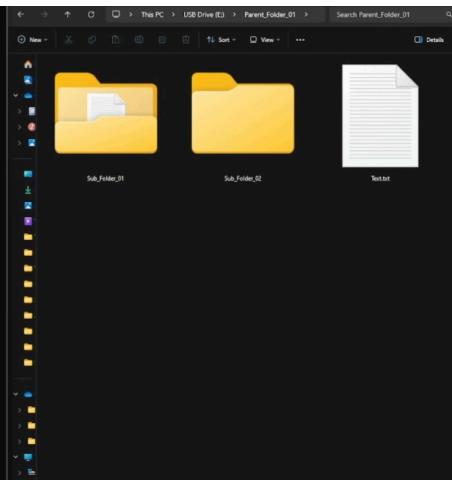
int main(){
    cout << "Hello World!";
    return 0;
}

*----- NTFS EXPLORER -----
Choose an option:
1. Read file or directory
2. Back to parent directory
3. Print current root directory
0. Exit
Your choice:

```

Khi yêu cầu đọc thư mục Parent\_Folder\_01, chương trình sẽ chuyển thư mục gốc hiện tại thành thư mục đã chọn (Parent\_Folder\_01)

Nếu ta yêu cầu in cây thư mục hiện tại, những thư mục và tập tin nằm trong cây thư mục gốc của Parent\_Folder\_01 sẽ được in ra

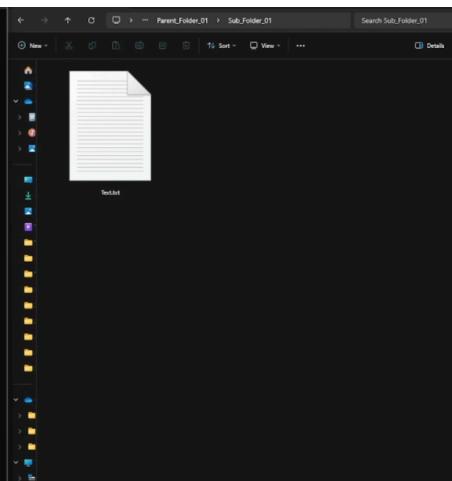


```

Enter file or directory name: Parent_Folder_01
... Change to directory Parent_Folder_01

=====
=====
*----- NTFS EXPLORER -----
*----- Choose an option:
1. Read file or directory
2. Back to parent directory
3. Print current root directory
0. Exit
Your choice: 3
... Printing current root directory: Parent_Folder_01
|Directory      | Sub_Folder_01
|Directory      | Sub_Folder_02
|Archive        | Text.txt
=====
=====
```

Tương tự, nếu ta muốn vào tiếp thư mục con Sub\_Folder\_01 của Parent\_Folder\_01, ta yêu cầu chương trình đọc thư mục Sub\_Folder\_01, chương trình sẽ chuyển hướng đến thư mục Sub\_Folder\_01 và thực hiện yêu cầu in ra cây thư mục hiện tại.



```

0. Exit
Your choice: 1
Enter file or directory name: Sub_Folder_01
... Change to directory Sub_Folder_01

=====
=====
*----- NTFS EXPLORER -----
*----- Choose an option:
1. Read file or directory
2. Back to parent directory
3. Print current root directory
0. Exit
Your choice: 3
... Printing current root directory: Sub_Folder_01
|Archive        | Text.txt
=====
=====
```

Tại đây, nếu ta chọn “2. Back to parent directory”, ta được chuyển hướng đến thư mục cha của Sub\_Folder\_01 là Parent\_Folder\_01 (đoạn sau là minh chứng chương trình đang ở thư mục Parent\_Folder\_01)

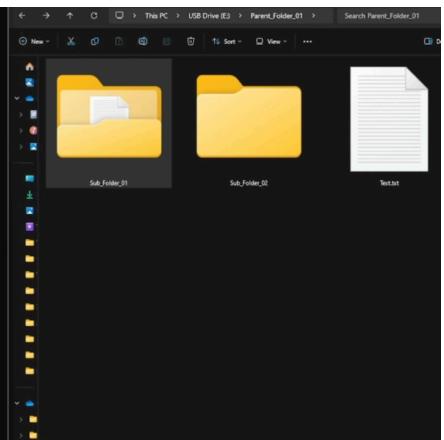
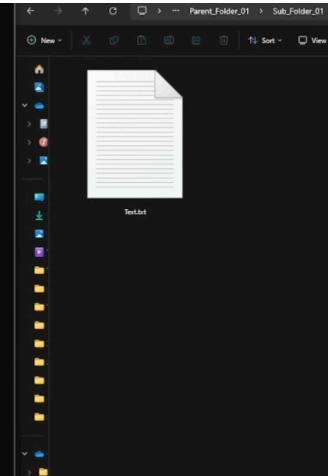
```
3. Print current root directory
0. Exit
Your choice: 3
... Printing current root directory: Sub_Folder_01
    |Archive          | Text.txt
=====
=====
=====
*-----
*----- NTFS EXPLORER -----
*-----

Choose an option:
1. Read file or directory
2. Back to parent directory
3. Print current root directory
0. Exit
Your choice: 2
... Back to parent directory: Parent_Folder_01

Your choice: 2
... Back to parent directory: Parent_Folder_01
=====
=====
=====

*-----
*----- NTFS EXPLORER -----
*-----

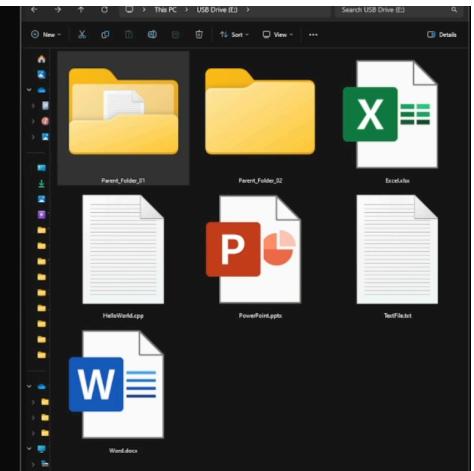
Choose an option:
1. Read file or directory
2. Back to parent directory
3. Print current root directory
0. Exit
Your choice: 3
... Printing current root directory: Parent_Folder_01
    |Directory      | Sub_Folder_01
    |Directory      | Sub_Folder_02
```



Tại Parent\_Folder\_01, nếu ta tiếp tục yêu cầu quay về thư mục cha, chương trình sẽ chuyển hướng đến thư mục gốc “.”, tại đây nếu ta tiếp tục yêu cầu quay về thì chương trình sẽ thông báo “Bạn đang ở thư mục gốc”

```
|Archive      | TextFile.txt
|Archive      | PowerPoint.pptx
|Archive      | Word.docx
|Archive      | Excel.xlsx
|Archive      | HelloWorld.cpp

=====
=====
=====
*----- NTFS EXPLORER -----
*----- Choose an option:
1. Read file or directory
2. Back to parent directory
3. Print current root directory
0. Exit
Your choice: 2
You are in the root directory
=====
```



## VI.Nguồn tham khảo

- [1]. *Ths. Lê Viết Long*, “Slide bài giảng Hệ điều hành 22CLC06 Tiếng Việt - Bài 3: Hệ thống tập tin FAT”
- [2]. *Ths. Lê Viết Long*, “Giải đáp thắc mắc Project 1 - Hệ điều hành 22CLC06”
- [3]. *NTFS overview - NTFS.com*. (n.d.). [https://www.ntfs.com/ntfs\\_basics.htm](https://www.ntfs.com/ntfs_basics.htm)
- [4]. *Active@ File Recovery: Defining cluster chains on NTFS*. (n.d.).  
<https://www.file-recovery.com/recovery-define-clusters-chain-ntfs.htm>
- [5]. *Active@ Disk Editor - Freeware Hex Viewer & Hex Editor for raw sectors/clusters*. (n.d.). <https://www.disk-editor.org/index.html>